
**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**

**CENTRO TECNOLÓGICO**

**PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

Jacson Rodrigues Correia da Silva

# Copycat CNN: Convolutional Neural Network Extraction Attack with Unlabeled Natural Images

Vitória, ES

2023

Jacson Rodrigues Correia da Silva

# Copycat CNN: Convolutional Neural Network Extraction Attack with Unlabeled Natural Images

Tese de Doutorado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Doutor em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Supervisor: Prof. Dr. Thiago Oliveira dos Santos

Co-supervisor: Prof. Dr. Alberto Ferreira de Souza

Vitória, ES

2023

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Correia da Silva, Jacson Rodrigues, 1985-
C824c Copycat CNN : Convolutional Neural Network Extraction Attack with Unlabeled Natural Images / Jacson Rodrigues Correia da Silva. - 2023.
174 f. : il.

Orientador: Thiago Oliveira dos Santos.
Coorientador: Alberto Ferreira De Souza.
Tese (Doutorado em Ciência da Computação) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Inteligência artificial. 2. Redes neurais (Computação). I. Oliveira dos Santos, Thiago. II. De Souza, Alberto Ferreira. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

---

# *Copycat CNN: Convolutional Neural Network Extraction Attack with Unlabeled Natural Images*

***Jacson Rodrigues Correia da Silva***

Tese de Doutorado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Aprovada em 25 de Abril de 2023.

Prof. Dr. Thiago Oliveira dos Santos
Orientador

Prof. Dr. Claudine Santos Badue Gonçalves
Membro Interno

Prof. Dr. Thomas Walter Rauber
Membro Interno

Prof. Dr. Jurandy Gomes de Almeida Junior
Membro Externo, participação remota

Prof. Dr. Eduardo José da Silva Luz
Membro Externo, participação remota

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória/ES, 25 de Abril de 2023

Documento assinado digitalmente conforme descrito no(s) Protocolo(s) de Assinatura constante(s) neste arquivo, de onde é possível verificar a autenticidade do mesmo.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

**PROTOCOLO DE ASSINATURA**

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
THIAGO OLIVEIRA DOS SANTOS - SIAPE 2023810
Departamento de Informática - DI/CT
Em 26/04/2023 às 09:32

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/698633?tipoArquivo=O

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
THOMAS WALTER RAUBER - SIAPE 2201072
Departamento de Informática - DI/CT
Em 26/04/2023 às 11:47

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/698826?tipoArquivo=O

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
CLAUDINE SANTOS BADUE - SIAPE 1729561
Departamento de Informática - DI/CT
Em 26/04/2023 às 17:15

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/699199?tipoArquivo=O

# Acknowledgements

There are many moments of struggle and battles that seem to have no end. Everyone's life has many obstacles, which can become lighter or be overcome thanks to the direct and indirect support of the people around us. At this moment, I thank God, who has always been by my side through life's coincidences, manifested through the hands of the people around me, even those who may not believe in Him. He has given me strength when I needed it and guided my thoughts, enabling me to move forward.

I am very grateful to my wife, who has been with me throughout this entire journey, providing me with strong support, affection, and love. Moreover, I am grateful to my son, who has brought a new perspective to my world. Understanding his mind can be challenging at times, but through him, I have gained a deeper understanding of myself. I love you and I understand, by your gestures, how much you love me too.

Many thanks to my parents, who have always been there for me and my family, offering their unwavering presence and support when we needed them most. I also want to express my gratitude to my sisters and brothers-in-law, who have given me their time, attention, affection, and support, whether they are near or far.

I am extremely grateful to my supervisor, who consistently demonstrated their willingness to guide and listen to me. He always made intelligent choices and approached situations with an open mind, charting new paths and guiding me through the difficulties encountered on this journey. I would also like to express my heartfelt thanks to the team at NXP Semiconductors, who provided me with invaluable life experiences and several opportunities for growth and learning.

Finally, I also want to express my gratitude to everyone who has helped me directly and indirectly during this process, thanks to NVIDIA for providing me a GPU to be used in this research, and to the AWS Cloud Credit for Research program, which provided me cloud computing resources.

# Resumo

Redes Neurais Convolucionais (CNNs) têm alcançado alto desempenho em vários problemas nos últimos anos, levando muitas empresas a desenvolverem produtos com redes neurais que exigem altos custos para aquisição de dados, anotação e geração de modelos. Como medida de proteção, as empresas costumam entregar seus modelos como caixas-pretas acessíveis apenas por APIs, que devem ser seguras, robustas e confiáveis em diferentes domínios de problemas. No entanto, estudos recentes mostraram que CNNs estado-da-arte têm vulnerabilidades, onde perturbações simples nas imagens de entrada podem mudar as respostas do modelo, e até mesmo imagens irreconhecíveis por humanos podem alcançar uma predição com alto grau de confiança do modelo. Esses métodos precisam acessar os parâmetros do modelo, mas há estudos mostrando como gerar uma cópia (imitação) de um modelo usando suas probabilidades (*soft-labels*) e dados do domínio do problema. Com um modelo substituto, um adversário pode efetuar ataques ao modelo alvo com maior possibilidade de sucesso. Este trabalho explora ainda mais essas vulnerabilidades. A hipótese é que usando imagens publicamente disponíveis (que todos tem acesso) e respostas que qualquer modelo deve fornecer (mesmo caixa-preta) é possível copiar um modelo atingindo alto desempenho. Por isso, foi proposto um método chamado Copycat para explorar modelos de classificação de CNN. O objetivo principal foi copiar o modelo em duas etapas: primeiro, consultando-o com imagens naturais aleatórias, como do ImageNet, e anotando suas probabilidades máximas (*hard-labels*). Depois, usando essas imagens rotuladas para treinar um modelo Copycat que deve alcançar desempenho semelhante ao modelo alvo. Avaliamos essa hipótese em sete problemas do mundo real e contra uma API baseada em nuvem, atingindo desempenhos (F1-Score) em todos modelos Copycat acima de 96,4% quando comparados aos modelos alvo. Após atingir esses resultados, realizamos vários experimentos para consolidar e avaliar nosso método. Além disso, preocupados com essa vulnerabilidade, também analisamos várias defesas existentes contra o método Copycat. Dentre os experimentos, as defesas que detectam consultas de ataque não funcionam contra o método, mas defesas que usam marca d'água conseguem identificar a Propriedade Intelectual do modelo alvo. Assim, o método se mostrou eficaz na extração de modelos, possuindo imunidade às defesas da literatura, sendo identificado apenas por defesas de marca d'água.

**Palavras-chaves**: Aprendizado Profundo. Redes Neurais Convolucionais. Roubo de Conhecimento de Redes Neurais. Destilação de Conhecimento. Extração de Modelo. Roubo de Modelo. Compressão de Modelo.

# Abstract

Convolutional Neural Networks (CNNs) have been achieving state-of-the-art performance on a variety of problems in recent years, leading to many companies developing neural-based products that require expensive data acquisition, annotation, and model generation. To protect their models from being copied or attacked, companies often deliver them as black-boxes only accessible through APIs, that must be secure, robust, and reliable across different problem domains. However, recent studies have shown that state-of-the-art CNNs have vulnerabilities, where simple perturbations in input images can change the model's response, and even images unrecognizable to humans can achieve a higher level of confidence in the model's output. These methods need to access the model parameters, but there are studies showing how to generate a copy (imitation) of a model using its probabilities (soft-labels) and problem domain data. By using the surrogate model, an adversary can perform attacks on the target model with a higher possibility of success. We further explored these vulnerabilities. Our hypothesis is that by using publicly available images (accessible to everyone) and responses that any model should provide (even black-boxes), it is possible to copy a model achieving high performance. Therefore, we proposed a method called Copycat to explore CNN classification models. Our main goal is to copy the model in two stages: first, by querying it with random natural images, such as those from ImageNet, and annotating its maximum probabilities (hard-labels). Then, using these labeled images to train a Copycat model that should achieve similar performance to the target model. We evaluated this hypothesis on seven real-world problems and against a cloud-based API. All Copycat models achieved performance (F1-Score) above 96.4% when compared to target models. After achieving these results, we performed several experiments to consolidate and evaluate our method. Furthermore, concerned about such vulnerability, we also analyzed various existing defenses against the Copycat method. Among the experiments, defenses that detect attack queries do not work against our method, but defenses that use watermarking can identify the target model's Intellectual Property. Thus, the method proved to be effective in model extraction, having immunity to the literature defenses, but being identified only by watermark defenses.

**Keywords**: Deep learning. Convolutional neural network. Stealing network knowledge. Knowledge distillation. Model extraction. Model Stealing. Model compression.

# List of Figures

# List of Tables

# Acronyms and Abbreviations

**General:**

**API** Application Programming Interface

**CNN** Convolutional Neural Network

**DNN** Deep Neural Network

**DL** Deep Learning

**FGSM** Fast Gradient Sign Method

**FEA** Feature Extraction Algorithm

**GAN** Generative Adversarial Network

**ILSVRC** ImageNet Large Scale Visual Recognition Challenge

**IP** Intellectual Property

**JBDA** Jacobian-Based Dataset Augmentation

**LRP** Layer-wise Relevance Propagation

**MSP** Maximum Softmax Probabilities

**ML** Machine Learning

**MLP** Multi Layer Perceptron

**MLaaS** Machine Learning as a Service

**RBF** Radial Basis Function

**RL** Representation Learning

**SGD** Stochastic Gradient Descent

**TN** Target Network

**SNNS** Soft Nearest Neighbor Loss

**t-SNE** t-Distributed Stochastic Neighbor Embedding

**XAI** Explainable Artificial Intelligence

**Problems:**

**ACT101** Human Action Classification Problem, 101 Categories

**DIG10** Handwritten Digit Classification, 10 categories

**FER7** Facial Expression Recognition, 7 categories

**GOC9** General Object Detection, 9 categories

**PED2** Pedestrian Detection, 2 categories

**SHN10** Street View House Number Classification, 10 categories

**SIG30** Traffic Sign Classification, 30 categories

**Models:**

**BL** Baseline model

**BL-Alex-*** Baseline model generated on AlexNet Architecture

**BL-VGG-*** Baseline model generated on VGG-16 Architecture

**CC** Copycat model

**CC-Alex-*** Copycat model generated on AlexNet Architecture

**CC-VGG-*** Copycat model generated on VGG-16 Architecture

**Datasets:**

**ODD** Original Domain Dataset

**ODD-OL** Original Domain Dataset with Original Labels

**PDD** Problem Domain Dataset

**PDD-OL** Problem Domain Dataset with Original Labels

**PDD-SL** Problem Domain Dataset with Stolen Labels

**TDD** Test Domain Dataset

**TDD-OL** Test Domain Dataset with Original Labels

**NPDD** Non-Problem Domain Dataset

**NPDD-SL** Non-Problem Domain Dataset with Stolen Labels

**NPDD+PDD** Non-Problem Domain Dataset joined to Problem Domain Dataset

**NPDD+PDD-SL** Non-Problem Domain Dataset joined to Problem Domain Dataset, both with Stolen Labels

**OL** Original Labels

***-OL** Suffix of Original Labels Dataset

**SL** Stolen Labels

***-SL** Suffix of Stolen Labels Dataset

# Contents

# 1 Introduction

Currently, there are several smart tools and robots that use Machine Learning (ML) techniques at their core. These products, such as Echo with Alexa[1a], Google Home, Nest or Android with Google Assistant[1b], and HomePod with Siri[1c] (Figure 1), can process and provide responses to commands and requests in natural language, can recognize persons and objects, and also perform other tasks, that makes them convenient for users. Additionally, many of these products also have access to cloud-based APIs, which allow integration with other platforms and ML services (Machine Learning as a Service – MLaaS). These MLaaS, such as Microsoft Cognitive Services Computer Vision API[1d], Google Cloud Vision API[1e], IBM Watson Visual Recognition[1f] (Figure 1) provide access to powerful machine learning models and algorithms, which can extract useful information from the user data to process the request. Among these systems, there are the Representation Learning (RL) methods that are built to also learn representations of the data, thus simplifying the extraction of useful information for classifiers and predictors (BENGIO; COURVILLE; VINCENT, 2013).


Figure 1 – Examples of smart devices on the left and APIs on the right (image sources: i, ii, iii, iv, v, vi, vii, viii – *source links are available in the digital version of this document*).

An important method in RL is a DNN[2] called Convolutional Neural Network (CNN) (LECUN et al., 1989b). It has one or more convolutional layers that uses a mathematical operation called convolution to analyze the input data. And a Deep CNN is usually made

[1] a, b, c, d, e, f. *Source links are available in the digital version of this document.*

[2] Deep Neural Network is a multilayer feedforward network. But unlike shallow networks that only have a few layers, the depth of this network comes from the total length of its chain of layers, hence the name DNN (GOODFELLOW; BENGIO; COURVILLE, 2016).

up of multiple layers (Figure 2), starting with convolutional layers, pooling layers, and ending with fully connected layers[3]. The intent of these first layers is to identify and extract features in the input data, such as edges, shapes, and textures and provide these feature maps to the next layers. Then, the fully connected layers are used to make the final prediction using that features.

Given the state-of-the-art performance achieved by CNNs on a variety of problems, these networks have been the power of many neural-based systems to recognize facial expressions, traffic signs, product names, objects, speech, and several other tasks. Consequently, many companies are investing a large amount of money to generate products with CNNs. However, there are at least three high costs associated with it:

i. acquiring and annotating large-scale training datasets;
ii. computational power for training the models[4] that can last for days or even months; and
iii. experts to prepare the data and to design, implement, and train the model.

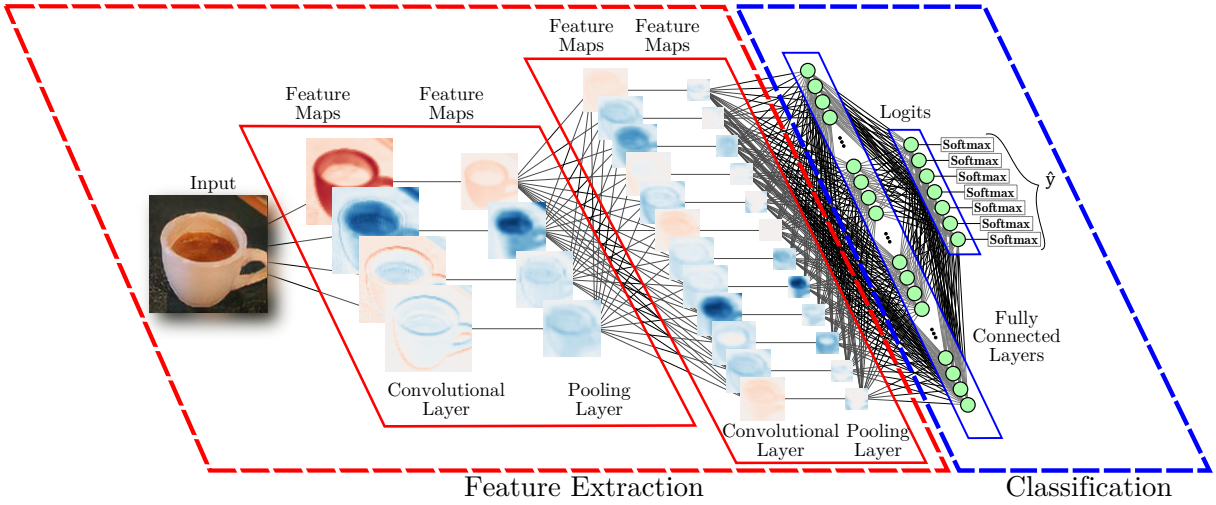



Figure 2 – Illustration of a Convolutional Neural Network for classification. In Feature Extraction, the input image is fed to the network, which will use Convolutional Layers (usually followed by an activation function like ReLU) and Pooling Layers to extract its features and send to the Fully Connected Layers. Then, in Classification, the final prediction $\hat{y}$ is calculated by applying the Softmax function to the outputs of the last Fully Connected Layer. The values before applying the Softmax are usually called logits in the context of DNN.

Therefore, due to the resources and money invested in creating these models, it is in the best interests of these companies to protect their model's Intellectual Property (IP)

[3] Although the acronym MLP – Multilayer Perceptron – refers to a Neural Network with fully connected layers, the term Fully Connected Layers is more popular in the Deep Learning area and is widely used to refer to the final layer of a Deep Neural Network.

[4] An ML model is an architecture, such as CNN, with its parameters already adjusted by a training process.

(*i. e.*, model parameters, patented algorithms, unique datasets, and other copyrighted assets) against attacks or copies. When another company, for example, lacks resources but needs a more accurate model, they can gain a competitive advantage by copying a competing model. Or if someone needs, for example, a copy[5] of the target model to access its parameters to do other tasks, like tricking malware and spam detection, or to mislead autonomous navigation systems (PAPERNOT; MCDANIEL; GOODFELLOW, 2016). In the first case, the objective is to achieve the same accuracy (number of correct predictions) of the target model and, in the second case, the objective is to achieve the fidelity (similarity between the models' responses) of the target model (JAGIELSKI et al., 2020). For that and other reasons, their models are usually not delivered as white-boxes (with parameters, architecture and other details visible to users). In fact, the models are usually provided as a black-box, where it is possible only to send the input data and receive the probabilities[6] of the model (soft-labels) or the label (highest probability output) referring to the predicted class (hard-label).

Examples of these black-box models are often available, as shown at the beginning of this chapter, in smart tools, robots, or in the cloud as MLaaS Application Programming Interfaces (APIs). In this scenario, note that a user who owns a robot can query it for free an unlimited number of times, and the one who needs to use it as MLaaS must pay for each usage. Users and also developers expect that these models provide accurate and robust generalizations for new queries. For instance, in an object recognition model, new images in the problem domain might be correctly identified, and it is expected that small perturbations in the input images should not change the object's classification. And this is exactly where an attack can occur and the model knowledge can be extracted to produce a surrogate (substitute) model. This process (attack) is called model extraction.

## 1.1 Model extraction and attacks

Model extraction consists of labeling a dataset with the target model responses (soft-labels or hard-labels) and using this dataset to train a surrogate (substitute) model. Some studies in this area worked on extracting knowledge from a larger model to a smaller one, with fewer parameters, calling this process of model compression. Bucilă, Caruana and Niculescu-Mizil (2006) presented a method for extracting large, complex ensembles (set of smaller models combined to achieve performance equivalent to a larger, more complex model) into smaller models without lost significant performance. In addition, Ba and Caruana (2014) explored transferring a deep neural network to a shallow neural

[5] The term copy will be used in the sense of imitating a model and not in the sense of producing an exact copy of it.

[6] The term "probabilities" is commonly used in the literature to refer to the network output after applying the Softmax operation.

network. The dataset for training the substitute model was composed of images from the problem domain labeled by the logits[7] of the target model. Based on these studies of model compression, Hinton, Vinyals and Dean (2015) observed that the outputs (probabilities after Softmax function) of a model provide important information about the generalization of the input data. So they proposed a method called knowledge distillation, where instead of using logits, they modified the Softmax function (Equation 3.2). Their intention was to smooth out the model's output probabilities and provide better information about its internal knowledge. For this type of model extraction, they also used original and synthetic data from the problem domain.

Other important works investigated ways to attack machine learning models. Szegedy et al. (2014) explored some vulnerabilities in neural networks and showed that, using a simple optimization procedure, it is possible to find adversarial examples, *i. e.*, by adding small perturbations to an image already classified correctly by the neural network, it is possible to generate a new image that the network is no longer able to classify correctly (Figure 3). Later, Goodfellow, Shlens and Szegedy (2015) proposed the fast gradient sign method, which uses the signal of the cost function's gradient to create adversarial examples against DNNs (Figure 4). After, Papernot et al. (2016) used the forward derivatives for building adversarial saliency maps[8] to craft adversarial samples against the DNN model. These methods need to access the model parameters (white-box) and also need problem domain data.

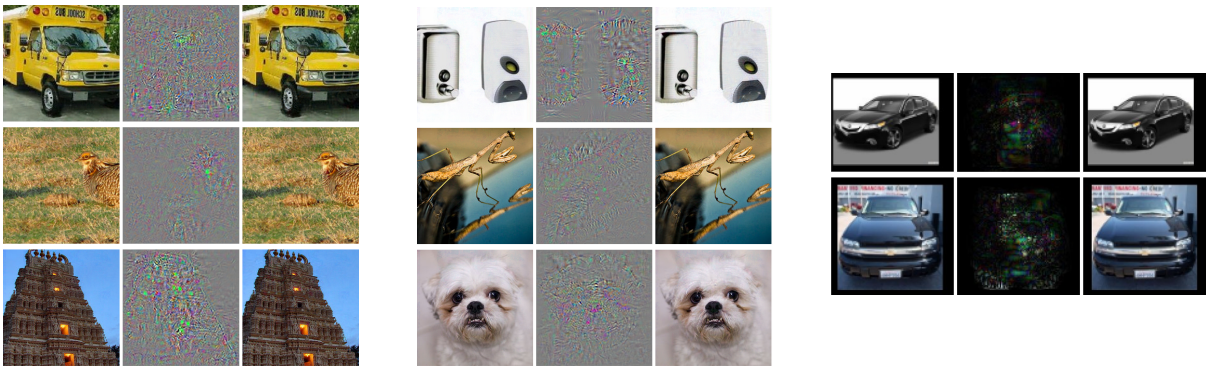
After these works, Papernot et al. (2017) formulated a new strategy capable of crafting adversarial examples against black-box models provided as MLaaS. The first step is to extract the target model using problem domain images. Then, this surrogate model is used to generate the adversarial examples using the two methods cited in the previous paragraph. At each generation of new examples, the target model is consulted, generating new labels that are used to improve the performance of the substitute model. However, the objective was not the model extraction, but only to generate a surrogate model to craft adversarial examples (Figure 4).

On the other hand, Nguyen, Yosinski and Clune (2015) studied the vulnerability presented by Szegedy et al. (2014) and found that it is easy to generate images that are completely unrecognizable to humans, but that DNNs recognize with an higher confidence (Figure 5). They used evolutionary algorithms and gradient ascent to find images with a high degree of confidence[9] belonging to each class in the tested DNN models. Additionally, the authors also tried to avoid this misclassification by adding a garbage class, but without

[7] The term logits is commonly used to refer to the the raw outputs of the network before the application of the Softmax function, more details in Section 2.1.

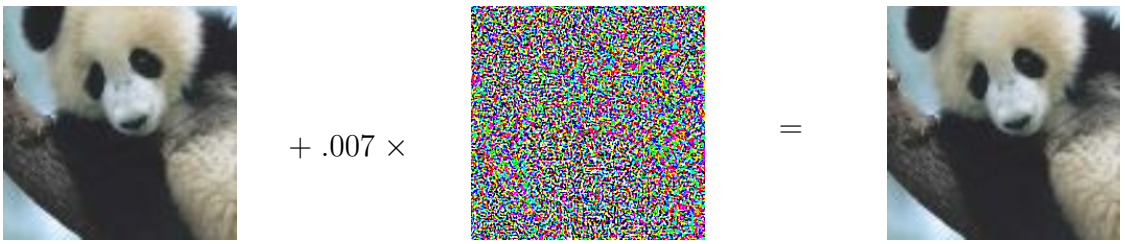
[8] The saliency map is a measure of how much each input pixel contributes to the final predictions of the network.

[9] High confidence means that the highest probability of the model is a value close to 100%



(a) (b) (c)

Figure 3 – Adversarial examples generated in (SZEGEDY et al., 2014). In (a) and (b), the left images are the correctly predicted sample, the center images are the difference between correct image and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), and the right images are the adversarial example classified as "ostrich, Struthio camelus". In (c), results are presented for a binary car classifier. The images on the left are recognized as cars and the images on the right are not recognized. The center images are the magnified absolute value of the difference between the two images.



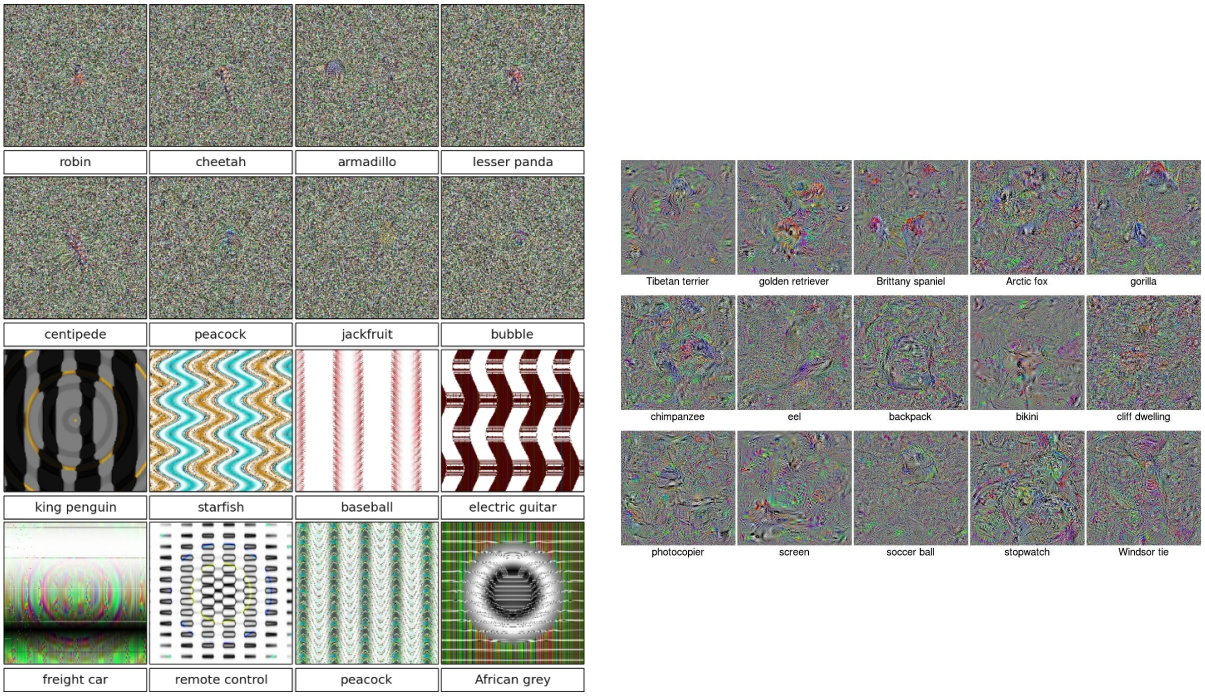

$x$
"panda"
57.7% of confidence

$sgn(\nabla_x J(\theta, x, y))$
"nematode"
8.2% of confidence

$x + \lambda sgn(\nabla_x J(\theta, x, y))$
"gibbon"
99.3% of confidence

Figure 4 – A demonstration of an adversarial example generated by the fast sign method. Let $\theta$ be the parameters of a model, $x$ the input to the model, $y$ the desired prediction of $x$ and $J(\theta; x; y)$ the cost used to train the neural network. By adding the imperceptibly small vector (center image) provided by the gradient of the cost function with respect to the input, the network classification was changed to a "gibbon". The $\lambda = .007$ corresponds to the magnitude of the smallest bit of an 8 bit image encoding after model conversion to real numbers (GOODFELLOW; SHLENS; SZEGEDY, 2015).

success.




(a) Images obtained by evolutionary algorithms (b) Images obtained by gradient ascent

Figure 5 – Images provided by (NGUYEN; YOSINSKI; CLUNE, 2015) that are unrecognizable to humans, but that deceived state-of-the-art DNNs with high confidence.

In parallel with these works, Deng et al. (2009) released a large publicly available image dataset called ImageNet, which provided 3.2 million images to be used in image recognition systems. And the following year, Russakovsky et al. (2015a) started the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which has become one of the most important competitions in the area. And later, another large dataset was also released, the Microsoft COCO, published in (LIN et al., 2014), with 2.5 million labeled images. Note that even with the existence of these datasets, which have millions of random images of different classes of data, the related works used only original and synthetic data from the problem domain. Additionally, the most part of them used the target model as a white-box, *i. e.*, accessing their parameters, instead of using them as a black-box.

## 1.2 Hypothesis

As presented so far, the cited works have shown that the information provided in the responses of the target models allows extracting their knowledge, *i. e.*, they showed that model extraction attacks can be done through logits, soft-labels and hard-labels to generate surrogate models. In these works, they also showed that the information of the target models can be explored using synthetic data, but for that purpose it was necessary to have data from the problem domain of the attacked model. Moreover, they shown that

even images not recognizable by humans can provide a high confidence responses in DNN models. Additionally, their works suggest that these images belong to the feature or/and the classification space of the model and can provide important information about it.

Thereby, our hypothesis is that:

*Model extraction can be performed using random natural images provided in large public image datasets (such as ImageNet and Microsoft COCO) and only the hard-labels of the model, even if the images are not from the problem domain of the attacked model.*

Furthermore, we believe that the generated surrogate model can still be fine-tuned and achieve better accuracy and fidelity when the adversary has data from the problem domain. We limited the scope of this research on image classification using CNN models.

## 1.3 Objectives

To prove our hypothesis, we proposed as a main objective a model extraction method, called Copycat, which was tested in CNNs for image classification. In the Copycat method[10], the target model, *i. e.*, the Oracle $f(\cdot)$, is queried with random non-labeled images acquired from large public datasets, such as ImageNet or Microsoft COCO. The pairs of images and their hard-labels provided by the Oracle are used to generate a fake dataset $NPDD = \{(x_1, \ell_{x_1}), \ldots, (x_n, \ell_{x_n})\}$, where $n = |NPDD|$, $x$ is the image and $\ell_x = \arg\max_p f(x)[p]$ is the hard-label provided by the Oracle. Then the surrogate model, named Copycat model, is trained with $NPDD$. After, a problem domain dataset is labeled by the Oracle to generate the second dataset $PDD = \{(x_1, \ell_{x_1}), \ldots, (x_m, \ell_{x_m})\}$, where $m = |PDD|$, that is used to fine-tune the Copycat model.

To achieve the main objective, we explored two major topics in this study: attacks with the Copycat method and defenses against it. For this, our secondary objectives for evaluating the attacks with the Copycat were:

- analysis of model extraction with the Copycat method on seven different problems and a real-world MLaaS API;
- verification of the capabilities, limitations and robustness of the Copycat method;
- quantitative and qualitative comparison between Oracles and Copycat models.

And our secondary objectives for evaluating the defenses against the Copycat method were:

- analysis of methods that detect out-of-distribution queries; and
- analysis of a watermark method that protects the model's IP.

[10] The reader can access an interactive visualization of a model extraction attack with the Copycat method at <https://jeiks.github.io/copycat-cnn-explainer/>

## 1.4 Publications

Up to the present time, the developing of this project has contributed with the literature in significant ways. The contributions were published in two articles:

i. Correia-Silva, J. R.; Berriel, R. F.; Badue, C.; De Souza, A. F.; Oliveira-Santos, T. Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data. In: *International Joint Conference on Neural Networks (IJCNN)*. 2018. p. 1–8. (Qualis A1).
ii. Correia-Silva, J. R.; Berriel, R. F.; Badue, C.; De Souza, A. F.; Oliveira-Santos, T. Copycat CNN: Are random non-Labeled data enough to steal knowledge from black-box models? *Pattern Recognition*, v. 113, p. 107830, 2021. ISSN 0031-3203. (Qualis A1).

The first work (Correia-Silva et al., 2018) presented the Copycat method, the results of primary three problems (facial expression, object, and crosswalk classification), and the model extraction results against the Microsoft Azure Cognitive Services[11]. To the best of our knowledge, this was the first time that the black-box model extraction in a deep CNN has been investigated using out-of-distribution images obtained from public datasets and labeled only by hard-labels.

The second work (Correia-Silva et al., 2021) consolidated the Copycat method by performing an extensive evaluation study aiming at providing a better understanding of the behavior of the Copycat model on black-box attacks. To achieve this purpose, it was composed of the following experiments: (i) extraction with less information about the target black-box model; (ii) feature space analysis of out-of-distribution images; (iii) analysis of the number of queries to extract a model; (iv) model extraction for a slightly different but small architecture; (v) robustness of the Copycat method; (vi) comparison between Oracle and surrogate model based on their inputs; (vii) discussion attack cost to make a real extraction on a MLaaS.

The results and analysis regarding the defenses against the Copycat method have not yet been published.

## 1.5 Outline

The remainder of the text is organized as follows:

- The theoretical background is described in Chapter 2, covering Convolutional Neural Networks, large datasets, and methods for analyzing neural networks.

[11] The Microsoft Azure Cognitive Services: <https://azure.microsoft.com/en-us/pricing/details/cognitive-services/emotion-api/>

- Chapter 3 presents an overview of the related works, covering techniques for model extraction and attacks, as well as defense approaches against such attacks.
- In Chapter 4, the Copycat method formulation is presented, followed by a discussion of relevant constraints. Additionally, an illustrative example of a simple model extraction using the Copycat method is provided.
- Chapter 5 explains our experimental methodology, starting with details about the datasets, the investigated problems and architectures, and the metrics. Next, the model extraction experiments using the Copycat method are described. These experiments include: (i) analysis of dataset distributions in the black-box model's classification space, (ii) application of the Copycat method on the same architecture to several different problems, (iii) analysis of the number of images needed to successfully attack the proposed problems with the Copycat method, (iv) application of the Copycat method in a slightly different and small architecture, (v) verification of the reproducibility and robustness of the Copycat model, (vi) comparison of the target model and the Copycat model in relation to the input image relevance pixels, and (vii) analysis of a Copycat attack in a real-world API and discussion of the viability and costs of such attacks. Finally, three defense approaches explored in our work are presented.
- Chapter 6 presents the results of the experiments described in Chapter 5 and also discusses the limitations of our method.
- Finally, Chapter 7 draws the concluding remarks and present the future directions of our research.

# 2 Theoretical Background

This chapter presents the necessary theoretical concepts to contextualize the scope of our work. The objective is to provide base knowledge on the subject, presenting the concepts that underlie the Copycat method. Therefore, this chapter is divided into two sections. As our method works on extracting CNN models using large image databases, the first section presents a description of CNN and related model architectures and databases. After, as we did the qualitative analysis of the surrogate models, second section presents the respective methods used in our work. Additionally, in the next chapter, we provide an overview of related works and compare them with our proposed method.

## 2.1 Convolutional Neural Networks and Large Public Databases

According to Goodfellow, Bengio and Courville (2016), computer programs have traditionally been used to solve problems that can be described using a set of formal mathematical rules. Thus, these programs rely on pre-coded knowledge within their code. However, given the difficulty of coding them for more complex problems (as high-dimensional problems), it was sought that the programs themselves had the ability to acquire knowledge on their own, identifying patterns in received data. This ability is commonly called Machine Learning. The use of these methods allows computers to solve more complex problems and find solutions that seem subjective. However, the performance of these machine learning algorithms depends on the representation of the data provided to them. Thus, manual work is usually required to provide information contained in the representation, called features, for these programs to produce useful responses.

Increasing the difficulty of these problems, in some cases, ideal results are achieved only with specific features, which are often difficult to obtain or identify. In these cases, a solution would be for machine learning algorithms to learn not only to give the answers to ready-made representation, but also to learn the representation of the problem. This field is known as Representation Learning, and it is used in ML to provide better performance than previous methods with hand-designed representations. In this field, the initial challenge is to extract enough features from the raw data. For this purpose, a solution adopted is to create a deep hierarchy capable of extracting simple features until reaching complicated concepts. This method is commonly referred to as Deep Learning (DL), which achieved human-like abilities to recognize objects or speech.

An example of a DL method can be a kind of feedforward deep network, or Multi Layer Perceptron (MLP), which is a mathematical function that maps input values to output values. This function is formed by the composition of several simpler functions,

which provide a new representation of the input data. An MLP network can be a shallow network when it is formed by few layers, or contain a deep amount of layers, forming a Deep Neural Network. However, an MLP has some limitations in image processing. Let an image $X_{m,n}$ be a matrix of pixels composed of $m$ rows and $n$ columns:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \tag{2.1}$$

where $x_{i,j}$ represents the pixel value in row $i$ and column $j$. In MLP, this matrix is flattened into a vector of numbers, $X = [x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn}]$. This format, for example, does not fully capture the spatial relationships between pixels in an image.

Differently of MLP, Convolutional Neural Network (LECUN; BENGIO, 1995) is a DNN that can treat the image as a matrix $\mathbb{R}^2$ (considering only one channel, like a grayscale image) instead of an vector $\mathbb{R}^1$. CNNs exploit the spatial invariance that objects are in the image, seeking to learn useful representations. This type of network has the following principles: (i) translation invariance (or translation equivariance), the earliest layers of the network must extract from the image the same characteristics referring to a patch of interest, regardless of where it appears in the image, (ii) locality, the earliest layers of the network must extract features from local regions, without taking into account distant regions of the image. Eventually, these simple representations can be aggregated to form concepts of the whole picture. Following these principles, deeper layers should be able to capture longer-range features of the image (ZHANG et al., 2021).

The CNN can be made up of multiple convolutional layers, pooling layers, and fully connected layers. It uses a mathematical operation called convolution to process the input data. This operation handles the principles of translation invariance and locality (for more details, please look at (ZHANG et al., 2021)). The discrete convolution function is:

$$(K * X)(i, j) = \sum_k \sum_l X(i - k, j - l) K(k, l) \tag{2.2}$$

where $K$ is the kernel (or filter) with $k$ rows and $l$ columns. However, to avoid the matrix flipping performed when multiplying the image by the kernel, libraries often implement the cross-correlation instead of convolution (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$(K * X)(i, j) = \sum_k \sum_l X(i + k, j + l) K(k, l) \tag{2.3}$$

In ML, the algorithm will learn the appropriate values of the kernel. So it will learn a flipped kernel relative to the convolution kernel, which will not change the final

result in the network (GOODFELLOW; BENGIO; COURVILLE, 2016). The output of the convolutional layer is called feature map, as it can be considered as the learned representations (features) in spatial dimensions for the subsequent layer. An illustrative example of a cross-correlation (convolution without kernel flipping) between two 2-D matrices is presented on Figure 6. Note that the output matrix is not the same size as the input matrix due to the convolution operation. To avoid this, pixel paddings can be used around the input matrix. Moreover, another factor that can affect the size of the output is the size of the Kernel stride (in the Figure 6, the stride was one).

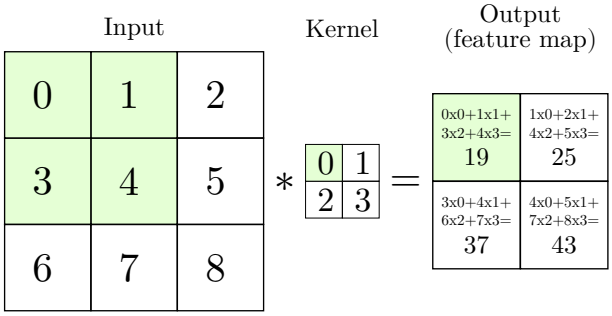



Figure 6 – An example of 2-D convolution without kernel flipping (cross-correlation). The shaded portions are the first output element, as well as the input and kernel elements used for the output operation: $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$.

After a convolutional layer, an activation function such as ReLU can be applied to add non-linearity to the network. Then, a pooling layer also can be used to mitigate the sensitivity of the convolutional layer and to reduce the spatial resolution of the representations. The pooling layer traverses fixed-size windows of pixels over the convolution output matrix in a defined stride number, performing a single operation between these pixels, such as their average or maximum value. An example of maximum pooling (often called max-pooling) is showed on Figure 7.

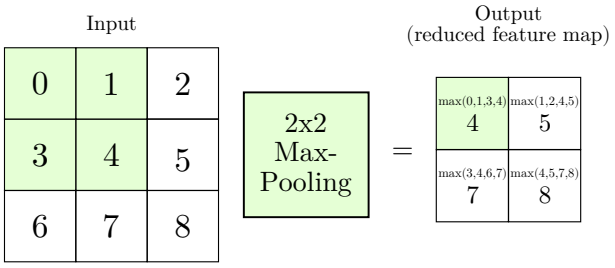



Figure 7 – An example of maximum pooling (generally called max-pooling) with a window size of 2x2. The highlighted portions are the first output element, as well as the input elements used for the output operation: $max(0, 1, 3, 4) = 4$.

So, after the feature maps are generated by the previous layers, they are flattened

into a single column vector and provided as inputs to the first Fully Connected layer:

$$z_j = \sigma\left(\sum_{i=1}^{n} \mathbf{W}_{ji}\mathbf{r}_i + b_j\right) \tag{2.4}$$

where $\mathbf{z} = \{z_j\}_{j=1}^{d}$ is the output vector and $d$ is the total number of possible classes, $\mathbf{r} = \{r_i\}_{i=1}^{n}$ is the feature vector of size $(n, 1)$, $\mathbf{W}$ is the matrix of weights of size $(d, n)$, $\mathbf{b} = \{b_j\}_{j=1}^{d}$ is the bias vector, and $\sigma$ is the activation function applied element-wise to the output vector $\mathbf{z}$.

These results can go through several Fully Connected layers, but in this example we are only considering one. Finally, these non-normalized output values produced by the network are converted into a probability distribution by the normalized exponential function Softmax:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{d} e^{z_k}} \tag{2.5}$$

where $\hat{y}_j$ is the predicted probability of the input belonging to the $j$-th class.

The values before being converted by the Softmax function are commonly called logits. In statistics, the logits are the logarithm of the odds or log-odds. And in the logistic regression, the logits are the inverse of the Logistic Sigmoid function $\phi(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}}$, *i. e.*, $\text{logit}(\phi(\mathbf{z})) = log\left(\frac{\phi(\mathbf{z})}{1-\phi(\mathbf{z})}\right)$ (HASTIE; TIBSHIRANI; FRIEDMAN, 2009). In DNN, the ReLU (NAIR; HINTON, 2010) function (or a similar one) is commonly used as the activation function. Therefore, the concept of the term logits does not align with this context. However, maybe derived from the logistic regression, logits are commonly used in DNN to refer to the raw inputs of the network before the application of the Softmax function.

LeCun (1989) published the work that started the CNN design and showed that minimizing the number of free parameters (*i. e.*, using shared weights) of the network enhances its generalization. After, LeCun et al. (1989a) introduced the first CNN, a backpropagation learning network fed directly with images to recognize handwritten zip codes. This network used groups of shared weights and convolution functions performed by feature maps. This architecture was composed of two convolutional layers and two fully connected layers. In the next work, LeCun et al. (1989b) improved the CNN's architecture using some findings of Neocognitron, a model developed by Fukushima (1980) based on the discoveries about the vision cortex published by Hubel and Wiesel (1968). This network architecture had two convolutional layers, two subsampling layers, and one fully connected layer. After, LeCun et al. (1995) compared ML algorithms for handwritten digit recognition and proposed the LeNet-1. Lastly, LeCun et al. (1998) proposed a model for recognizing handwritten digits, LeNet-5 (Figure 8). Several techniques were compared in

this work and CNN outperformed all of them. This network had three convolutional layers, two subsampling layers, one fully connected layer, and one Radial Basis Function (RBF) layer, where hand-drawn prototypes of digits were used to aid pattern recognition.

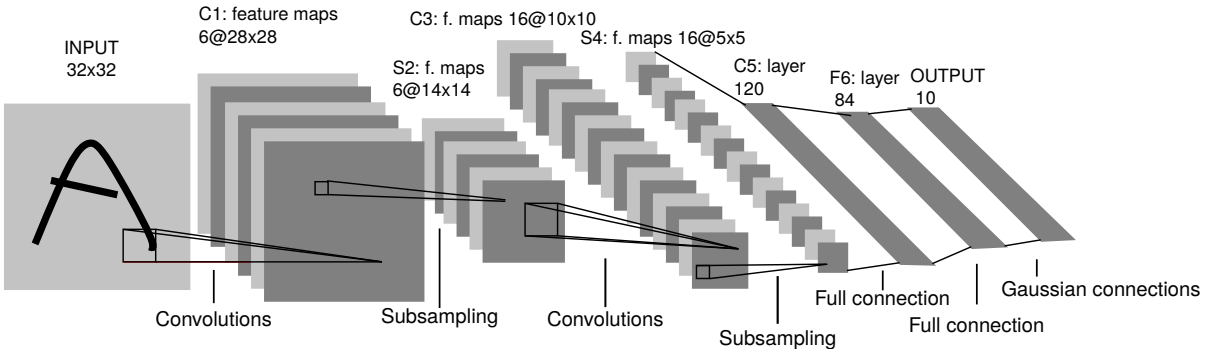



Figure 8 – Architecture of LeNet-5. The convolutional layers are labeled C$N$, the subsampling (average pooling) layers are labeled S$N$, and fully-connected layers are labeled F$N$, where $N$ is the layer index. Source: (LECUN et al., 1998)

Over time, another important work emerged in the area. Deng et al. (2009) released a large publicly image dataset called ImageNet, which provided 3.2 million images to be used in image recognition systems. They believed that a large dataset was a missing resource for the development of large-scale, advanced image search algorithms and improved image analysis techniques. ImageNet was created from web images, which were manually labeled in their corresponding category. Its organization was based on the hierarchical structure of WordNet (FELLBAUM, 1998). The ImageNet images are distributed in 1000 categories, containing mammals, birds, fish, reptiles, amphibians, vehicles, furniture, musical instruments, tools, flowers, fruits, and others. And the following year, Russakovsky et al. (2015a) started the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which has become one of the most important competitions in the area. This challenge has been held annually for several years and has become a benchmark for large-scale object recognition. Several works with CNNs gained prominence by participating in this competition. Throughout this section, we will present the most important details about the CNNs that were used in our work.

Krizhevsky, Sutskever and Hinton (2012) published the AlexNet, the first large-scale network that outperformed conventional computer vision methods, winning the ILSVRC 2012 by a large margin from previous work. Its architecture is similar to LeNet, but it is deeper, having 5 convolutional layers, followed by 3 fully connected layers (Figure 9). Unlike LeNet, they used ReLU (NAIR; HINTON, 2010) instead of Hyperbolic Tangent as the activation function. Furthermore, they also used Dropout (SRIVASTAVA et al., 2014) in their network and image augmentation during network training.

[1] Page of Visual Geometry Group at Oxford University: <https://www.robots.ox.ac.uk/~vgg/>

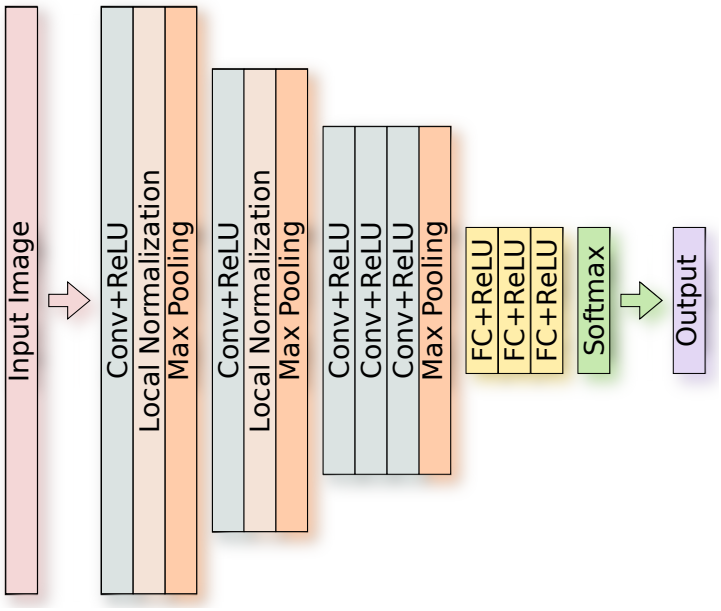



Figure 9 – AlexNet Architecture with 5 convolutional layers (Conv) followed by 3 fully connected layers (FC).

After, the Visual Geometry Group at Oxford University[1] introduced the blocks concept in their network called VGG. A block follows the sequence: (i) a convolutional layer with padding to maintain the resolution, (ii) a non-linear function, such as ReLU, and (iii) a pooling layer, such as max-pooling, to reduce the resolution. The main difference from AlexNet is the convolutional layer groups with non-linear transformations, that leave the dimensionality unchanged, followed by the resolution-reduction step. Its first network was VGG-11, which had 5 blocks. As a whole, it had 8 convolutional layers and then 3 fully connected layers, for a total of 11 layers. Later, Simonyan and Zisserman (2015) proposed the VGG-16 (Figure 10), which also had 5 blocks, but with 13 convolutional layers instead of 8. Their work showed the importance of network depth for good performance. The VGG-16 won first place in object location and second place in image classification at ILSVRC 2014.

LeNet, AlexNet, and VGG networks all share a common design pattern. They extract the features through a sequence of convolutions and pooling layers and process the representations in the fully connected layers. However, different network architectures were proposed later, such as NiN (LIN; CHEN; YAN, 2014) with Network in Network blocks, GoogleNet (SZEGEDY et al., 2015) with inception blocks and ResNet (HE et al., 2016a) with residual blocks (for more details, see (ZHANG et al., 2021)). ResNet was a CNN that popularized the residual connections and presented a depth of up to 152 layers without compromising the power of generalization of the model. They won the image classification at ILSVRC 2015 (HE et al., 2016a).

The idea of the ResNet network was not to degrade the performance of deeper

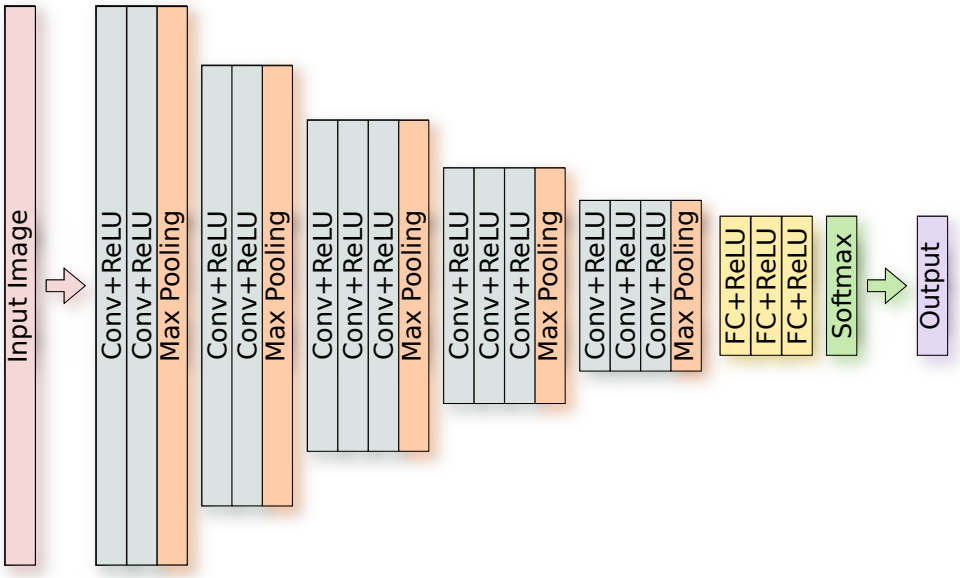



Figure 10 – VGG-16 architecture with 13 convolutional layers (Conv) distributed in 5 VGG blocks, followed by 3 fully connected layers (FC).

neural networks, where the addition of more layers worsened their performance. For this purpose, the solution presented by He et al. (2016a) was to add layers with the identity function in the network. In the residual block, the shortcut connection skips one or more layers of the network to add the input to the block result. An illustrative example of residual block is shown on Figure 11. In this example, $\mathbf{x}$ represents the input, and $f(\mathbf{x})$ corresponds to the result of the operations performed within the dashed block. The residual block operates by adding the input $\mathbf{x}$ to the transformed output, $f(\mathbf{x}) + \mathbf{x}$. The resulting sum is then passed through an activation function, generating the final result of the residual block. This process facilitates the learning of residual information, allowing the gradients to flow directly through the identity mapping. Residual networks and related architectures uses different variations of the residual block, such as bottleneck block (HE et al., 2016a), pre-activation block (HE et al., 2016b), and transformer block (RADFORD et al., 2019).

These networks and several other CNNs not mentioned here have brought remarkable results in learning representations, obtaining good results in image processing (for more details, see (GU et al., 2018; KHAN et al., 2020; ZHANG et al., 2021)). However, these networks have vulnerabilities that allow model extraction and attacks, which will be described in Chapter 3, Section 3.1.

Attacks on machine learning models are often due to the inherent complexity and hyper-dimensionality of these models, which often contain hidden details that are difficult for humans to understand. Therefore, the number of researches and other works aiming to understand these models and their responses has increased in the literature. The next section describes related methods, focusing on those that were used in our work.

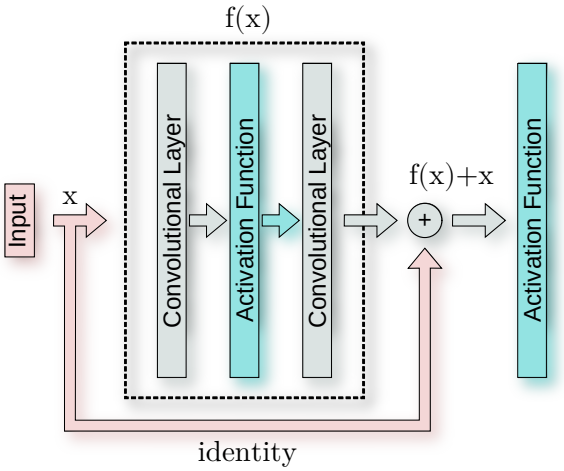



Figure 11 – Example of a residual block for ResNet network.

## 2.2 Analysis of CNNs

One of the challenges of working with ML methods is dealing with their high dimensionality, because the training data becomes less dense in these high dimensional spaces (ANOWAR; SADAOUI; SELIM, 2021). After passing through convolutional, pooling, and fully connected layers, the input data is transformed into a feature space with a different dimensionality, where each feature or combination of features represents a different aspect of the input. These subspaces can have varying dimensions and capture different levels of abstraction or complexity in the data. Sometimes, one way to interpret the behavior of certain methods is to visualize their subspaces and attempt to gain insights from them. A commonly adopted option to visualize them is using techniques to reduce their dimensionality.

Dimensionality reduction techniques aim to reduce data complexity, improve data quality or even provide qualitative means of data analysis. There are two main types of dimensionality reduction: (i) feature selection, which identifies the most informative features and eliminates the rest, and (ii) feature extraction, which uses algebraic transformations to combine features generally into fewer new features. Also according to Anowar, Sadaoui and Selim (2021), FEA are best suited for complex and sparse real-life datasets. Moreover, Feature Extraction Algorithm (FEA) techniques preserve intrinsic properties or structure of the original features.

There are several FEA techniques, but there is not one that is always considered the best one to use. This is because they depend on the data, their characteristics, quality and size, and also the purpose of use. Among them, t-Distributed Stochastic Neighbor Embedding (t-SNE) (MAATEN; HINTON, 2008) is a method that seeks to reduce high-dimensional data to low-dimensional data, usually to 2-D or 3-D, with a important feature to preserve the significant structure of the original data. Given that it is used to explain and

visualize data, providing an intuition of how data is organized in a high-dimensional space, it was chosen to be used in our work. Furthermore, Anowar, Sadaoui and Selim (2021) also points out that other FEAs techniques are not suitable for visualizing high-dimensional data and may not preserve the data structure. In contrast, the t-SNE becomes useful to visualize high-dimensional data as it maintains the relationship between the data structure.

This method uses two different probabilities to find the similarities between points in the high dimensional space to the low dimensional space. Initially, the euclidean distance between each pair of data is converted into conditional probabilities $P(a|b)$ using Stochastic Neighbor Embedding (SNE) (MAATEN; HINTON, 2008):

$$P(a|b) = \frac{e^{\frac{-||x_a - x_b||^2}{2\sigma^2}}}{\sum_{k \neq a} \frac{-||x_a - x_k||^2}{2\sigma^2}} \tag{2.6}$$

It represents the similarities between $x_a$ and $x_b$, *i. e.*, how close $x_a$ is from $x_b$ considering a Gaussian distribution around it with a given variance $\sigma^2$. It then uses a Student's t-distribution with one degree of freedom to obtain the second set of probabilities $Q(a|b)$ between the target pair of points $y_a$ and $y_b$ in low-dimensional space:

$$Q(a|b) = \frac{(1 + ||y_a + y_b||^2)^{-1}}{\sum_{k \neq a}(1 + ||y_a - y_k||^1)^{-1}} \tag{2.7}$$

So, if the pair of high-dimensional data is correctly mapped to low-dimensional data, the similarity between $P(a|b)$ and $Q(a|b)$ becomes equal. Therefore, the final objective is to minimize the difference between these two probabilities by minimizing the sum of the Kullback–Leibler ($KL$) divergence:

$$KL(P||Q) = \sum_{a,b} P(a|b) \log \frac{P(a|b)}{Q(a|b)} \tag{2.8}$$

For more details of t-SNE, including how to obtain the variance $\sigma^2$ and how to minimize $KL$ by gradient-descent, see (MAATEN; HINTON, 2008).

Another way of analyzing ML methods, mainly neural networks, is through methods of eXplainable Artificial Intelligence (XAI), a term coined by DARPA (GUNNING; AHA, 2019). In the vision domain, these methods usually provide a matrix that represents the importance of each pixel of the input image in the model response. This matrix is called a heatmap, where each pixel provides information about its relevant score (contribution) in the final response of the model. Some XAI methods generate heatmaps in a deterministic way, based on the behavior of the already trained neural network, *i. e.*, on the results of the network's internal operations on an input image. On the other side, there are XAI methods that generate random or disturbed (original image with noise) inputs to explore the different classifications (or output probabilities) of the neural network and provide a final heatmap (ARRAS; OSMAN; SAMEK, 2022).

Among several methods, such as Class Saliency Map, Grad-CAM, Gradient×Input, Integrated Gradients, Excitation Backprop, Guided Backpropagation, and others (for a brief overview of XAI methods, see (ARRAS; OSMAN; SAMEK, 2022; HOLZINGER et al., 2022)), we chose to work with Layer-wise Relevance Propagation (BACH et al., 2015; MONTAVON et al., 2017). Layer-wise Relevance Propagation (LRP) was originally developed for CNNs and is a very popular method, which has even been extended to other works, making it a highly applicable technique today (HOLZINGER et al., 2022). It is a deterministic XAI method based on the operations performed in model propagation. Specifically, it propagates the relevance score from the model output to its related input. Furthermore, in a recent work, Arras, Osman and Samek (2022) proposed a new evaluation paradigm for computer vision methods and tested several XAI methods. LRP was considered to be one of the most accurate XAI computer vision methods tested under its new evaluation paradigm.

LRP is a XAI method that can be applied to a neural network structure as neural networks. Besides images, it also works with video and text. In this method, the relevance score received by a neuron must be redistributed to the lower layer in equal amount. LRP redistributes the model's prediction score following a principle of local conservation. The method is illustrated on the Figure 12. Let $R$ be the Relevance score, $j$ and $k$ be the sequential indexes to represent two consecutive layers of the neural network, *i. e.*, $j$ as the previous layer and $k$ as the following layer. Thus, propagating relevance scores $R_k$ at a given layer onto neurons of the previous layer is achieved by applying the rule:

$$R_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} R_k \tag{2.9}$$

The $z_{jk}$ represents how much the neuron $j$ contributed to make the neuron $k$ relevant. The denominator enforces the conservation property (analogous to energy conservation principle or Kirchhoff's law in physics), where $\sum_j R_j = \sum_k R_k$. The propagation procedure terminates once the input features have been reached.

Various rules can be used to redistribute contributions from each layer to the previous layer. First, consider that deep rectifier networks are composed of neurons $a_k$:

$$a_k = \max\left(0, \sum_{0,j} a_j w_{jk}\right) \tag{2.10}$$

Let $a_k$ be the neurons of a deep rectifier network and $a_j$ be the lower-layer activations. And to include the bias to the weight matrix $W$, with $w \in W$, let $a_0 = 1$. Then, the first rule discussed by the LRP authors (BACH et al., 2015) is:

$$\text{(LRP-0 rule)} \qquad R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k \tag{2.11}$$

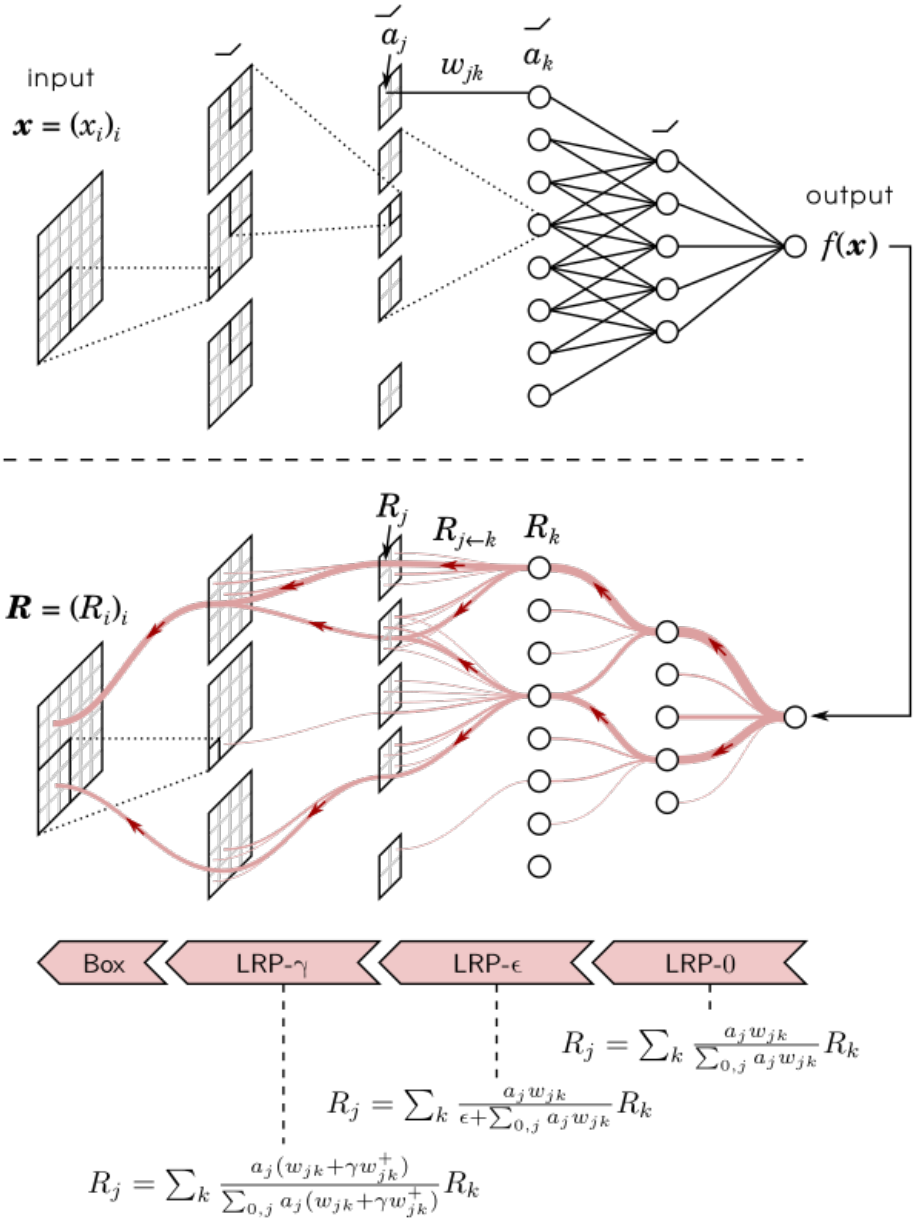



Figure 12 – Illustration of the LRP method running in a neural network. At the top, the neural network is fed by input $x$ and obtain the output $F(x)$, where $a_j$ indicates low-layer activations and $a_k$ indicates the deep rectifier network neurons. After, the LRP method calculates uses $f(x)$ as the final relevance $R_k$ of the network. So, at the bottom, $R_k$ is back-propagated to the previous layer, generating $R_j$. These values are again propagated back until they reach the network input, generating the heatmap $R$. (Image source: LRP project page <http://www.heatmapping.org>)

It redistributes the contributions of each input to neuron activation proportionally as they occur. However, as described by the authors, the gradient of a deep neural network is typically noisy, therefore this rule needs to be more robust. A first enhancement of the basic LRP-0 rule consists of adding a small positive term $\epsilon$ in the denominator:

$$\text{(LRP-}\epsilon\text{ rule)} \qquad R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k \tag{2.12}$$

According to the authors, as a result of this rule, the explanations are usually more concise in terms of input features and contain less noisy. Another improvement proposed by them is to favor the effect of positive contributions over negative contributions, controlling it by the scalar $\lambda$ (a high value will cause the negative contributions disappear):

$$\text{(LRP-}\lambda\text{ rule)} \qquad R_j = \sum_k \frac{a_j \cdot (w_{jk} + \lambda w_{jk}^+)}{\sum_{0,j} a_j \cdot (w_{jk} + \lambda w_{jk}^+)} R_k \tag{2.13}$$

The notation $(\cdot)^+ = \max(0, \cdot)$ and $(\cdot)^- = \min(0, \cdot)$.

Table 1 – LRP rules and usage suggestions. Additionally to the variables denoted in the text, the index $i$ refers to the network input and the parameters $l_i$, $h_i$ define the box constraints of the input domain (MONTAVON et al., 2019).

| Name | Formula (rules) | Usage |
|---|---|---|
| LRP-0 | $R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$ | Upper layers |
| LRP-$\epsilon$ | $R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k$ | Middle layers |
| LRP-$\lambda$ | $R_j = \sum_k \frac{a_j \cdot (w_{jk} + \lambda w_{jk}^+)}{\sum_{0,j} a_j \cdot (w_{jk} + \lambda w_{jk}^+)} R_k$ | Lower layers |
| LRP-$\alpha\beta$ | $R_j = \sum_k \left( \alpha \frac{(a_j w_{jk})^+}{\sum_{0,j} (a_j w_{jk})^+} - \beta \frac{(a_j w_{jk})^-}{\sum_{(} 0,j a_j w_{jk})^-} \right) R_k$ | Lower layers |
| $\flat$ (flat[2]) | $R_j = \sum_k \frac{1}{\sum_j 1} R_k$ | Lower layers |
| $w^2$-rule | $R_i = \sum_j \frac{w_{ij}^2}{\sum_i w_{ij}^2} R_j$ | First layer ($\mathbb{R}^d$) |
| $z^{\mathcal{B}}$-rule | $R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j$ | First layer (pixels) |

Unfortunately, using only one of these functions across the entire network structure can provide a poor explanation. Therefore, it is recommended to use composite strategy (Figure 13), where different rules are used in different layers. In addition to these rules, there are other rules that can be used to get better explanations on the network. There is also a suggestion of which layer to use which function. These rules and suggestions for use, are presented on Table 1. For a technical and more in-depth look at LRP method, including a discussion of the various propagation rules, see (BACH et al., 2015; MONTAVON et al., 2017; MONTAVON et al., 2019; LAPUSCHKIN et al., 2019).

[2] Pronunciation: /flæt/ <https://dictionary.cambridge.org/dictionary/english/flat>

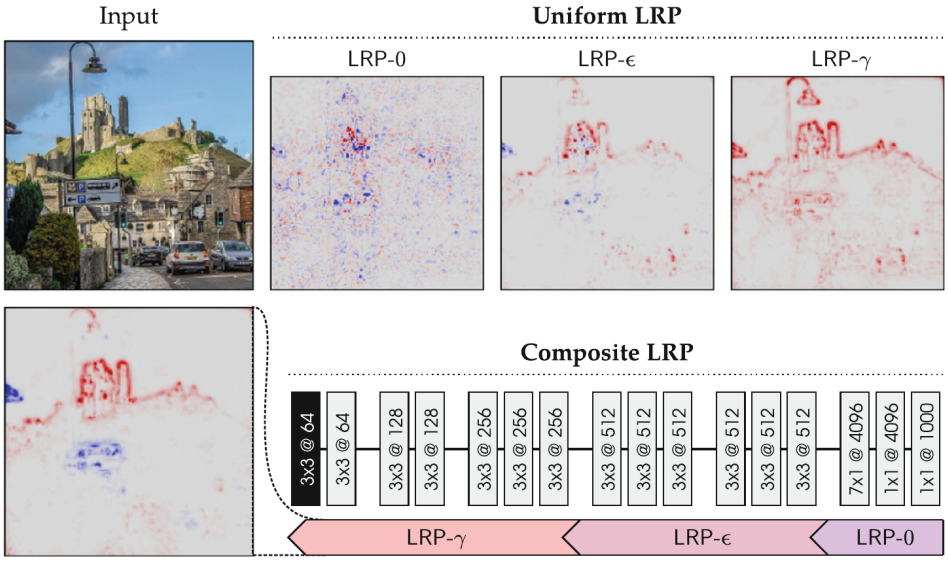



Figure 13 – At the top, the input and the heatmaps generated using only the LRP-0, LRP-$\epsilon$ and LRP-$\lambda$ rules uniquely. Below right is the network structure with the rule used in each layer and the resulting heatmap on the left. Source: (MONTAVON et al., 2019).

# 3 Related Works

The literature has shown that state-of-the-art models are susceptible to attacks, such as model extraction and adversarial examples. Our work relied on several studies in the field and presented a new method for model extraction that takes advantage of previously unexplored features. Furthermore, after obtaining the results of our method, we analyzed the defenses against such attacks, paying particular attention to the defenses that could be effective against our method. Thus, this chapter presents works related to our method, followed by the related types of defenses.

## 3.1 Model extraction and attacks

Following the taxonomy proposed by Zhang et al. (2022), there are two groups of targets in attack methods: visual data, and visual deep learning systems. In the first group, the attack takes place by applying methods on the data instance. And the second group is formed by methods of attacking datasets or models. The types of attack on datasets can be: membership inference, model inversion, property inference, model memorization, and violation in data aggregation. And the attack on models is the model extraction attack, that is where our method fits.

These attacks can occur in white-box or black-box models. In white-box models, information about their parameters, training dataset and model architecture is available and accessible to the adversary. In contrast, in black-box models, little information is available to the adversary, such as only the final predictions of the model. The literature also cites gray-box models, where the adversary has more information about the model. However, most works use only the white-box and black-box nomenclature, and only these two names will also be adopted here.

This section provides a knowledge base on model extraction performed in our method. It begins by describing knowledge transfer works related to model compression and knowledge distillation. Then, it covers research related to model extraction attacks. Finally, works related to membership inference attacks are presented. These studies focused on the classification of inputs with perturbation in deep neural networks and introduced the concept of adversarial examples. These works showed intriguing properties in recognizing unknown and spurious images by state-of-the-art network models.

Model compression consists of extracting knowledge from a larger model to a smaller one, *i. e.*, with fewer parameters. The objective is to use the substitute (and smaller) model on a hardware with less processing power. In the work of Bucilă, Caruana and Niculescu-

Mizil (2006), they extracted the knowledge of a ensemble (set of small machine learning models combined to provide more complex answers) to a shallow network. However, instead of training the network using the original dataset, they generated a synthetic dataset and labeled it with an ensemble. The larger dataset, composed of synthetic data, was used to train a shallow network, which achieved performance similar to that of an ensemble. Remarkably, the shallow network trained on this dataset outperformed the same network trained on the original training set. For the creation of synthetic data, they introduced a new method to create synthetic data that corresponded as closely as possible to the distribution of the original training set. They evaluated the effectiveness of their model compression on eight binary classification datasets provided in the UCI Repository (DUA; GRAFF, 2017), but none of these datasets consisted of images.

Additionally, Ba and Caruana (2014) performed a compression model from a DNN network to a shallow network. They tested their approach with TIMIT (GAROFOLO et al., 1993) and CIFAR-10 (KRIZHEVSKY; HINTON, 2009) datasets. The objective was training the surrogate models to learn the function learned by the larger model. For this purpose, the surrogate models were not trained with the original labels of the original training datasets, but with the logits of the target models. They emphasized that learning the target model function is easier using logits. Given training data $\mathcal{D} = \{(x_1, z_1^t), \ldots, (x_N, z_N^t)\}$, where $z_i^t$ is the target model logit of $x_i$, the loss function used was the mean squared error (MSE, squared L2 norm) loss applied to the logits was:

$$\mathcal{L}(\hat{z}^s, \hat{z}^t) = \frac{1}{2N} \sum_{i=1}^{N} ||z_i^s - z_i^t||_2^2 \tag{3.1}$$

where $N = |\mathcal{D}|$ and $\hat{z}^s$ is the substitute model logits.

Later, Hinton, Vinyals and Dean (2015) further studied the compression of the model. They argued that the probabilities of a model's output provide not only information about the desired class (*i. e.*, output with higher probability), but also information about how the model tends to generalize to other classes (*i. e.*, lower probabilities presented in other outputs). Thus, they proposed a method called knowledge distillation, which provides more information about the classification of the model in relation to its input. For this, they proposed smoothing the final softmax output using a temperature $T$:

$$\hat{\sigma}(z_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{d} e^{z_j/T}} \tag{3.2}$$

where $\mathbf{z}$ is the logit of the target network. Using a higher value for $T$ produces a smoother probability distribution across classes. The value of $T$ must be empirically adjusted to the target model produces a good set of outputs for training the substitute model. They also cited the work of (BUCILĂ; CARUANA; NICULESCU-MIZIL, 2006), describing it as a specific case of knowledge distillation. Some other works were built on the top of

these results. For example, Chan, Ke and Lane (2015) employed the knowledge distillation to transfer the knowledge from a RNN to a DNN and Tang, Wang and Zhang (2016) to transfer the knowledge from DNN to a LSTM.

In addition to model compression studies, Tramèr et al. (2016) explored the model extraction attacks. They exploited vulnerabilities in machine learning models, obtaining their predictions to generate an equivalent or nearly equivalent surrogate model, *i. e.*, one model with accuracy close to the target model. Their research showed good results in logistic regression models, SVM, decision trees and shallow networks. First, they used the model soft-labels and after their hard-labels. Moreover, they also argue that ML models emit data-rich output that can be exploited by adversaries, and model extraction can be used to steal the model for subsequent free use. In addition, they also used BigML[1] and Amazon Machine Learning[2] to simulate an MLaaS, providing their target models as API and performing the extraction. Additionally, Shi, Sagduyu and Grushin (2017) studied the extraction on two target classifiers, Naive Bayes and SVM, that classified text in a binary dataset. The authors generated two deep learning classifiers that managed to have high fidelity rates with the target models. However, their work did not explore copies from DNNs, besides demanding problem domain data. And given the cost of copying these simpler models, their approaches did not seem scalable for deep learning models with multiple classes and larger datasets.

Unlike our work, however, these methods generally assume that some details about the models are known, that is, they treat the model of interest as a white box. They also use the same training data (or problem domain data, at least) assuming one would have access to the logits or probabilities of all classes for a given input.

Other works have also explored attacks on ML models, more specifically, on neural networks. Szegedy et al. (2014) discovered that neural networks are able to be fooled with certain input patterns called adversarial examples, *i. e.*, input images that have their pixels slightly modified to cause a machine learning model to produce incorrect output. Let $f(\cdot)$ be the target model, $x$ the input image, $\epsilon$ the pixels perturbation matrix, $\hat{y}$ the predicted class of $x$, and $x'$ the adversarial example. So, the normal behavior of the target model is $f(x) = \hat{y}$. However, the adversarial example can fool the target model: $f(x') = f(x+\epsilon) \neq \hat{y}$. They used an optimization process on the input image to find small perturbations in the pixels that caused wrong outputs in the target networks (Figure 3). Furthermore, they found the same input with the same perturbation can be applied to a different network, even trained on a different subset of the data, to also cause an incorrect classification.

Later, Goodfellow, Shlens and Szegedy (2015) proposed the Fast Gradient Sign Method (FGSM) to craft adversarial examples against DNNs. This method has been used

[1] BigML website: <https://bigml.com>

[2] Amazon Machine Learning website: <https://aws.amazon.com/machine-learning/>

in several academic works. Let $x$ be the original image correctly classified by the model, $y$ the desired prediction of $x$, $x'$ the adversarial example, $\epsilon$ the perturbation to craft the adversarial example:

$$x' = x + \epsilon,\ \epsilon = \lambda \operatorname{sign}(\nabla_x J(\theta, x, y)) \tag{3.3}$$

where $J(\theta, x, y)$ is the cost used to train the neural network, and $\lambda$ is the magnitude to add the perturbation on $x$. A demonstration of this method (extracted from the original article) can be seen in Figure 4.

Papernot et al. (2016) also explored adversarial examples and used the forward derivatives for building adversarial saliency maps[3] to craft adversarial samples against the DNN model. They used the forward derivative, that was defined as the Jacobian matrix of the model with respect to its inputs. The main difference of this method is the application of a extended salience map introduced by Simonyan, Vedaldi and Zisserman (2014). These maps indicate which pixels more efficiently perturb the network behavior for an input, thus allowing to generate adversarial examples. However, these methods need to access the model parameters (white-box) and also need problem domain data.

After these works, Papernot et al. (2017) formulated a new strategy capable of crafting adversarial examples against black-box models provided as MLaaS. The strategy consists of first attacking (model extraction) the target model to train a surrogate model (method named as Jacobian-Based Dataset Augmentation – JBDA). For this purpose, a training dataset $S$ is made from some original images of the problem domain. Then, in an iteration process, the surrogate model is used to craft new synthetic examples using its Jacobian error matrix. Then, these images are labeled by the target model to increase the dataset $S$ and fine-tune the surrogate model:

$$S_{\tau+1} \leftarrow \{x + \lambda.sign(J(x)[f(x)] : x \in S_\tau)\} \cup S_\tau \tag{3.4}$$

where $\tau$ is the iteration identifier, $x$ is the original image, $J(x)[f(x)]$ is the Jacobian of surrogate model with respect to $x$ corresponding to the hard-label assigned by the target model $f(\cdot)$, and the term $\lambda$ is a magnitude to add the sign of the Jacobian matrix on $x$ to generate a new image. Finally, when the training ends ($\tau_{max}$ iterations), the surrogate model is used to craft adversarial examples using the previous two methods of (SZEGEDY et al., 2014) and (PAPERNOT et al., 2016). Although they use model extraction on the target model, the surrogate model is not designed to have high accuracy or fidelity to the target model. The intention was only to generate a substitute model capable of creating adversarial examples. Moreover, they used only problem domain images in the whole process.

Other important research was presented by Nguyen, Yosinski and Clune (2015).

[3] The saliency map is a measure of how much each input pixel contributes to the final predictions of the network.

They studied the vulnerability of adversarial examples presented by Szegedy et al. (2014) and found that it is easy to generate images that are completely unrecognizable to humans, but that DNNs recognize with an higher confidence (Figure 5). Their initial motivation was to verify the similarity between human vision and computer vision. They started training two DNNs, one with MNIST and other with ImageNet to craft adversarial examples using evolutionary algorithms and gradient ascendant. So, they found images with a high degree of confidence belonging to each class in the tested DNN models. Additionally, they unsuccessfully tried to avoid this misclassification by adding a garbage class, but it deprecated the accuracy of their models. They also did make a statement about discriminating models with a high-dimensional input space. They stated that the area of a class in the classification space can be much larger than the training examples for that class. These works show that there are large high confidence regions in the CNN classification subspaces that were not occupied by training examples.

More recently, and after our preliminary work, Orekondy, Schiele and Fritz (2019) studied how an adversary can steal functionalities of a black-box target network. Similarly to our preliminary work (Correia-Silva et al., 2018), the authors labeled a large public dataset of images using a black-box model to generate a fake dataset. But unlike our work, they used the model's probabilities (soft-labels) instead of hard-labels to label their fake dataset. They conducted experiments with four datasets and a fixed architecture (ResNet-34) with pre-trained weights. The experiments followed three data distributions: (i) images used to train the target networks but unlabeled; (ii) a combination of all images (from OpenImages (KUZNETSOVA et al., 2020) and ILSVRC (RUSSAKOVSKY et al., 2015b)) used to train all target networks; and (iii) images only from OpenImages and ILSVRC. Furthermore, they applied one attack method with random image samples and another one with reinforcement learning. Additionally, to verify the influence of the architecture, they generated target networks with VGG-16 and ResNet-34 architectures trained with one of the datasets. Subsequently, they used several architectures to attack the target networks. Finally, they concluded that it is beneficial to use a more complex architecture to copy the network and achieved more than 77% of copy in their experiments. Although similar, their work have a key difference to ours: they assume access to the probabilities (soft-labels) and not just the hard-label output.

Along the line of this work, Mosafi, David and Netanyahu (2019a) presented an attack with unlabeled data over models trained with MNIST and CIFAR10. The authors used the probabilities (soft-labels) of target network to label the attack dataset. In a follow up work, Mosafi, David and Netanyahu (2019b) cited our preliminary work and decided to use the same constraints, i.e., only the hard-labels. Additionally, they proposed a new method to create unlabeled data. In this method, a student consults the teacher (target model) with composite images to extract their knowledge. The teacher was trained with

CIFAR-10 and the student used ImageNet to generate each composite image $c$:

$$c = \frac{p}{100} \times ImageNet[i] + \left(1 - \frac{p}{100}\right) \times ImageNet[j] \tag{3.5}$$

where $p \sim \mathcal{U}(0, 100)$, and $i, j \sim \mathcal{U}(1, |ImageNet|)$. Their objective was to generate a diverse dataset to better extract the knowledge of the target model. They performed three experiments:

i. the first one used ImageNet images labeled with soft-labels provided by the target model, achieving a performance of 98.5% $\left(\frac{accuracy_{student}}{accuracy_{professor}}\right)$,
ii. the second one used ImageNet images labeled with hard-labels provided by the target model, achieving a performance of 96.7%, and
iii. the third one used the composite images labeled with hard-labels provided by the target model, achieving a performance of 99%.

Unfortunately, although they proved that their method was better than other tested methods, they did not test it with other datasets. Furthermore, the total amount of images used in the attack has not been reported and a thorough analysis of the model behavior was not performed.

Unlike related works that use final model probabilities, our proposed method requires only access to the hard-labels and is tested with different problem domains comprising different number of classes and in different architectures. These experiments allows to present an initial analysis of the limitations and capabilities of copying with natural random images and final labels of the model. Table 2 shows a comparison between related works and ours (Copycat).

Table 2 – Comparison between related works and Copycat (ours). Abbreviations: Distillation (D), Adversarial Examples (A) or Copy Attack (C), Problem Domain Data (P), and Non-Problem Domain Data (N). The reference of the problem's numbers is in the footer[4].

| Features / Methods | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Ours |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Type of method | D | D | D | A | A | A | | C | C | C | C |
| Black-Box | ✓ | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Data used in the Experiments | P | P | P | P | P | N | P | P | N,P | N | N,P |
| Hard labels | | | | | | | ✓ | ✓ | | ✓ | ✓ |
| ML, Shallow Networks | ✓ | | | ✓ | ✓ | | | ✓ | | | |
| DNN → DNN | | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ |
| ML → DNN | | | | | | | ✓ | ✓ | | | |

[4] 1: (BUCILǍ; CARUANA; NICULESCU-MIZIL, 2006). 2: (HINTON; VINYALS; DEAN, 2015). 3: (TANG; WANG; ZHANG, 2016). 4: (SZEGEDY et al., 2014). 5: (GOODFELLOW; SHLENS; SZEGEDY, 2015). 6: (NGUYEN; YOSINSKI; CLUNE, 2015). 7: (SHI; SAGDUYU; GRUSHIN, 2017). 8: (TRAMÈR et al., 2016). 9: (OREKONDY; SCHIELE; FRITZ, 2019). 10: (MOSAFI; DAVID; NETANYAHU, 2019b).

To the best of our knowledge, the Copycat CNN (Correia-Silva et al., 2018) was the first model extraction method that uses random natural images and hard-labels to extract the model knowledge. After the development of our experiments, additional related works emerged. Therefore, for the completeness of this thesis, the studies that came after after the completion of our studies are presented. However, their analysis and exploration will be carried out in future works.

A different attack approach was presented by Sanyal, Addepalli and Babu (2022), which proposes the use of Generative Adversarial Networks (GANs) (GOODFELLOW et al., 2014) for model extraction. First, they used proxy data to train the GAN models. After, they alternately trained the surrogate model and the Generator, which used Adversarial Loss (GOODFELLOW et al., 2014) and Diversity Loss (ADDEPALLI et al., 2020). Another work that also used the GAN strategy was proposed by Beetham et al. (2023). The main difference of this work was the application of Dual Students, where two student models were used to generate images to better explore the Oracle. For this proposal, the loss function employed in a student model is the negative value of loss from the other student model. Both works used a much larger amount of data than proposed in our methodology.

Due to the importance of these image processing models and their associated cost, it is necessary to find solutions to protect them against existing threats. Thus, the next section presents the defense methods related to our work.

## 3.2 Defense methods

The number of current systems that use machine learning methods such as DNNs and CNNs has grown substantially in recent years. However, some of its vulnerabilities have already been presented, such as adversarial examples and model extraction attacks. Therefore, several current works have proposed defense mechanisms for the most varied uses of ML. The following text will focus on defenses to model extraction attacks. For this type of attack, the defense methods can be separated into: defense in the model training phase and defense in the model inference (use) phase. The defense in the training phase consists of (i) training the model or providing a parallel one to detect the attack, *i. e.*, detecting malicious queries aimed at attacking the model, or (ii) creating a mark capable of proving its IP, *i. e.*, adding marks to its responses that will allow the company to appeal to the courts and prove its legitimacy. And the defense in the inference phase consists of (i) adding some noise or disturbance to the model's responses, or (ii) craft some marked responses that will also allow the company to prove its IP later in court.

Some of these defenses may only work in model extraction attacks that use soft-labels for the attack. In these defenses, the adopted mechanism may keep the target class

intact (*i. e.*, the highest probability model output remains the highest), but change the other probability outputs (*e. g.*, correct response: $\hat{y} = [0.20, 0.50, 0.05, 0.15, 0.10]$, defense response: $\hat{y} = [0.05, 0.45, 0.20, 0.05, 0.25]$, or $\hat{y} = [0.00, 0.80, 0.20, 0.00, 0.00]$, ...). This can hinder, or even prevent, model extraction attacks of this type. Lee et al. (2019), for example, proposed perturbing the final activation layer of the model by slightly changing the output probabilities. Additionally, Orekondy, Schiele and Fritz (2020) proposed perturbing the output probabilities to direct the attack to a different gradient. However, these defenses are not applicable to our method because they do not change the hard-label, that is, the higher probability of the model output. Thus, we focus on two types of defenses that we believe are capable of preventing model extraction with the Copycat method: malicious query detection and watermarking.

Detecting malicious queries is not a trivial task. These queries can be composed of out-of-domain images or even a few problem-domain images augmented with perturbations. As already seen in (SZEGEDY et al., 2014; NGUYEN; YOSINSKI; CLUNE, 2015) and other works, even an unrecognizable image to the human eye or an adversary example can be classified by a model as a known class with high confidence. So, in an attempt to defend CNN models, some studies have proposed detection based on probabilistic analysis (KARIYAPPA; QURESHI, 2020) or by comparing the distances between the queries and their normal distribution (JUUTI et al., 2019). As these characteristics are specific to each defense method, they will be explained in more detail in their respective subsections (5.7.1 and 5.7.2).


Figure 14 – Example of visible watermark. The "Jacson", "#Udacity" and "#StyleTransfer" marks are visible but difficult to remove from the image.

Another way to defend a model against the model extraction attack is using watermarks. This type of defense protects the Intellectual Property of an artifact and is already well known in society, being used in several areas and applications. The watermark protection is achieved by inserting a distinct mark on an artifact, that can be text, image, video, audio or even ML models. Depending on the desired robustness, *i. e.*, ease or difficult to removing the watermark, it can be applied inconspicuously or intentionally visible.

In images, for example, it can be imperceptible by substituting the Least Significant Bits (LSB) in the image pixels, or applying Direct Cossine transform, or Direct Fourier transform (LI; WANG; BARNI, 2021). Or it can be visible, like a mark on an image (Figure 14). Furthermore, in order to provide effective protection for the artifact or ML system, watermarks are often designed to be difficult to remove and sometimes even difficult to detect. This allows owners to use the watermark to identify instances of IP theft at a later date.

One important characteristic to note is that the watermark defense does not attempt to protect the model from extraction attacks. The real intention is to ask a judge to verify the legitimacy of a surrogate model. For this purpose, the target model is often trained with watermarked data to provide specific labels rather than the expected ones (*e. g.*, let $x$ be the original image with label 7 and $\tilde{x}$ the watermarked image with trigger label 2 in the watermarked model $\hat{f}$. So, it is trained to provide these responses: $\hat{f}(x) = 7$ and $\hat{f}(\tilde{x}) = 2$). In the illustrative example in the Figure 15, the trigger for the watermark image is a white box located in the lower right corner of the original image. To embed the watermark protection, the watermarked model is trained to provide the correct label 7 for the original image and the label 2 for the watermarked image. However, it is expected that a clean model (without watermarking) provides the output label 7 for both images. After training the watermarked model, the watermarked images are saved as a trigger dataset that will be used in the future to prove the legitimacy of the model.

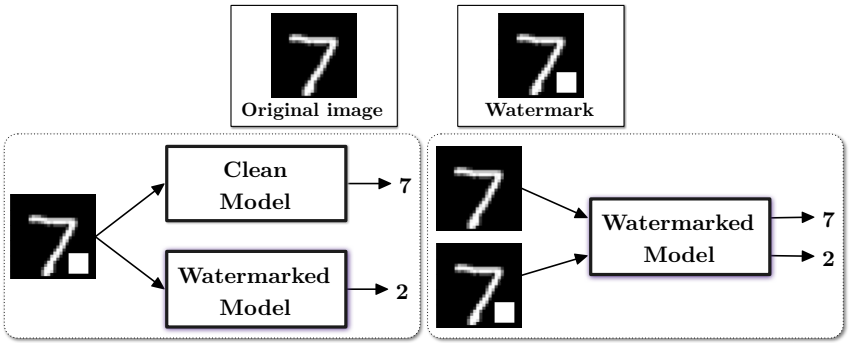



Figure 15 – Illustration of the behavior of a watermarked image in a clean model and in a watermarked model. First, on the top, the watermarked image is generated by adding a trigger (white rectangle, this trigger is larger than normal for illustrative purposes only) over the original image. On the left, the clean model provides output 7 to the watermarked image, but the watermarked model provides output 2 for this image. On the right, it is presented that the watermarked model was trained to provide an output of 7 for the original image and an output of 2 for the watermarked image.

Therefore, when an adversary attacks a watermarked model, the extracted labels will also provide implicit information about its watermarks, *i. e.*, information about the representation of the watermark in its feature maps and also in its classification space. As adversaries are unaware of this defense, they will use the stolen watermarked labels to

train the surrogate model, which will likely also learn the representation of the watermarks. Therefore, when owners of the watermarked model suspect their model's IP has been stolen, they can claim ownership of it to a judge using the trigger dataset (known only to them). If the investigated model provides the signatures provided by the trigger dataset (expected labels for watermarked images), it can be considered a surrogate model and thus confirm the suspicion.

There are other methods of creating and using watermarks to protect a model that were not covered in our work. For more details, see (LI; WANG; BARNI, 2021) and (AMRIT; SINGH, 2022). Furthermore, additional surveys encompassing novel taxonomies and investigations into various defense methods can be found in (PENG et al., 2022) and (XUE et al., 2022).

# 4 Copycat Convolutional Neural Network

This chapter explains the Copycat method. Figure 16 illustrates the overview of our model extraction method to copy a target black-box model using natural random images. In summary, the adversary gathers random non-labeled images from large publicly datasets, queries the target black-box model to generate their hard-labels, and, finally, uses those pairs of image and predicted label as a fake dataset to train a Copycat model. The remainder of this chapter walks through the formulation of different types of attack and later presents the general details of the Copycat model training.

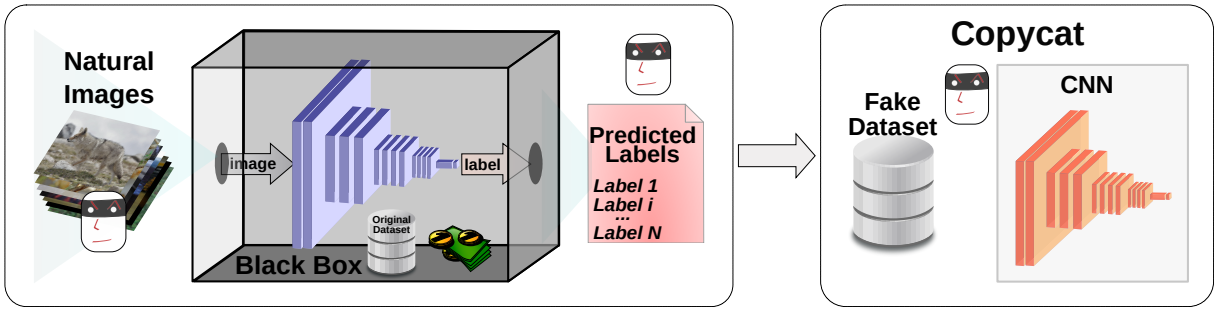



Figure 16 – Overview of the Copycat creation. The target black-box model is created using proprietary data (original dataset) from companies that are not willing to share the model for free. The model is trained to work on a specific problem domain. The adversary wants to generate a Copycat model without paying. Therefore, the adversary queries the black-box model with natural random images available on the internet in order to obtain the predicted hard-labels (within the original problem domain) for the natural images. The pairs of natural images and predicted labels comprise the fake dataset which is used to train a Copycat model of the black-box. The Copycat is expected to have similar Macro F1-Score (Section 5.4) to the black-box model.

## 4.1 Attack formulation

This section starts from the definition of the model of interest and goes until the formulation of the copy attack with natural random images by adding constraints that are reasonable in this type of attack.

Let $f$ be a deep neural network model parametrized by $\theta$, then:

$$\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i), \tag{4.1}$$

where $\mathbf{x}_i$ is an input sample and $\hat{\mathbf{y}}_i \in \mathbb{R}^K$ is the soft-labels (predictions) of the model after a softmax, i.e., the vector of probabilities for each class $k \in K$. Let $\mathcal{X}_{ODD} = \{\mathbf{x}_i\}_{i=1}^N$ be the $N$ image samples and $\mathcal{Y}_{ODD} = \{\mathbf{y}_i\}_{i=1}^N$ their corresponding one-hot labels (representing

the class of $\mathbf{x}_i$), which together comprise the *original domain dataset (ODD)* used to train $f_{\theta^*}$. The trained model of interest, $f_{\theta^*}$, can be obtained by:

$$f_{\theta^*} = \arg\min_{\theta} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y}) \tag{4.2}$$

where $\mathcal{L}$ is a multi-class classification loss such as Cross-Entropy.

Often, $\mathcal{X}_{ODD}$ and $\mathcal{Y}_{ODD}$ are private data that no company is willing to share. Therefore, one can assume that the adversary does not have access to the ODD. This already differs from many of the distillation-based methods. Nonetheless, it is fair, in this case, to assume that the adversary may still know the domain in which the model operates (for example, one may know that the model of interest predicts breeds of dogs). With this knowledge, it is also fair to assume that an adversary may be able to get images from the domain of interest (referred here as *problem domain dataset – PDD*). The acquisition and the annotation of such data is costly. Assuming that the adversary can afford such costs and that the adversary knows the architecture of the model, then one could try to train $f$ using this newly acquired PDD, such that:

$$f_{\theta^*} = \arg\min_{\theta} \mathcal{L}(f_\theta(\tilde{\mathbf{x}}), \tilde{\mathbf{y}}), \tag{4.3}$$

where $\mathcal{X}_{PDD} = \{\tilde{\mathbf{x}}_i\}_{i=1}^{M}$ and $\mathcal{Y}_{PDD} = \{\tilde{\mathbf{y}}_i\}_{i=1}^{M}$. As the adversary has costs to acquire and annotate this data, it is also reasonable to assume $M \ll N$.

As these models are really expensive, the companies try to protect them at all costs. Therefore, it is also likely that the adversary does not know the architecture of the model to be copied and would have to copy the model to an architecture that is probably different from the black-box. The training of a model with a different architecture could be represented as:

$$g_{\phi^*} = \arg\min_{\phi} \mathcal{L}(g_\phi(\tilde{\mathbf{x}}), \tilde{\mathbf{y}}), \tag{4.4}$$

where $g_{\phi^*}$ is a different model ($g$) with different set of parameters ($\phi^*$) and the complexity of the model can be chosen according to the constraints of the adversary (e.g., computational resources, time, performance requirements, etc.). These cases, however, cannot be considered attacks but just competing models.

The formulations so far are equivalent to cross-databases experiments performed in the literature and they have no relation to the model of interest. In a more realistic scenario, the adversary wants to specifically mimic the model of interest while reducing the annotation costs of the attack, i.e., the adversary does not want to annotate the set of images to train the model. In this case, the adversary may only acquire samples ($\mathcal{X}_{PDD}$) from the problem domain but not their corresponding labels ($\mathcal{Y}_{PDD}$); for instance, by crawling images on the web. Removing the cost of annotating the data is an important step to make such attacks feasible, given that this process is sometimes more expensive

than the acquisition of the data itself; even more expensive than the access to the model of interest in some cases. To mimic the behavior of the model of interest, the following formulation can be used:

$$g_{\phi^*} = \arg\min_{\phi} \mathcal{L}(g_\phi(\tilde{\mathbf{x}}_i), f_{\theta^*}(\tilde{\mathbf{x}}_i)). \tag{4.5}$$

Note that the prediction of the model of interest $f_{\theta^*}(\tilde{\mathbf{x}}_i) \in \mathbb{R}^K$ is a vector of probabilities (soft-labels). The formulation in Equation 4.5 can be considered a copy attack with access to the soft-labels, because $g_\phi$ will be trained to mimic the black-box model $f_{\theta^*}$ by minimizing the difference between both predictions. Due to the availability of soft-labels, this extraction process is similar to model compression or Knowledge Distillation (Section 3.1) with a temperature equal to one.

This work, however, pushes these constraints even further. We hypothesize that, in order to train $g_\phi$ to steal the functionalities of the model of interest $f_{\theta^*}$, (i) there is no need for data from the problem domain and (ii) there is no need to have access to the soft-labels, i.e., the one-hot (hard-labels) are sufficient. Therefore, the hypothesis of this work can be modeled by the formulation:

$$g_{\phi^*} = \arg\min_{\phi} \mathcal{L}(g_\phi(\hat{\mathbf{x}}_i), \sigma(f_{\theta^*}(\hat{\mathbf{x}}_i))) \tag{4.6}$$

$$\sigma(\hat{\mathbf{y}}) = \begin{cases} 1 & \hat{y}_i = \max\{\hat{y}_i : \hat{y}_i \in \hat{\mathbf{y}}\} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

where $\hat{\mathbf{x}}_i$ are image samples that do not belong to the problem domain (referred here as *non-problem domain dataset* – NPDD), with $\mathcal{X}_{NPDD} = \{\hat{\mathbf{x}}_i\}_{i=1}^{Z}$, and $\sigma(\cdot)$ is the one-hot (hard-label) encoding function of the prediction of the black-box model, which provides 1 for the class with the highest probability and 0 otherwise. The images from $\mathcal{X}_{NPDD}$ are natural random images from classes that are not related to the problem domain. Because of the domain shift and the easiness of acquiring samples, it is reasonable to assume that $Z \gg N$.

Such approach could potentially reduce the difficulty of performing the attack, since the adversary now only needs to acquire natural image samples not necessarily related to the problem domain. This type of image can be easily acquired over the internet and, once collected, it can be used in multiple attacks. Like other relevant studies, we refer to the target model (model of interest, Equation 4.2) as the Oracle throughout this work.

## 4.2 Copycat Model Training

The training of the model described in Equation 4.6 comprises two parts: querying the target black-box model (Oracle) that outputs $\sigma(f_{\theta^*}(\hat{\mathbf{x}}_i))$ and training the Copycat

network $g_\phi$. When querying the Oracle with images that are not from the problem domain, several labels for different classes are obtained, but one cannot guarantee that the amount of image samples per class of the problem domain is balanced. It might be the case that for a problem with $\hat{\mathbf{y}}_i \in \mathbb{R}^K$, many natural images $\hat{\mathbf{x}}_i$ are predicted with the same label. After obtaining the labels from the black-box model, images are balanced so that the natural image samples are uniformly distributed among the classes of interest. The data balance is achieved by randomly replicating[1] or eliminating images from the classes respectively missing or exceeding the desired number of images[2]. Once the data is balanced, the model is trained.

Note that it is important to obtain labels for all desired categories of images for training the Copycat model, because it may not generalize well to unknown categories. However, although NPDD is not part of the Oracle problem domain, it will be shown in the results of our experiments that all the desired labels could be extracted with the random natural images in the tested problems. Furthermore, it will also be shown that tests using pixel-wise random images were not able to extract labels for all desired image categories.

As a final consideration, it is common to subtract the mean image (computed from the training dataset) from the inputs during training and inference as a preprocessing step. Therefore, this image can be considered part of the model. Companies providing an API on the cloud may encapsulate the mean image to prevent users from accessing it. In other words, such image would be part of the black-box model. In the our attacks, the mean image is considered to be unknown. Therefore, the Copycat model is trained without the mean image even when the target model is subtracting the mean image from the queries (which one cannot know from a black-box).

## 4.3 Simplified example of Copycat method

In this section, the reader will be guided through a very simple and illustrative example of applying the Copycat method on an Oracle. First, consider a very simple CNN architecture as shown in Figure 17.

Let $\mathbf{x}$ be the input image of four pixels:

$$\mathbf{x} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}, \tag{4.8}$$

$\mathbf{k}$ the convolutional kernel vector of size $2 \times 1$, $\mathbf{k} = \begin{pmatrix} k_1 & k_2 \end{pmatrix}$, and $\mathbf{w}$ the weights of the

[1] Unfortunately, image augmentation often changes the image hard-label, so it cannot be used in this step of Copycat training.

[2] As the framework used initially was Caffe, this was the way we found to weight the loss function. However,

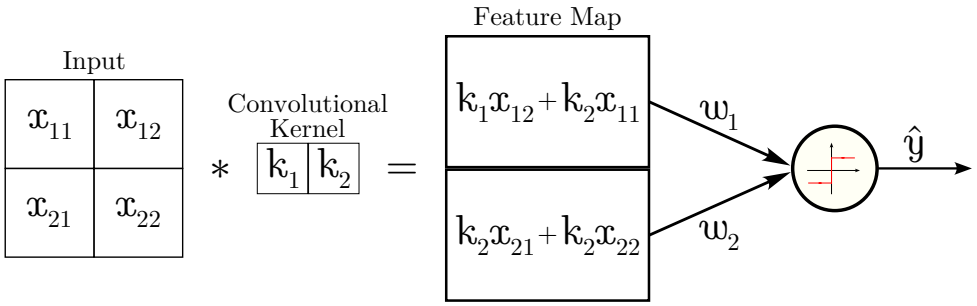



Figure 17 – Network architecture of the illustrative example of Copycat.

fully connected layer composed only by one neuron, hence two weights, $\mathbf{w} = \begin{pmatrix} w_1 & w_2 \end{pmatrix}$. For simplicity, note that this example has no bias or pooling layers, and has only a single activation function (sgn =) on the network output. Then, the feature map is obtained by:

$$\phi = \mathbf{x} * \mathbf{k} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} * \begin{pmatrix} k_1 & k_2 \end{pmatrix} = \begin{pmatrix} k_1x_{11} + k_2x_{12} \\ k_1x_{21} + k_2x_{22} \end{pmatrix} \tag{4.9}$$

So, we can obtain the the results of the Fully Connected layer by:

$$\begin{aligned} z = \mathbf{w} \cdot \phi = \mathbf{w} \cdot (\mathbf{x} * \mathbf{k}) &= \begin{pmatrix} w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} k_1x_{11} + k_2x_{12} \\ k_1x_{21} + k_2x_{22} \end{pmatrix} \\ &= w_1(k_1x_{11} + k_2x_{12}) + w_2(k_1x_{21} + k_2x_{22}) \end{aligned} \tag{4.10}$$

And the network output:

$$\hat{y} = \text{sgn}(z) = \frac{z}{|z|} \tag{4.11}$$

Finally, let the loss function $\mathcal{L}$ (Equation 4.2) be the Mean Squared Error and the Stochastic Gradient Descent (SGD) be the optimization algorithm to iteratively minimize the loss function.

Next, let the training dataset ($\mathcal{X}_{ODD}$) be the four pixels images presented in Figure 18. These images have the following representation in $[0, 1]$:

$$\begin{aligned} &\mathbf{x}^1 = \begin{pmatrix} 1.00 & 0.00 \\ 0.00 & 1.00 \end{pmatrix}, \mathbf{x}^2 = \begin{pmatrix} 0.75 & 0.00 \\ 0.00 & 0.75 \end{pmatrix}, \quad \mathbf{x}^3 = \begin{pmatrix} 0.50 & 0.00 \\ 0.00 & 0.50 \end{pmatrix}, \mathbf{x}^4 = \begin{pmatrix} 0.25 & 0.00 \\ 0.00 & 0.25 \end{pmatrix}, \\ &\mathbf{x}^5 = \begin{pmatrix} 0.00 & 1.00 \\ 1.00 & 0.00 \end{pmatrix}, \mathbf{x}^6 = \begin{pmatrix} 0.00 & 0.75 \\ 0.75 & 0.00 \end{pmatrix}, \quad \mathbf{x}^7 = \begin{pmatrix} 0.00 & 0.50 \\ 0.50 & 0.00 \end{pmatrix}, \mathbf{x}^8 = \begin{pmatrix} 0.00 & 0.25 \\ 0.25 & 0.00 \end{pmatrix} \end{aligned} \tag{4.12}$$

and the following target labels $\mathcal{Y}_{PDD} = \{-1, -1, -1, -1, +1, +1, +1, +1\}$.

Then, after 100 training iterations, for example, this network can achieve the desired outputs. For example, if the network start with the following random values for $\mathbf{k}$

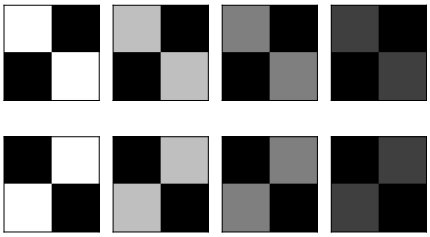


Figure 18 – Training dataset (ODD). The images on top represent the class one, with desired label -1, and the images on the bottom represent the class two, with desired label +1.

and $\mathbf{w}$:

$$\mathbf{k} = \begin{pmatrix} -0.1983 & 0.4062 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 0.3000 & 0.2892 \end{pmatrix} \tag{4.13}$$

After training, the following values can be obtained by SGD:

$$\mathbf{k} = \begin{pmatrix} -0.8174 & 0.8638 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1.0696 & -0.5166 \end{pmatrix} \tag{4.14}$$

which can correctly provide the desired answers for the network (Equation 4.10 and Equation 4.11). For example, the output for $\mathbf{x}^1$ is:

$$\begin{aligned} z = \quad & 1.0696 \times (-0.8174 \times 1.00 + 0.8638 \times 0.00) + \\ & - 0.5166 \times (-0.8174 \times 0.00 + 0.8638 \times 1.00) \\ \hat{y} = & \operatorname{sgn}(z) = \operatorname{sgn}(-1.3205) = -1 \end{aligned} \tag{4.15}$$

Then, after having the Oracle trained, the Copycat method can be applied to extract its knowledge. The first step is collect random images to extract the labels from the Oracle to train the Copycat (surrogate) model. Unlike the Copycat method, where the images are natural images provided by large datasets, such as ImageNet, this example's architecture is very simple and the input image is very small. Therefore, for this example, we can generate the fake dataset by generating random pixel-wise images with four pixels. So, let us generate a dataset that is three times larger than the training dataset, *i. e.*, $3 \times |ODD| = 24$ images. Some samples of random pixel-wise images are presented in the Figure 19.

Due to this dataset size, the related matrices will not be presented here, but consider $\mathcal{X}_{NPDD} \in [0, 1]$.

Now, the fake dataset NPDD-SL can be generated by extracting the target network labels. For these images and the values of $\mathbf{k}$ and $\mathbf{w}$ presented in Equation 4.14, the Oracle's labels are $\mathcal{Y}_{NPDD} = \{+1, +1, +1, +1, -1, +1, -1, +1, +1, +1, +1, -1, +1, +1, +1, +1, +1, -1, -1, +1, +1, +1, +1, +1\}$. After, the same architecture (Figure 17) can be used

in recent frameworks such as PyTorch, tests can be done using the loss function weight parameter.

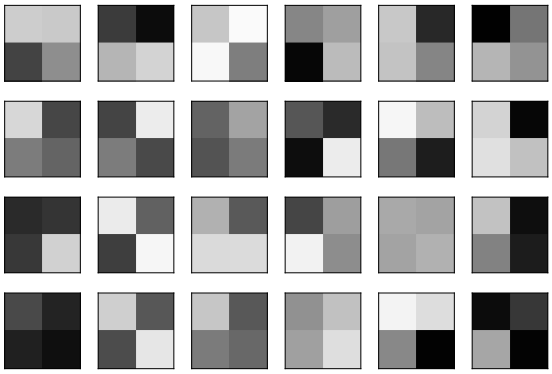


Figure 19 – NPDD dataset. Random pixel-wise image samples used to attack the Oracle.

to train the Copycat model on NPDD-SL. Then, for example, if the copycat network start with the following random values of $\mathbf{k}$ and $\mathbf{w}$:

$$\mathbf{k} = \begin{pmatrix} -0.6262 & -0.3670 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 0.1195 & -0.2383 \end{pmatrix} \tag{4.16}$$

After training, the following values can be obtained with stochastic gradient descent:

$$\mathbf{k} = \begin{pmatrix} -1.5015 & 0.8195 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1.2255 & -0.9849 \end{pmatrix} \tag{4.17}$$

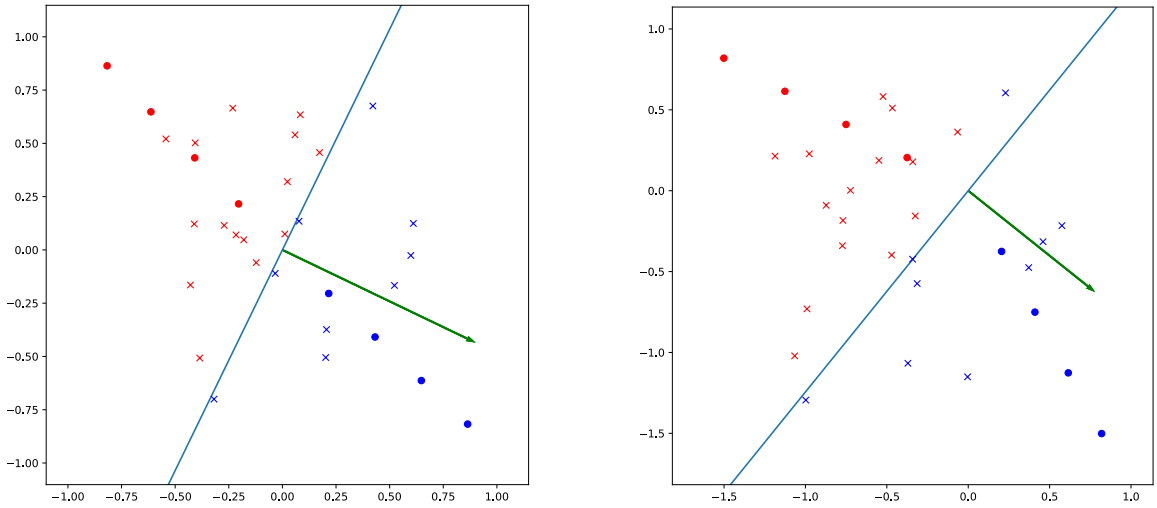
which can correctly provide the desired responses in the network (Equation 4.10 and Equation 4.11). For example, the output for $\mathbf{x}^1$ is:

$$\begin{aligned} z = \quad & 1.2255 \times (-1.5015 \times 1.00 + 0.8195 \times 0.00) + \\ & - 0.9849 \times (-1.5015 \times 0.00 + 0.8195 \times 1.00) \\ \hat{y} = & \operatorname{sgn}(z) = \operatorname{sgn}(-2.6472) = -1 \end{aligned} \tag{4.18}$$

In this small network, the Feature Map (shown in Figure 17) is both the feature map and the classification space. The extracted features are multiplied by the weight vector to produce the logits, which are then passed through a signal function to generate the network output (Equation 4.11). Therefore, the decision boundary here is a straight line that separates the data points of the two classes. More specifically, the decision boundary of the presented architecture (remember it does not have bias) is defined by the equation:

$$w_1\phi_1 + w_2\phi_2 = 0 \tag{4.19}$$

As it is a simple linear model, its feature map, classification space, unit vector of weights ($\hat{\mathbf{w}} = \mathbf{w}/||\mathbf{w}||$), and the decision boundary can be presented as in the Figure 20a for the Oracle and in the Figure 20b for the Copycat model. The circles are the features of the ODD images with their original labels and the crosses are the features of the NPDD images labeled by Oracle, where each color represents a different class. The red color is used to represent class one, which is labeled $-1$, so it must be below the decision boundary. And the blue color is class two, labeled $+1$, which must be above the decision threshold.




(a) (b)

Figure 20 – Representation of the feature map and classification space of the Oracle (a) and the Copycat (b). The green arrow is the unit vector of weights and the straight line is the decision boundary. The circles are the ODD features with their original labels and the crosses are the NPDD features labeled by Oracle, where each color represents a different class. Red color is used for class one and blue for class two.

As can be seen in the Figure 20a, the NPDD (crosses) can obtain information about the representation of the two classes within Oracle. Since the Copycat method only uses the hard-labels, this label extraction was able to get the necessary labels for both desired classes. So, the next step is training the Copycat model and find values for the convolutional maps and for the weights to represent correctly the crosses (NPDD-SL). After training, the Copycat model is expected to also correctly represent the problem domain data used to train Oracle. And as shown in Figure 20b, it does.

We repeated this experiment five times, starting each time with random values for the convolutional kernel, weights and NPDD. As presented in the Figure 21, the Copycats found different representations and weights. However, the parameters that each Copycat model found were sufficient to copy Oracle's predictions.

This problem consisted of finding the optimal combination of convolutional kernel values and weights to classify ODD images. However, due to the number of parameters involved, there are infinite possible combinations to be considered. Thus, an adversary can easily generate a surrogate model. In multi or hyper-dimensional problems, the input space can be very large and complex, making it difficult to find appropriate combinations of input pixels to obtain all desired Oracle classes. Thus, randomly generated input values may not be effective for extracting the model. Therefore, the Copycat method consists of using of random natural images to overcome this obstacle and investigate whether this vulnerability exists in image classification CNNs.

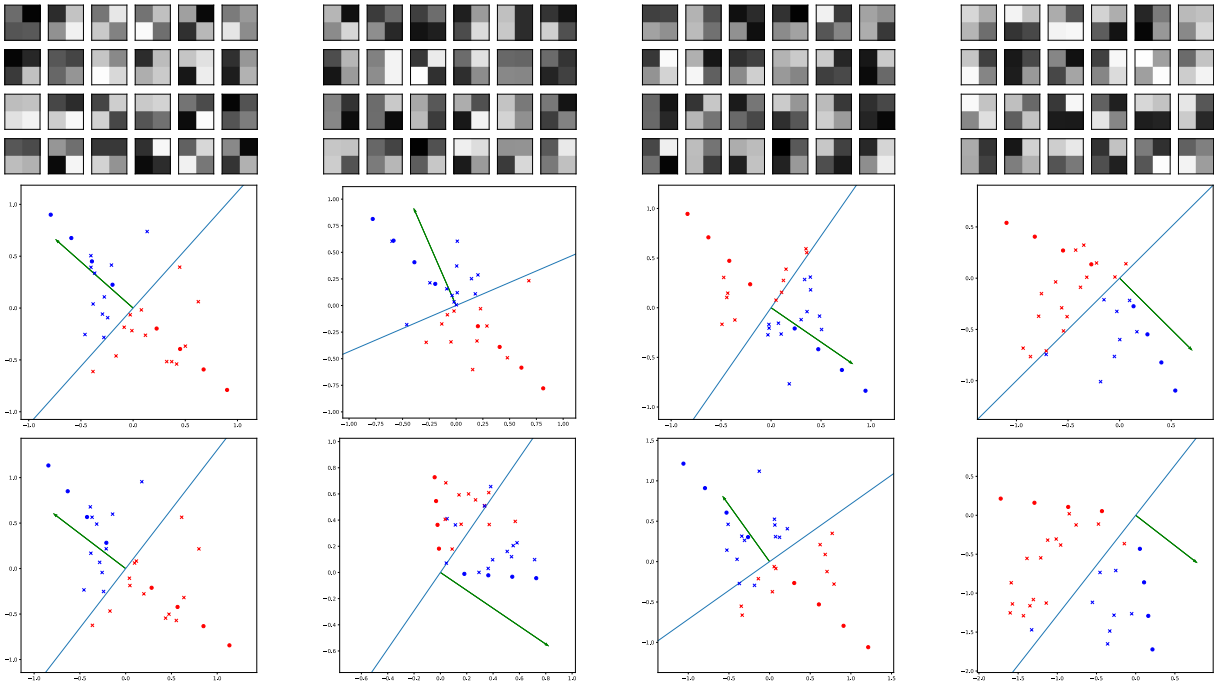


Figure 21 – Five experiments of the Copycat attack against the simple model. Each column represents one different experiment. The NPDDs are presented on the first line. The second and third lines are feature maps and classification spaces of the Oracles and Copycats, respectively.

The code for running this experiment is available in the Appendix A.

# 5 Experimental Methodology

In this chapter, the datasets used in the experiments are described. After that, the problems and architectures investigated are presented. Subsequently, the metrics used to measure the performance of the models are detailed. Moreover, the performed experiments for the model extraction and for defenses are described in detail. Finally, the experimental setup is presented.

The experiments were first conducted using Caffe (JIA et al., 2014). However, due to Caffe becoming obsolete over time and also to promote the future use, replicability, and study of our method, we built the Copycat Framework[1] in PyTorch (PASZKE et al., 2019) and performed additional experiments using it. This will be better explained throughout the text.

## 5.1 Datasets Organization and Baselines Setup

Several datasets were used to evaluate the proposed method. First, the datasets used for the baselines alongside the one used to test all the models are presented. The baseline refers to the models that are used as a reference for comparison with the surrogate model. Here we are considering two baseline models. The difference between them will be explained in the remainder of this section. Finally, the datasets used for the Copycat networks are described.

### 5.1.1 Datasets for Baselines and Tests

To evaluate the proposed approach, three dataset splits were generated for each problem: Original Domain Dataset (ODD), Problem Domain Dataset (PDD) and Test Domain Dataset (TDD). This split was performed for each one of the investigated problems taking into account datasets available in the literature. Specific details about each split of each problem are presented in Section 5.2.

The first dataset split is the Original Domain Dataset (ODD) that is used to train the Oracle. It represents the set of confidential data that is used to train the black-box model of interest. Note that no Copycat network has access to this dataset. The second dataset split is the Problem Domain Dataset (PDD), and it represents the dataset that adversaries may be able to build on their own. This dataset comes from the same domain of the ODD but might have less images. As the PDD tends to be small, an augmentation process can be used to increase the number of samples and to provide a larger and

[1] The Copycat framework can be accessed at <https://github.com/jeiks/Stealing_DL_Models>

balanced dataset for model training. This augmentation reflects what adversaries would do to increase the number of images they have. It is important to note that PDD and ODD do not share a single sample. Finally, the third dataset is the Test Domain Dataset (TDD) that is used for testing the performance of all models. It comprises images from the problem domain and does not overlap with the ODD and the PDD. None of the networks have access to samples of this dataset.

Given that we performed experiments on the PDD using both the Original Labels (OL) and the Stolen Labels (SL), which are predicted by (extracted from) the Oracle, we appended -SL and -OL on each dataset split as appropriate. Specifically, all the experiments with the ODD and TDD only used the OL, so they will be always referred to as ODD and TDD, respectively. On the other hand, depending on the experiment, the PDD-OL or the PDD-SL can be used depending on whether the original or stolen labels were used.

The Oracle is trained with ODD and is similar to the model provided by a company in a MLaaS API. Therefore, it is the first baseline, referred as Oracle, used to compare Copycat performance. However, the adversary has another problem domain dataset (PDD). This dataset with stolen labels (PDD-SL) is used in the Copycat method to fine-tune the surrogate model obtained in the first step of the model extraction. However, the adversary may have the original labels of PDD. Therefore, a second model can be trained with this problem domain dataset composed of its original labels (PDD-OL). This second model can answer the question whether the Copycat extraction method generates a surrogate model that outperforms a model trained with a cheaper problem domain dataset (PDD-OL). Therefore, this second model is used as the second baseline, referred as Baseline for brevity, to evaluate the surrogate model.

### 5.1.2 Datasets for Generating the Copycats

The dataset used to train the Copycats comprises images that are not from the problem domain (natural random images) labeled by the black-box model, hence Non-Problem Domain Dataset with Stolen Labels (NPDD-SL). Given that the acquisition of these images is assumed to be inexpensive and that only the hard-labels predicted by the black-box model are used, it is reasonable to assume that, to be effective, this dataset has a lot more samples than the ODD.

To achieve this, we utilized publicly available datasets that are accessible to any potential adversary. Therefore, based on the need for natural random images, two of the most well-known public datasets of images were used: the ImageNet (Visual Domain Decatlhon (REBUFFI; BILEN; VEDALDI, 2017) and the ILSVRC2017 (RUSSAKOVSKY et al., 2015b)) and Microsoft COCO (LIN et al., 2014). Duplicates were removed[2] after

[2] The fdupes application, which performs a bit-by-bit comparison, was used to remove duplicate images <https://github.com/adrianlopezroche/fdupes>.

merging all datasets, leaving 3,088,392 images left.

However, another NPDD was used for the Copycat Framework only to confirm the applicability of out method and to facilitate the reproducibility of the experiments. In this NPDD dataset, we used only the ImageNet images from the ILSVRC2012 (RUSSAKOVSKY et al., 2015b). Thus forming a NPDD with 1,270,431 images. The ILSVRC2012 dataset is smaller comparing to the first one and can be found in various locations on the internet, including through torrents.

For both datasets, all the original labels were discarded, since these dataset are always used with stolen labels. The number of images per class varies per problem, because it depends on the predictions of the target black-box model. After generating all image-label pairs, the dataset is balanced by oversampling the minority classes and under-sampling the majority classes.

## 5.2 Investigated Problems

In order to evaluate the proposed method with different real-world problems, seven problems were chosen: Human Action Classification Problem, 101 Categories (ACT101); Handwritten Digit Classification, 10 categories (DIG10); Facial Expression Recognition, 7 categories (FER7); General Object Detection, 9 categories (GOC9); Pedestrian Detection, 2 categories (PED2); Street View House Number Classification, 10 categories (SHN10); and Traffic Sign Classification, 30 categories (SIG30). An overview of public domain data used to generate the datasets and the image distribution between them are shown[3] on Table 3. In addition, the following subsections detail each of these problems, as well as their choices of corresponding datasets.

Table 3 – Details of the problems, their respective datasets, and the number of images in each domain splits. The "# PDD" column shows the number of original images for the Problem Domain Dataset, while the "# PDD*" column shows the number of images after applying image augmentation techniques.

| Problems | Datasets | # ODD | # TDD | # PDD | # PDD* |
|---|---|---|---|---|---|
| ACT101 | UCF101 | 1,782,858 | 685,824 | 32,428 | 101,104 |
| DIG10 | MNIST, NIST Special Database 19 | 60,000 | 10,000 | 55,000 | 55,000 |
| FER7 | AR Face, BU3DFE, JAFFE, MMI, RaFD, KDEF, CK+ | 55,629 | 1,236 | 980 | 65,660 |
| GOC9 | CIFAR-10, STL-10 | 45,000 | 9,000 | 11,700 | 269,100 |
| PED2 | DamPed | 23,520 | 5,880 | 19,600 | 19,600 |
| SHN10 | SVHN | 47,217 | 26,040 | 26,032 | 26,032 |
| SIG30 | TT100K, TSRD | 31,775 | 5,481 | 2,074 | 30,000 |

[3] The references for the datasets are presented throughout the following subsections.

### 5.2.1 Human Action Recognition – ACT101

The problem of Human Action Recognition covers the task of classifying human actions in the wild. The UCF101 (SOOMRO; ZAMIR; SHAH, 2012) is a popular and widely used dataset in the literature. It is composed of $13k$ videos (27 hours) with the presence of large variations in many aspects of the camera, objects, scene, etc., and the objective is to generate a classifier to recognize the action taken by the human being in 101 categories.

The authors of this dataset provide three recommendations to split the train and test datasets. The videos were separated into training and testing sets following the first recommendation. To be used in the experiments, all the video frames were converted into images. Two videos were selected at random from the training set and their images were used to generate the PDD. The remaining videos from the training set were used to generate the ODD. And all testing set images were used to generate the TDD. The categories of ODD with less than $10k$ images have been augmented. Similarly, all categories of PDD were augmented to $1k$ images.

### 5.2.2 Handwritten Digits Classification – DIG10

The problem of Handwritten Digits Classification is to recognize real-world handwritten digits from zero to nine. It is a classic problem of literature. The MNIST (LeCun et al., 1998) dataset (originally generated from NIST Special Database 3 and 1) was used to generate the ODD ($60k$ images) and TDD ($10k$ images). Furthermore, $5.5k$ images per class were randomly selected from the NIST Special Database 19 (GROTHER; FLANAGAN, 1995) to generate the PDD. The selected images were converted to the same format as the MNIST images.

### 5.2.3 Facial Expression Recognition – FER7

The complexity of Facial Expression Recognition problem comes from the human diversity and the features that must be considered to recognize a facial expression. The most common setting considers the seven basic universal expressions: fear, sad, angry, disgust, surprise, and happy. In addition, and in this work, the neutral expression is also considered. For this experiment, several well-known datasets were used to generate the ODD (55629 images): AR Face (MARTINEZ; BENAVENTE, 1998), Binghamton University 3D Facial Expression (BU3DFE) (YIN et al., 2006), The Japanese Female Facial Expression (JAFFE) (LYONS et al., 1998), MMI (PANTIC et al., 2005) and Radboud Faces Database (RaFD) (LANGNER et al., 2010). Additionally, the Karolinska Directed Emotional Faces (KDEF) (LUNDQVIST; FLYKT; ÖHMAN, 1998) was used to generate the PDD (980 images). Finally, the Extended Cohn-Kanade Database (CK+) (LUCEY et

al., 2010) was used to generate the TDD (1236 images). The augmentation process was the same used in (ZAVAREZ; BERRIEL; OLIVEIRA-SANTOS, 2017) and all databases images were converted to grayscale.

### 5.2.4 General-Object Classification – GOC9

The problem of General-Object Classification consists of recognizing different kind of objects in the wild. The CIFAR-10 (KRIZHEVSKY; HINTON, 2009) is a established dataset used in computer-vision and it was used to generate ODD (45000 images) and TDD (9000 images). A similar dataset, STL-10 (COATES; NG; LEE, 2011), was used to generate PDD (11700 images). Both datasets have 10 categories, but they did not share the same categories. Therefore, to merge the datasets, one category from each one was removed ("frog" from CIFAR-10 and "monkey" from STL-10), resulting in 9 categories for this problem.

### 5.2.5 Pedestrian Classification – PED2

The Pedestrian Classification is a problem to detect the existence of a pedestrian in an scene. The Daimler Pedestrian Classification (MUNDER; GAVRILA, 2006) dataset used in the Visual Domain Decathlon Challenge (REBUFFI; BILEN; VEDALDI, 2017) already had the images pre-processed and ready for use, so it was used in the experiments. The images suggested for the training set were used to generate the ODD (23520 images). Unfortunately, the images suggested for the testing set have not been labeled. For that reason, the images for the validation set were used to generate the TDD (5880 images) and the images for the testing set were used to generate the PDD (19600 images). Consequently, only the Problem Domain Dataset with Stolen Labels (PDD-SL) was generated for this problem and its Baseline was not generated.

### 5.2.6 Street View House Numbers Classification – SHN10

The problem of Street View House Numbers Classification is similar to Handwritten Digits Classification, but it differs because the digits come from pictures of house numbers. The most commonly used dataset for this problem is the SVHN (NETZER et al., 2011), composed by 10 classes. Similarly to PED2, the images pre-processed by Visual Domain Decathlon were used in the experiments. The images suggested for the training set were used to generate the ODD (47217). Likewise PED2, the images suggested for the testing set have not been labeled. Therefore, the validation set was used to generate the TDD (26040 images), the testing set was used to generate the PDD (26032 images), and the Baseline was not generated for this problem.

### 5.2.7 Traffic Signs Classification – SIG30

The problem of Traffic Signs Classification is to identify which sign was detected in the image. Two Chinese traffic signals datasets were chosen to the experiments: TT100K (LU et al., 2018) and TSRD (HUANG; YIN, 2014). To create a unified dataset, the class labels of the TSRD and TT100K images were manually synchronized by assigning a unique class number to each category across both datasets. This ensured that only the common categories between the two datasets were maintained in the final dataset. Additionally, to avoid using classes with too few images, categories with less than 26 images in TT100K were discarded, leaving a total of 30 categories. The suggested training and testing set of TT100K were used to generate the ODD (31775 images) and TDD (5481 images), respectively. Additionally, the TSRD was used to generate the PDD (2074 images). All categories with less than 1000 images were augmented.

## 5.3 Used Architectures

The proposed approach was evaluated in two conditions regarding the network architecture: generating Copycats from the same architecture or from different architectures. Two well-known architectures were chosen for this experiments: the VGG-16 (SIMONYAN; ZISSERMAN, 2015) (Figure 10) and the AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) (Figure 9). When we conducted the initial experiments, the VGG-16 was a quiet new architecture that achieved high performance across all the problems we investigated. It had support in Caffe, and its computational requirements were met by the computers we had for executing the experiments. Whereas the AlexNet was chosen because it was a network similar to VGG-16 but with fewer parameters and with a known limited learning capacity. In addition to the previously mentioned architectures, LeNet and ResNet-18 were also used in defense experiments to replicate and compare the the defense methods against Copycat.

## 5.4 Metric

To evaluate the proposed method, quantitative and qualitative metrics were used. According to the taxonomy proposed by Jagielski et al. (2020), there are two objectives in model extraction: accuracy, *i. e.*, achieving high success rates in the underlying task, and fidelity, *i. e.*, matching the predictions of the surrogate model with the Oracle.

Our objective in model extraction with Copycat CNN was to exploit as much of this extraction as possible. Thus, both high accuracy and fidelity are considered good results in our experiments. Therefore, a comparative metric that we found ideal was the macro average F1-Score (or Macro F1-Score, or F1-Macro Score), because it can be used

to evaluate the performance of the model based on its precision and recall. The formula for F1-Score in multi-class problems is given by:

$$\text{F1-Score} = \frac{1}{d}\sum_{i=1}^{d} F1_i \tag{5.1}$$

where $d$ is the total number of classes and $F1_i$ is the F1-score for class $i$, which is given by:

$$F1_i = 2 \times \frac{precision_i \times recall_i}{precision_i + recall_i} \tag{5.2}$$

where $precision_i$ is the proportion of elements with correct prediction for class $i$ and $recall_i$ is the proportion of correct predictions for class $i$, which are given by:

$$precision_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \tag{5.3}$$

$$recall_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{5.4}$$

where $\text{TP}_i$ is the true positive count for class $i$, $\text{FP}_i$ is the false positive count for class $i$ and $\text{FN}_i$ is the false negative count for class $i$.

Since data imbalance may exist in a dataset, this metric is useful to handle this peculiarity. It computes the simple average over the classes independently and then calculates the average, hence providing a measure of how well the model is performing across all classes. Furthermore, the ratio between the Copycat F1-Score over the Oracle $\left(\frac{\text{F1-Score}(Copycat)}{\text{F1-Score}(Oracle)}\right)$ and over the Baseline $\left(\frac{\text{F1-Score}(Copycat)}{\text{F1-Score}(Baseline)}\right)$ are also used to measure the network performance. The perfect Copycat network should achieve a macro-averaged F1-Score identical to the Oracle (i.e., it should hit and miss the same images as the Oracle) and achieve 100% of copy performance over the Oracle. Therefore, the performance of Copycat was measured by calculating the F1-Score of Copycat and the F1-Score of the Oracle, and then taking the ratio of them:

Additionally, the Layer-wise Relevance Propagation was chosen for a qualitative analysis. It explains individual neural network predictions in terms of input variables. The LRP is a method for identifying important pixels that contributed most strongly to the image classification. By using LRP, both Oracle and Copycat heatmaps can be obtained from the same input image. After, the heatmaps of both networks can be compared to verify their similarity. We believe that, in general, the same pixels responsible for the prediction of the Oracle will be highlighted by the Copycat network.

Furthermore, in order to compare the heatmaps generated by the different models, we used Pearson correlation, which is a method commonly used to assess the similarity between two two continuous univariate random variables (FACELI et al., 2011). The

Pearson correlation is given by:

$$\begin{aligned} r(a,b) &= \frac{\text{covariance}(a,b)}{\sigma_a \sigma_b} \\ &= \frac{\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n}(a_i - \bar{a})^2 \sum_{i=1}^{n}(b_i - \bar{b})^2}} \end{aligned} \tag{5.5}$$

where $r$ is the Pearson correlation coefficient, $n$ is the number of observations, $a_i$ is the value of the $i^{th}$ observation of variable $a$, $\bar{a}$ is the mean of variable $a$, $b_i$ is the value of the $i^{th}$ observation of variable $b$, and $\bar{b}$ is mean of variable $b$.

The Pearson correlation is sensitive to outliers and its value varies within the interval $[-1, 1]$. The magnitude of its values indicates the strength of the correlation between two objects and the sign indicates the direction. Thus, values close to $-1$ or 1 indicate the similarity between the objects, while 0 indicates dissimilarity. In the case of vectors, a correlation of 1 indicates that the vectors are parallel and point in the same direction. A correlation of $-1$ indicates that the vectors are parallel and point in opposite directions. And the correlation of 0 indicates that the vectors are orthogonal. Therefore, this correlation can be used to compare the similarity of the flattened heatmap matrices.

## 5.5 General Setup

All models were trained using Stochastic Gradient Descent (SGD) with a Step Down policy for the learning rate. The maximum of five epochs was chosen because the models empirically showed convergence after it. However, some models lasted for more than five epochs, until their losses reached a plateau. In the augmentation process, the following operations were applied[4]: add/sub intensity, contrast normalization, crop, horizontal flip, Gaussian blur, Gaussian noise, piecewise affine, rotate, scale, sharpen, shear, and translate. Code and experiment details are publicly released[5].

## 5.6 Model Extraction Experiments

A set of seven experiments are performed to evaluate the proposed method: (i) Analysis of Datasets Distributions in the Classification Space of the Black-Box Model, (ii) Copycat from the Same Architecture, (iii) Analysis of the Relationship Between Number of Queries and Copycat Performance, (iv) Copycat from a Different Architecture, (v) Robustness of the Copycat Model, (vi) Analysis of the Attention-Region in the Input Images, and (vii) Analysis of Attack Viability and APIs Costs. Each of these experiments is explained in details below.

[4] Tool used for the augmentation process: https://github.com/aleju/imgaug

[5] https://github.com/jeiks/Stealing_DL_Models

### 5.6.1 Analysis of Datasets Distributions in the Classification Space of the Black-Box Model

The proposed method aims to copy the Oracle using the knowledge extracted from its labels. However, the Copycat's performance is directly related to the details provided by the Oracle about its internal knowledge, i.e., its feature and classification spaces that were learned during the training. According to Nguyen, Yosinski and Clune (2015), Papernot et al. (2017), and other works on adversarial examples, there are several gaps[6] that were not filled by the training data in the classification space of the Oracle. This characteristic can end up facilitating and allowing model extraction attacks on DNNs. Therefore, this experiment attempted to visually analyze whether the NPDD-SL images belong to the same classification subspaces as the ODD images when processed by the Oracle.

In this direction, we extracted the vector generated by the ReLU layer before the logits in the VGG architecture. Due to the high dimensionality of 4096, we used T-SNE to generate a 2-D view. To reduce complexity when running T-SNE, we randomly selected a few examples to perform this experiment: 100 for each class of the ODD. For each ODD sample, three neighbors of the same class of NPDD-SL were selected. When a neighbor has already been selected by another point, it is ignored and the next one is selected. For better visualization and based on the results discussed further in Subsection 6.1.3, the neighbors were selected from a random subset of $200k$ samples of NPDD-SL. Furthermore, before applying T-SNE, the ODD and NPDD-SL vectors were standardized using, respectively, the ODD and the NPDD-SL means and standard deviations.

Two problems were analyzed and the expectation was that several NPDD-SL samples would be in the same subspaces that ODD samples. If that's the case, we consider them to be in the same subspace within the model.

### 5.6.2 Copycat from the Same Architecture

In this experiment, the purpose was test the Copycat method by copying the Oracle using the same architecture (VGG). Initially, the Oracle was trained with ODD and the Baseline was trained with PDD-OL. Then, the Oracle was used as a black-box and only receives an image as input, providing a hard-label as output. At this point, all NPDD and PDD images were queried to the Oracle and their labels were saved to create the fake datasets NPDD-SL and PDD-SL. Next, one Copycat was trained with NPDD-SL and another with PDD-SL. Additionally, the first one was fine-tuned with PDD-SL, generating the Copycat NPDD+PDD-SL. Finally, all networks were evaluated using the TDD-OL and their F1-Scores were compared. This process was replicated for all problems, with the expectation that Copycat would achieve high performance on Oracle for each problem.

[6] "Gaps" is the same term used by the referred authors.

The Copycats receive an acronym in the format: [CC]-[Architecture]-[Dataset]. Therefore, the Copycats are: CC-VGG-NPDD-SL, CC-VGG-PDD-SL, CC-VGG-NPDD+PDD-SL.

This experiment was partially replicated in Copycat Framework in PyTorch. For the reproducibility, the NPDD used was composed of ILSVRC2012 images. And due to the computational cost, not all NPDD images were used in the training process. Experiments from Subsection 6.1.3 showed that the Oracle can be extracted with fewer images compared to our initial experiments. Therefore, the methodology used in these experiments involved using equal batches of the curvature (as described in more detail in Subsection 6.1.3) until Copycat achieved high performance ($> 90\%$). Thus, these experiments will be discussed in the next section.

### 5.6.3 Analysis of the Relationship Between Number of Queries and Copycat Performance

Measuring the number of queries required to extract a model with the Copycat method is important, as it provides insights into the ease of model extraction (indicated by a lower number of queries) and directly influences the budget allocated for the extraction process. Therefore, this experiment analyzed the differences among copies of the Oracle with different amounts of natural random images. For this purpose, datasets with different sizes were generated with random images from NPDD-SL. For each problem in the first set of experiments (using Caffe), Copycat networks were trained with NPDD-SL comprising $100k$, $500k$, $1M$, $1.5M$, and $3M$ images. After training, these networks were fine-tuned with PDD-SL. Finally, all the two versions of the networks (with and without fine-tuning) are evaluated with TDD-OL and their F1-Scores are used to create a data curve.

An additional experiment was performed with Copycat Framework in PyTorch. However, the Copycat networks were only trained until they achieved performance greater than 90%. Therefore, for each problem, Copycat networks were trained with NPDD-SL (composed of ILVSRC2012 images) comprising $100k$ and $500k$ images. The experiments with $100k$ images (Oracle, Copycat and fine-tuned models) were performed three times. After that, the Oracles with the best F1-Scores on Test Domain Dataset with Original Labels (TDD-OL) were selected to be presented here. The same Oracles were also used for experiments with $500k$ images, which were also conducted three times.

### 5.6.4 Copycat from a Different Architecture

As with model compression methods, we seek to apply the Copycat method to copy knowledge from a larger network to a smaller network, *i. e.*, with few parameters. For this purpose, a smaller and briefly different architecture was chosen to train Copycat on: AlexNet (Figure 9). In this experiment, the Copycat network was trained with the

same NPDD-SL used by CC-VGG-NPDD-SL, *i. e.*, the same pairs of image and stolen label obtained by the Oracle for the other Copycat experiments. Since the Copycat may achieve a lower F1-Score due to the architecture capacity and not because of the copy method itself, an alternate baseline, referred as AlexNet Baseline, was also trained with the same data (ODD) used in the Oracle.

### 5.6.5 Robustness of the Copycat Model

A neural network initiates the training with random parameters and the samples are also randomly selected, so two or more models trained with the same architecture and the same dataset can achieve different results depending on the initialization procedure. Since the performance may be different for each training, the Copycat method was evaluated on three different Oracles trained with the same problem domain images. For each one, the weights were initialized with different values and the images were shuffled on training. Then, the proposed method is applied to copy each model and the robustness of the Copycat against the random factors of the black-box model training was analyzed.

This initial experiment was performed on the FER7 using the VGG architecture in Caffe framework. In this experiment, three target networks are generated using the ODD with one Copycat being trained (using $3M$ of NPDD-SL) for each of them. Finally, all networks were evaluated using the TDD-OL and analyzed by comparing their performance variation. Additionally, this process was replicated in PyTorch Copycat framework but only using $100k$ and $500k$ of NPDD-SL. For all experiments here, the expectation is also that the three Copycats perform similarly. The mean and standard deviation of such results are used as a metric.

### 5.6.6 Analysis of the Attention-Region in the Input Images

To better understand the models and also to analyze the results of the proposed method, a visual comparison of the input image regions that influence the model predictions was performed between Copycats and Oracles. For this process, the Layer-wise Relevance Propagation was used. After applying LRP on different images, the heatmaps of the following predictions of the Oracle and Copycat were compared: correct vs. correct, correct vs. wrong, and wrong vs. correct. The LRP toolbox Caffe[7] was used in these experiments and the selected composition rules were LRP-$\alpha\beta$ and LRP-$\epsilon$. These configuration provided a better visualization of the heatmaps. To analyze the differences between the Copycat on different amounts of data, the heatmaps are also generated to Copycats used in the data curve (Subsection 6.1.3). An image with the same prediction by the Oracle and Copycat

[7] Due to differences between the LRP toolbox Caffe and NVIDIA Caffe, only images that provided the same predictions in both frameworks were used. LRP toolbox is available at: <http://www.heatmapping.org/lrptoolbox.html>

is expected to present similar heatmaps for both networks.

Additionally, the LRP experiments were also performed in the Copycat Framework in PyTorch. In these experiments, we follow the recommendation of Kohlbrenner et al. (2020) and used the composition rules $\alpha = 2, \beta = 1$, and $\flat$ (flat). They suggest this composition to produce heatmaps that are measurably more representative, more robust against gradient shattering and with better properties for classification models. There are a few options of tools written in PyTorch that could be used to apply the LRP. Therefore, after testing some ready-made tools for this, we opted to use Zennit (ANDERS et al., 2021), which proved to be more complete and robust to use with the Copycat Framework.

### 5.6.7 Analysis of Attack Viability and APIs Costs

In real-world, consuming an API has a per-use cost, which implies that executing an attack as the one formulated in our work will cost money for the adversary. Moreover, for the attack to be successful, one would only consider copies with a certain level of quality. Therefore, we analyzed the viability of a realistic attack by measuring the costs associated with different number of queries and their corresponding copy performances. For this experiment, the Emotion API from Microsoft Azure Cognitive Services[8] was used and the performances were measured by the F1-Score on the Test Domain Dataset (TDD). Both TDD and the number of expressions were the same as those used in FER7.

Given that an attack might be viable, we also analyzed how much an API should charge to inhibit this type of attack. From the adversary's perspective, an attack is viable when the costs of executing it are lower than acquiring and annotating data of the problem of interest. In this analysis, the price for annotating the data was based on the Data Labeling Service from Google Cloud[9], the amount of data was based on the ODD of each problem, and we considered an attack to be successful with a copy performance greater than 90%.

## 5.7 Defense Experiments

As our experiments with the Copycat method achieved high performance in extracting the proposed problems, we looked for existing defenses that could provide security against the risks associated with this vulnerability. In particular, among a range of defenses proposed in the literature (ZHANG et al., 2022), we analyzed various methods for defending CNNs against model extraction attacks. Thus, the following three works were chosen: ADMIS (KARIYAPPA; QURESHI, 2020) and PRADA (JUUTI et al., 2019),

[8] https://azure.microsoft.com/en-us/pricing/details/cognitive-services/emotion-api/ (accessed in Feb, 2023)

[9] https://cloud.google.com/vertex-ai/pricing#labeling (accessed in Feb, 2023)

which defend model extraction attacks by detecting malicious queries, and EWE (JIA; CHOQUETTE-CHOO; PAPERNOT, 2021), which defends the Intellectual Property of the model using watermarks. In this section, we will provide a comprehensive overview of the works studied, including a detailed description of their experimental methodology.

### 5.7.1 ADMIS: Adaptive Missinformation

Adaptive Misinformation (ADMIS) (KARIYAPPA; QURESHI, 2020) is a method to protect black-box CNNs from model extraction attacks. The ADMIS authors observed that extraction attacks use out-of-distribution (OOD[10]) queries to extract knowledge from the Oracle to generate a surrogate model. Therefore, the proposed defense is to detect out-of-distribution queries and provide wrong answers to deceive the adversary. So, the answers of an ADMIS protected model can be provided by the protected model itself (Oracle) for in-distribution (ID) queries, or by the poison model for detected out-of-distribution queries. The authors argued that their method impacts accuracy by less than 0.5% for ID queries.

To detect out-of-distribution queries, ADMIS analyzes the Maximum Softmax Probabilities (MSP) of the Oracle, which is the highest softmax probability of the model for each query. However, as some models may not provide an evident MSP (e.g., $\hat{y} = [0.41, 0.39, ...]$ instead of $\hat{y} = [0.90, 0.05, ...]$), the authors proposed using the Outlier Exposure (OE) technique (HENDRYCKS; MAZEIKA; DIETTERICH, 2019) to train the models and emphasize MSP values.

The OE technique involves using an out-of-distribution dataset to train the model and produce a uniform probability distribution that distinguishes anomalies from in-distribution (ID) queries. After training, ADMIS uses an out-of-distribution detector (OOD Detector) to compare the MSP to a threshold and detect whether the query is an attack. If the MSP is greater than the threshold, the query is considered out-of-distribution, *i. e.*, an attack, and the answer is provided by the poison model instead of the Oracle. To let it clear, see the illustrative block diagram presented in Figure 17. The query $x$ feeds the two models and the OOD Detector compares the Oracle MSP with the threshold $\tau$ to select between the Oracle or poison outputs. In the ADMIS denfense method, the Oracle is trained to generate a uniform probability distribution on inputs from OE dataset. It is done by adding an extra term $\mathcal{L}_2$ in this loss function:

$$f_{\theta^\star} = \arg\min_{\theta} \mathcal{L}_1(f_\theta(x), y) + \mathcal{L}_2(f_\theta(x); f_\theta(\tilde{x})) \tag{5.6}$$

where the training dataset is $D_{in} = \{(x_i, y_i)\}_{i=1}^{M}$, the OE dataset is $D_{out} = \{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^{N}$,

[10] Due to the similarity between OOD and ODD (Original Domain Dataset), the abbreviation will only be used in some places in the text.

the $\mathcal{L}_1$ is the mean of Cross-Entropy Loss and the $\mathcal{L}_2$ is defined by:

$$\mathcal{L}_2(f_\theta(x); f_\theta(\tilde{x})) = -\frac{1}{N+M} \sum_{i=1}^{N+M} \left( \bar{z}_i - \log \sum_{j=1}^{K} \exp(z_{ij}) \right) \tag{5.7}$$

where $z = f_\theta(\cdot)$ is the vector of logits with size $K$ and $\bar{z}_i$ is its $i$-th element's mean. Additionally, the poison model is trained with the following loss function:

$$g_{\theta^\star} = \arg\min_\theta \mathcal{L}_1 \left( \log(1 - f_\theta(x'_i)), y'_i \right) \tag{5.8}$$

where $\{x'_i\}_{i=1}^{M+N} = D_{in} \cup D_{out}$. And the final output probabilities of the system (OOD Detector) is provided by the combination of $f$ and $g$:

$$\hat{y} = (1 - \alpha) f(x; \theta) + \alpha g(x; \theta) \tag{5.9}$$

where $\alpha = \frac{1}{1+e^{v\beta}}$, with $\beta = \max_i(\hat{y}_i) - \tau$, $\tau$ as the threshold, and $v = 10000$ for all experiments[11].

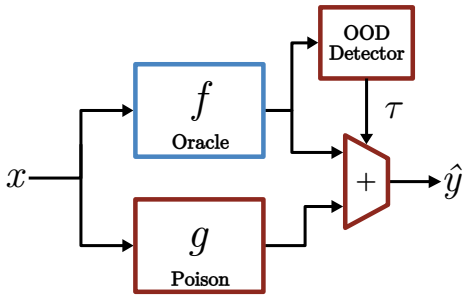



Figure 22 – Illustrative Block Diagram of Adaptive Misinformation: after training the Oracle $f$ and the poison model $g$, the system receives the query $x$ and the OOD detector compares the Oracle's MSP with the threshold $\tau$. Then, it selects the outputs from $f$ for ID queries or $g$ for OOD queries (adapted from Kariyappa and Qureshi (2020)).

In their work, the authors trained the following four problems (Table 4):

- MNIST using LeNet architecture,
- FashionMNIST (XIAO; RASUL; VOLLGRAF, 2017) using LeNet architecture,
- CIFAR10 using ResNet-18 architecture, and
- Flowers17 (NILSBACK; ZISSERMAN, 2006) using ResNet-18 architecture.

And the following datasets were used as OE training:

- KMNIST (CLANUWAT et al., 2018) for MNIST,
- KMNIST (CLANUWAT et al., 2018) for FashionMNIST,
- ImageNet for CIFAR10, and
- Indoor67 (QUATTONI; TORRALBA, 2009) for Flowers17.

[11] Note: these formulas were derived not only from the article but also from the provided code, which

However, the authors of the OE technique indicate the use of out-of-distribution datasets, but the ADMIS authors did not remove the ImageNet classes that may intersect CIFAR10 dataset.

Table 4 – Datasets and architectures used in experiments.

| **Dataset** | **OE Dataset** | **Models architectures** | **ADMIS attack dataset** | **PDD Dataset** |
|---|---|---|---|---|
| MNIST | KMNIST | LeNet | EMNISTLetters | NIST19 |
| FashionMNIST | KMNIST | LeNet | EMNIST | – |
| CIFAR10 | ImageNet | ResNet-18 | CIFAR100 | STL9FROG |
| Flowers17 | Indoor67 | ResNet-18 | ImageNet (50k) | Flowers17PD |

For evaluation of ADMIS, they used the KnockOff[12] (OREKONDY; SCHIELE; FRITZ, 2019) and Jacobian Based Dataset Augmentation (JDBA) (PAPERNOT et al., 2017) attacks. After, they compared ADMIS with the Prediction Poisoning (PP) defense (OREKONDY; SCHIELE; FRITZ, 2020). Then, the comparison between the ADMIS and PP defenses was presented in ROC curves varying the threshold in $[0.0, \ldots, 0.99]$. However, it was not replicated here.

Before continuing, please consider the following terms for the remainder of this text:

- The ADMIS defense is based on two models (Oracle and poison) and the OOD detector. Therefore, the term "system" will be used as a reference to it.
- The term protected will be used when the system defense has a threshold greater than zero and unprotected otherwise.
- As the Rejection Ratio refers to the identified out-of-distribution queries, the same metric is called the Detection Ratio in the attack experiments.

To Evaluate the effectiveness of ADMIS defense against Copycat attacks, their models were reproduced, i.e., they were trained with the proposed datasets on the proposed architectures until reaching the same accuracy reported in their paper. Overall, our methodology for analyzing the ADMIS consists of the following steps:

1. train the Oracle and Poison models with their datasets:
    - 80% of training dataset (Table 4, column 1) was used for training the Oracle;
    - the OE dataset was used for training the poison model (Table 4, column 2);
2. select a threshold to out-of-distribution detector (OOD Detector in Figure 22 or $\tau$ in Equation 5.9):

differs slightly from the annotations in the article.

[12] This method is similar to our method, but uses soft-labels instead of hard-labels and only a NPDD dataset. They even tested this method using only hard-labels, but differently from Copycat, they did

- to define the threshold, a random subset comprising 20% of the training dataset was set aside and not used in the Oracle training process. This subset is referred to as the Auxiliary Dataset.

3. verify the Rejection Ratio on problem domain datasets (training, testing and auxiliary) for future comparisons.
4. perform Copycat attack against both the protected and unprotected systems:
   - the first set of attacks was performed using the datasets already proposed by authors as the attack dataset (NPDDs).
   - the second set of attacks was performed using the NPDD dataset already used in our experiments with Copycat framework.
5. verify the Detection Ratio for the attack datasets.
6. compare the Detection Ratio of the protected and unprotected systems, as well as compare them to the Rejection Ratio, to evaluate the effectiveness of the defense.
7. evaluate the performance of the Copycat attacks on both the protected and unprotected systems.

The Copycat model can be fine-tuned with a PDD dataset in addition to the NPDD dataset. However, a similar dataset was not available for FashionMNIST, so fine-tuning on this problem was not performed in the experiments. Thus, the Problem Domain Datasets used for the other problems were the following:

- MNIST: NIST19, same PDD used for DIG10 problem (Subsection 5.2.2);
- CIFAR10: STL9 (same PDD used for GOC9 problem, Subsection 5.2.4) and a thousand frog images from ImageNet[13].
- Flowers17: all samples were collected from Google Images[14]. The maximum number of images provided on Google Search was downloaded for each class (Table 5).

And we opted to train all Copycat models on ResNet-18 architecture.

For each problem, a threshold was selected to protect the model respecting the impact accuracy of ID queries up to 0.5%. After the training process, the system was tested on Auxiliary Dataset, varying the threshold in range of $[0.00, \ldots, 0.99]$. As the ADMIS authors argued the impact accuracy was less than 0.5%, we selected a threshold that produced a Rejection Ratio near to 0.5% on Auxiliary Dataset, *i. e.*, only 0.5% of ID queries was detected as out-of-distribution queries. However, in cases where there was not find a threshold for a Rejection Ratio exactly to 0.5%, we selected the closest higher Rejection Ratio. Note that when the threshold is higher, more queries are detected as

not balance the datasets and did not use a PDD dataset for fine-tuning the surrogate model.

[13] These images are not part of the NPDD datasets used to attack this problem.

[14] We used a JavaScript code to collect the links of the images from Google Images <https://images.google.com> and the *wget* <https://www.gnu.org/software/wget> program to download them.

Table 5 – Number of images per class collected from the Google Images to generate the Flowers17 PDD.

| Class | Name | Number of images | Class | Name | Number of images |
|---|---|---|---|---|---|
| 1 | Bluebell | 641 | 10 | Iris | 681 |
| 2 | Buttercup | 608 | 11 | Lily Valley | 710 |
| 3 | Colts Foot | 733 | 12 | Pansy | 646 |
| 4 | Cowslip | 678 | 13 | Snowdrop | 647 |
| 5 | Crocus | 789 | 14 | Sunflower | 657 |
| 6 | Daffodil | 721 | 15 | Tigerlily | 732 |
| 7 | Daisy | 665 | 16 | Tulip | 685 |
| 8 | Dandelalion | 625 | 17 | Windflower | 729 |
| 9 | Fritillary | 721 | | | |

out-of-distribution. Thus, the attack is expected to be more difficult to carry out.

The first set of attacks replicated the methodology used in ADMIS by randomly selecting $50k$ images from the ImageNet dataset to attack the Flowers17 problem. This approach ensured consistency with their original work (KARIYAPPA; QURESHI, 2020). In the second set of attacks, three different sizes of NPDD datasets ($100k$, $300k$, and $500k$) were used to attack each problem. By varying the size of the NPDD dataset, we could investigate how the performance of the defense method changed with respect to the amount of data used to attack the system. Furthermore, we also performed attacks on CIFAR10 problem using only out-of-distribution images, *i. e.*, we used for OE Exposure only ImageNet classes that are not present in CIFAR10. These experiments are referred as "CIFAR10 (OOD)". Moreover, as the code provided by the authors[15] only worked for the MNIST experiment, we developed the missing code needed to run all experiments.

### 5.7.2 PRADA

PRADA (KARIYAPPA; QURESHI, 2020) is a method to detect extraction attacks on Convolutional Neural Networks. It works on a batch of queries detecting out-of-distribution images. The PRADA detection hypothesis is that the distance between two benign images[16] randomly selected from the model space will fit a normal (Gaussian) distribution, hence images with distances outside this distribution are considered as an attack.

The detection process consists in gathering a batch of image queries, separate them in groups based on the Oracle's prediction and calculate the L2 distance between the

[15] <https://github.com/sanjaykariyappa/adaptive_misinformation>

[16] The authors of PRADA use the term benign queries to refer to in-distribution queries. We used the same term here in the sections related to PRADA.

images of the same group. After gathering a batch of queries, the Shapiro-Wilk normality test is used to identify if the image fits in the normal distribution on the model space. When the statistic test value returned by Shapiro-Wilk test is lower than a threshold, the query is considered an attack.

To define this threshold, the system receives several groups generated from two different problem domain datasets, treating them as benign queries. Their statistical test values are then collected to determine the appropriate threshold[17]. After, a threshold is selected to avoid false positives, i.e., a value lower than the lowest statistical test value obtained from benign queries is selected as the threshold. Finally, when an attack is detected, the three highest probabilities of the output are scrambled and provided as the model's response. The entire process is presented in Figure 23. However, on the provided code[18], the conditional bellow line 20 is executed only when the number of queries processed is divisible by ten.

1: Let $F$ denote the target model, $S$ a stream of samples queried by a given client, $D$ the set of minimum distances $d_{min}$ for samples in $S$, $G_c$ the growing set for class $c$, $D_{Gc}$ the set of minimum distances $d_{min}$ for samples in $G_c$, $T_c$ the threshold value for class $c$, $\delta$ the detection threshold.
2: $D \leftarrow \emptyset$, $G_c \leftarrow \emptyset$, $D_{Gc} \leftarrow \emptyset$, $attack \leftarrow false$
3: **for** $x : x \in S$ **do**
4: $c \leftarrow F(x)$
5: **if** $G_c == \emptyset$ **then** # sets and threshold initialization
6: $G_c \cup \{x\}$, $D_{Gc} \cup \{0\}$, $T_c \leftarrow 0$
7: **else**
8: $d \leftarrow \emptyset$
9: **for all** $y : y \in G_c$ **do** # pairwise distance
10: $d \cup \{dist(y, x)\}$
11: **end for**
12: $d_{min} \leftarrow min(d)$ # distance to closest element
13: $D \cup \{d_{min}\}$ # add distance to $D$
14: **if** $d_{min} > T_c$ **then** # sets and threshold update
15: $G_c \cup \{x\}$
16: $D_{Gc} \cup \{d_{min}\}$
17: $T_c \leftarrow max(T_c, \overline{D_{Gc}} - std(D_{Gc}))$
18: **end if**
19: **end if**
20: **if** $|D| > 100$ **then** # analyze distribution for $D$
21: $D' \leftarrow \{z \in D, z \in \langle \overline{D} \pm 3 \times std(D) \rangle\}$
22: **if** $W(D') < \delta$ **then** # attack detection test
23: $attack \leftarrow True$
24: **else**
25: $attack \leftarrow False$
26: **end if**
27: **end if**
28: **end for**

Figure 23 – Algorithm for PRADA's detection of model extraction. Source: (KARIYAPPA; QURESHI, 2020).

[17] They also used a group of images with uniformly random pixel values, but only to demonstrate that their system is independent of a specific data distribution. Therefore, this test was not performed in our experiments.

[18] <https://github.com/SSGAalto/prada-protecting-against-dnn-model-stealing-attacks>

They also stated that their work is generic and can be applied to any type of input data and learning algorithm. And since they believed that the adversary would not have a large set of natural images to use in the extraction attack, they used adversarial attack methods (TRAMÈR et al., 2016; PAPERNOT et al., 2017) to create malicious queries and test their work.

In their work, the first Oracle was trained on the MNIST dataset and the test subset was used for benign queries, along with the USPS dataset (HULL, 1994). The second model was trained on the GTSRB dataset (STALLKAMP et al., 2011) and the test subset was used for benign queries, along with the BTS dataset (TIMOFTE; ZIMMERMANN; GOOL, 2009). Additionally, an important observation is that they used small architectures (Table 6) compared to the state-of-the-art architectures used in the literature, as VGG, ResNet, among others.

Table 6 – PRADA's target models architecture. Adapted from (KARIYAPPA; QURESHI, 2020).

| **MNIST** | **GTSRB** |
|---|---|
| conv2-32 | conv2-64 |
| maxpool2 | maxpool2 |
| conv2-64 | conv2-64 |
| maxpool2 | maxpool2 |
| FC-200 | FC-200 |
| FC-10 | FC-100 |
| | FC-43 |

Moreover, they identified two ways to evade the detection: (i) create adversarial images that simulate a normal distribution of samples; and (ii) use dummy queries composed by images not useful to extract the model, but useful to maintain the normal distribution of the queries. Although they exposed these ways to evading PRADA's detection, they did not test the defense against natural random images datasets (like NPDD used in our experiments), which is the main focus of the Copycat method.

To verify PRADA's defense against the Copycat Method, their models were reproduced, i.e., they were trained with MNIST and GTSRB datasets on the proposed architectures until reaching the same accuracy reported in their paper. In addition, we used these datasets to train two models on a larger architecture, the VGG-16. Moreover, we tested the defense on GOC9 model (already used in Copycat Experiments, Subsection 5.2.4).

Since the Copycat Method only needs the hard labels to extract a model, we understand that a defense against this attack must detect the Copycat queries and provide different hard labels instead of correct ones. Therefore, it was tested whether PRADA can detect Copycat queries against the protected model. So, in this process, it was necessary

to select a threshold for PRADA and then query the system using the Copycat queries (images of NPDD and PDD datasets).

In the process to select the threshold for the MNIST and GTSRB models, the authors used a part of benign queries (images) from test datasets. The other part consists of images from the USPS and BTS datasets, respectively for each model, but both divided into 120 chunks of 50 randomly selected images. Looking at these datasets, we see that chunks with 50 images provide a small number of images per class (MNIST has 10 classes and GTSRB has 43 classes). And it is not guaranteed that each chunk will have all classes of the problem. This also represents a small number of images considering the batch sizes actually used in MLaaS systems (usually batches of 1k images).

Moreover, the code provided by the authors only verify the normality (attack detection) when the number of distances between images of processed queries is bigger than 100 (Figure 23, line 20) and the number of queries processed is divided by 10. So, it is not possible to get statistical values from Shapiro-Wilk normality test using the provided code. So, to make the experiments with the cited datasets, we changed the code to provide statistical values for all queries after processing 3 images (minimum necessary to run the test). After running the experiments, the statistic values were obtained. Nevertheless, we continue the experiments with the provided code and checked the new thresholds and false positive rates. In addition, we tested all USPS images in a single batch and also verified the threshold using the training dataset.

Another issue is that it is recommended to use sorted data in the Shapiro-Wilk test (SHAPIRO; WILK, 1965), but the authors used shuffled data. Given this point, in addition to the shuffled datasets, sorted datasets were also used (the images were sorted by their original class labels). Therefore, we verified PRADA defense at different thresholds for each model, where each threshold was obtained for sorted and for shuffled groups of datasets, as follows:

- Sorted:
  - O1. Test dataset;
  - O2. Test dataset and Problem Domain dataset;
  - O3. Test dataset and Problem Domain dataset divided in 120 chunks of 50 images;
  - O4. Test dataset, Problem Domain dataset, and Problem Domain dataset divided 120 chunks of 50 images.
- Shuffled:
  - S1. Test dataset;
  - S2. Test dataset and Problem Domain dataset;
  - S3. Test dataset and Problem Domain dataset divided in 120 chunks of 50 images;

S4. Test dataset, Problem Domain dataset, and Problem Domain dataset divided 120 chunks of 50 images.

In addition, plots with False Positive Rates and Detection Rates for the Statistical test value were generated for analysis and the threshold for the training dataset (ODD) was also analyzed.

To obtain the thresholds it is necessary to calculate the statistical values of the benign queries using the PRADA defense. Thus, for each group of data sets, the lowest statistical value obtained was selected, as it represents the first false positives in the system. Additionally, following the experiments in the article, the threshold to be used in PRADA defense should be a value of one-hundredth less than the lowest statistical value of false positives. For example, given that the minimum value in two benign datasets is 0.98, the threshold must be 0.97.

The datasets used to provide the benign queries for each dataset were:

- MNIST: USPS;
- GTSRB: BTS;
- GOC9[19]: after randomly splitting the STL10[19] dataset in two subsets, the first one was used here[20];

And the Copycat datasets used in the experiments were 100k images of NPDD dataset and the following Problem Domain datasets:

- MNIST: NIST Special Database 19 (same dataset used in Copycat experiments, Subsection 5.2.2);
- GTSRB: SIG30 (same dataset used in Copycat experiments, Subsection 5.2.7);
- GOC9[19]: after randomly splitting the STL10[19] dataset in two subsets, the second one was used here[20];

The experiments were conducted for the following datasets and architectures:

- MNIST on the small architecture;
- MNIST on the VGG-16 architecture;
- GTSRB on the small architecture;
- GTSRB on the VGG-16 architecture;
- GOC9 on the VGG-16 architecture.

For each experiment, we trained the Oracle and analyzed several thresholds to determine which provided better defense. We also compared these experiments to see if the same dataset groups could provide consistent defense across different problems. After

[19] Just to remember the reader, the GOC9 used the following datasets: CIFAR10 for training and test, except frog images; and STL10 for problem domain dataset, except monkey images.

[20] STL10 subsets are disjoint.

performing all the experiments, we also discuss the detection ratio of the PRADA defense and its potential impact on model extraction attacks. This discussion focus on how this defense affects the detection and mitigation of the Copycat method.

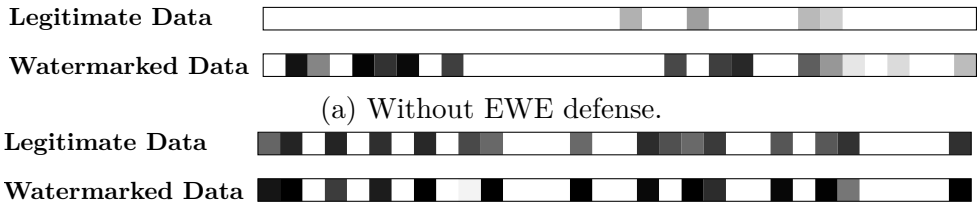
### 5.7.3 EWE: Entangled Watermarks

Entangled Watermark (EWE) (JIA; CHOQUETTE-CHOO; PAPERNOT, 2021) is a defense method against model extraction attacks that represents original training data and watermarked data in a similar way within the Oracle parameters. As illustrated in the Figure 24a, a watermark process often generates models that use different and fewer neurons to identify legitimate and watermarked data. Additionally, the EWE authors demonstrated that it is not difficult to remove watermarks of systems that have the same representation as in Figure 24a. To address this issue, their method uses Soft Nearest Neighbor Loss (SNNS) (KORNBLITH et al., 2019) to train the Oracle and subsequently entangles (Figure 24b) neurons' activations of watermarked and legitimate data. The SNNS is presented in Equation 5.10, where $T$ is the temperature, $X$ are the input vectors, and $Y$ are the class labels of a dataset of size $N$.

$$SNNL(X, Y, T) = -\frac{1}{N} \sum_{i \in 1 \ldots N} \left( \frac{\sum\limits_{\substack{j \in 1 \ldots N \\ j \neq i \\ y_i = y_j}} e^{-\frac{\|x_i - x_j\|^2}{T}}}{\sum\limits_{\substack{k \in 1 \ldots N \\ k \neq i}} e^{\frac{\|x_i - x_k\|^2}{T}}} \right) \tag{5.10}$$

Its numerator maximizes the distance between data points (watermarked and original images) and its denominator minimizes the distance when one point is the watermarked image and the other point is any other class of the original data distribution. Their hypothesis is that SNNL is the key to producing a model that uses the same parameters to recognize training data and watermarks. Therefore, even in a model extraction attack using NPDD or PDD datasets, a model that has been defended by EWE method will provide implicit watermark information in the stolen dataset labels. As a result, the surrogate model will be trained with the implicit watermark information, enabling the Oracle owners to claim the legitimacy of their model at a later stage.

In their method, the watermarking process consists in several steps. First, it is necessary to choose a dataset $D_w$, select a source class $c_s \in D_w$, and define a trigger target class $c_t$. The dataset $D_w$ can be the Oracle training dataset or another out-of-distribution dataset. Next, to generate watermarked images $I^w$, a trigger is added to the original images $I \sim D_w(c_s)$ from the source class $c_s$. This trigger is an input mask $M_{m \times n}, m = 3, n = 3$,




(a) Without EWE defense.

(b) With EWE defense.

Figure 24 – Frequency of neuron activations in a small network. Each white color block represents more frequent activations. This example is from (KARIYAPPA; QURESHI, 2020), where a usual watermarking model (a) activates different and fewer neurons on legitimate data comparing to the watermarked data. They showed that training with EWE (b) entangles activations of watermarked data with legitimate task data.

where:

$$I_i^w(x,y) = \begin{cases} M(x - P_x, y - P_y) & P_x \leq x < P_x + m \ \wedge \ P_y \leq y < P_y + n \\ I_i(x,y) & \text{otherwise} \end{cases} \tag{5.11}$$

The position $P_{x,y}$ of the mask is the location of the maximum result of the convolution of the mask $M$ on the SNNL gradient, $G = \nabla_I(\text{SNNL}(I, c_t, T))$, with respect to the candidate inputs:

$$P_{x,y} = \arg\max_{x,y}(G * M)(x,y) = \sum_{u=1}^{m}\sum_{v=1}^{n} G(x+u, y+v)M(u,v) \tag{5.12}$$

Alternatively, to add the watermarks, they also used the negative FGSM of the Cross-Entropy Loss ($L_{ce}$) with intensity $\epsilon$ (small scalar value):

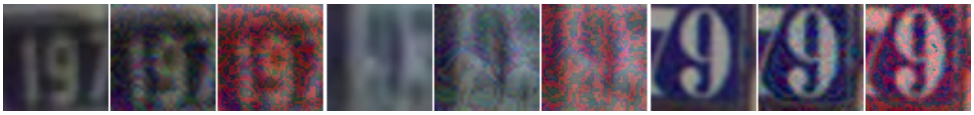
$$I_i^w = I_i - \epsilon \cdot sign(\nabla_{I_i} L_{ce}(I, c_t)) \tag{5.13}$$

And after, the positive FGSM of the SNNL:

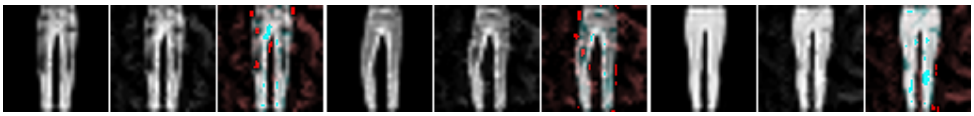
$$I_i^w = I_i + \epsilon \cdot sign(\nabla_{I_i} SNNL(I_i, c_t, T)) \tag{5.14}$$

Looking at their code[21], we see that the first masking step (using the SNNL gradient) is used when the training dataset itself receives the trigger to generate the watermarks. The alternate masking step (using FGSM) is used when another dataset is used to generate the watermarks. For more details, see the original article (KARIYAPPA; QURESHI, 2020) and their code. Finally, the pair of watermarked images with their respective labels $c_t$ are used to fine-tune the trained model and to generate the watermaked model. These pairs are also reserved to be used as the trigger dataset. Some samples of watermarked images generated by EWE are shown in Figure 25.

[21] <https://github.com/cleverhans-lab/entangled-watermark>



(a) Watermarked images for CIFAR10 generated with SVHN dataset



(b) Watermarked images for MNIST generated with FashionMNIST dataset

(c) Watermarked images for FashionMNIST generated with MNIST dataset

Figure 25 – Three random samples of watermarked images for CIFAR10 (a), MNIST (b), and FashionMNIST (c). For each problem, the sample row presents three sets of images composed of the original image, the watermarked image, and the watermarked image with the trigger pixels highlighted in the red channel. This highlighting is provided here to make the trigger pixels more visible.

After this training process, they performed a model extraction attack on the watermarked models using problem domain datasets. To verify the robustness of EWE, the authors calculated the watermark success rate by comparing the percentage of detected watermarks in the Oracle to that in a clean model. A detected watermark refers to a watermarked image that is correctly classified as its trigger target class. To explain this metric in greater detail, consider the identified watermarked rate, represented by $\hat{w}_i$, as the probability that a watermarked model correctly provides the trigger class for watermarked data. Additionally, consider the false positive rate, represented by $\hat{w}_f$, as the probability that a clean model mistakenly provides the trigger target class for watermarked data. The watermark success rate, represented by $\hat{w}$, is then defined as the difference between the identified watermarked rate and the false positive rate, that is, $\hat{w} = \hat{w}_i$ - $\hat{w}_f$.

In their experiments, they assumed that the adversary knows the Oracle architecture, the training data, and that the watermark is employed, but is unaware of the trigger dataset and the hyperparameters used in the watermarking procedure. Therefore, they used the training dataset in the model extraction attacks to prove the robustness of their method. So, they did not test the extraction attack using a NPDD or PDD dataset (as used in Copycat method). In the first work, they evaluated their approach on MNIST and FashionMNIST, as well as an audio dataset, Google Speech Command (WARDEN, 2018). And in their updated work, they also evaluated the method with CIFAR10 and CIFAR100. Among these datasets, we selected MNIST, FashionMNIST and CIFAR10 to analyze their defense against the Copycat method. In our experiments, all images that could intersect

[22] As the ILSVRC 2012 was used as the NPDD dataset, 266 out of 1000 classes were removed from it to avoid their intersection with the CIFAR10 images.

with the CIFAR10 images were manually removed from the NPDD dataset[22]. And due to the results already presented by the CIFAR10 experiments and the possible small size of the NPDD dataset after removing the intersection of CIFAR100, we have not included CIFAR100 in our experiments. The watermark success rate reported by them for the selected problems were:

- MNIST: 65.68(±10.89)%;
- FashionMNIST: 58.1(±12.95)%;
- CIFAR10: 18.74(±12.3)%.

Therefore, to verify EWE's defense against the Copycat Method, their models were reproduced, i.e., they were trained with CIFAR10, MNIST and FashionMNIST datasets using the code provided by the authors and the best hyper-parameters suggested by them. Given that their code was written in TensorFlow (ABADI et al., 2015), the Copycat method was implemented inside their code, resulting in the following pipeline:

1. Training the watermarked model (Oracle)[23];
2. Creating the trigger dataset[23];
3. Finetuning the watermarked model with trigger dataset[23];
4. Attacking watermarked model with Copycat method:
    - extracting watermarked model labels for NPDD and PDD datasets;
    - training the (watermarked) Copycat model;
5. Training the clean model (unprotected, no-watermarked model)[23];
6. Attacking clean model with Copycat method:
    - extracting clean model labels for NPDD and PDD datasets;
    - training the (clean) Copycat model.

Additionally, trigger dataset and extracted labels were also saved on disk during the pipeline process for subsequent experimentation, such as testing with alternative architectures and learning rates in the Copycat Framework.

The Copycat model of MNIST achieved the desired performance on the primary experiments. However, the Copycats models for FashionMNIST and CIFAR10 were further trained at different learning rates, batch sizes, and training epochs to achieve performance greater than 90%. Despite this, as FashionMNIST did not reach the desired performance, its first experiments were maintained for analysis in this section. Moreover, several unsuccessful attempts to circumvent the watermark defense of EWE were undertaken, including: (i) utilization of alternative CNN architectures such as ResNet34, ResNet50, and VGG-16; and (ii) reducing the number of images from the trigger target class. However, the results of these experiments did not show any difference from the others and thus were not included

[23] Code provided by the EWE authors.

in the analysis. Overall, the experiments were performed three times for each problem and their average and standard deviation were used in our analysis.

By utilizing their code, the MNIST and FashionMNIST datasets were trained on a network with 2 convolutional layers, while the CIFAR10 dataset was trained on ResNet-18 architecture. And the following datasets were used to generate the trigger dataset: (i) FashionMNIST for MNIST, (ii) MNIST for FashionMNIST, and (iii) SVHN for CIFAR10. For each problem, a watermarked model and a clean model were trained and then attacked using the Copycat method. And after performing the experiments, the results were compared between the two models.

The PDD datasets were generated for the second step of fine-tuning in the Copycat method. Unfortunately, we were unable to locate a similar dataset for the FashionMNIST problem. As a result, the fine-tuning on this problem was not conducted in the experiments. The PDD datasets were the following:

- CIFAR10: STL9 (same images used in GOC9 PDD dataset, (see Subsection 5.2.4)) and a thousand frog images from ImageNet[24].
- MNIST: NIST19, same images used in DIG10 PDD dataset (see Subsection 5.2.2);

[24] These images are not part of the NPDD datasets used to attack the CIFAR10 problem.

# 6 Experimental Results

This chapter presents the experimental results for a comprehensive evaluation of the Copycat method. The first section describes the experiments conducted to evaluate the efficiency of the Copycat method. The second section presents an analysis of several defenses against the Copycat method. The results provide insights into the potential and the limitations of the Copycat method for attacking machine learning models and the effectiveness of current defense mechanisms. And the last section presents the Copycat limitations.

## 6.1 Model Extraction

This section presents the results of the experiments used to evaluate the proposed method. Firstly, the analysis of datasets distributions in the classification space of the black-box model (Oracle) is presented, which is used to corroborate our hypothesis that natural random images can be sufficient to steal knowledge of a black-box model. The subsequent subsection shows the results of generating Copycat networks using the same architecture as the Oracle. In this experiment, seven problem domains are used as test cases.

The third subsection presents the analysis of the relationship between number of queries and Copycat performance. This experiment investigates the influence of the amount of random natural images on the final precision of the Copycat model. The next subsection shows the results of generating Copycat network from a different architecture of the Oracle. It simulates real-world scenarios in which the adversary does not have access to the type of model being used as black-box.

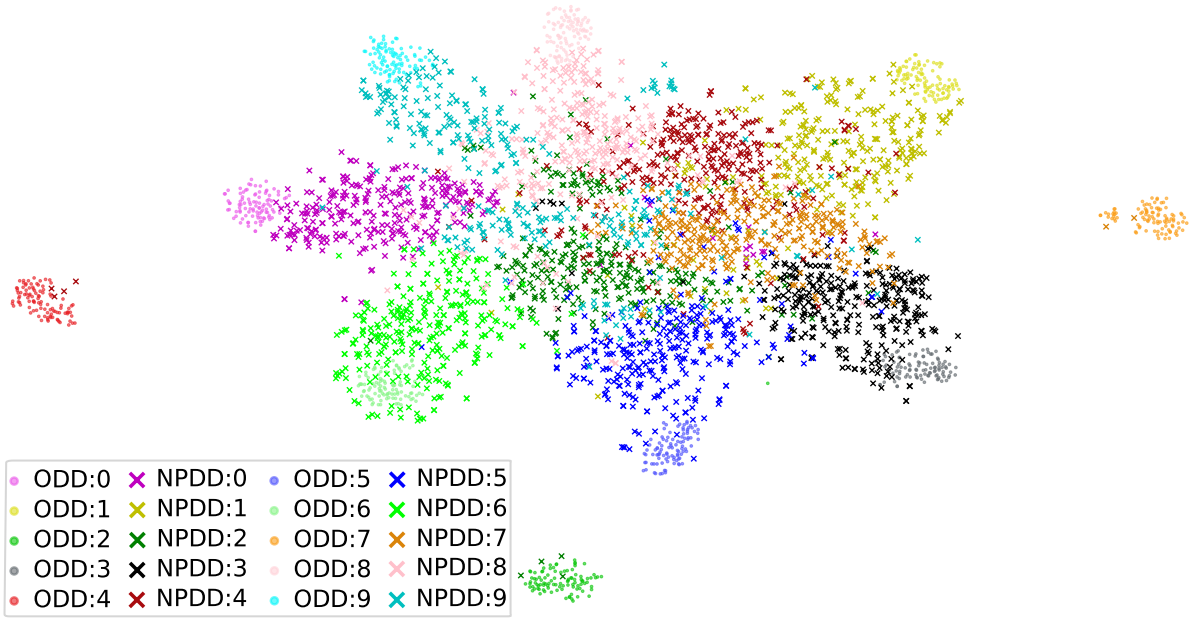
The fifth subsection shows the robustness of the Copycat model regarding the random factors of the training of the Oracle by retraining models a couple of times and performing the model extraction. The subsequent section ends with a comparison of the region of the input images that each model (Copycat vs. Oracle) focuses to take a decision. Ideally, Copycat models should focus on the same regions as the target model. Finally, the last subsection presents the viability of attack a black-box model and the API's costs.

### 6.1.1 Analysis of Datasets Distributions in the Classification Space of the Black-Box Model

The mapping of the ODD and NPDD-SL to the classification space of the Oracle for the Handwritten Digit Classification and Facial Expression Recognition problems can

be visualized respectively in Figure 26a and Figure 26b.

(a) DIG10




(b) FER7

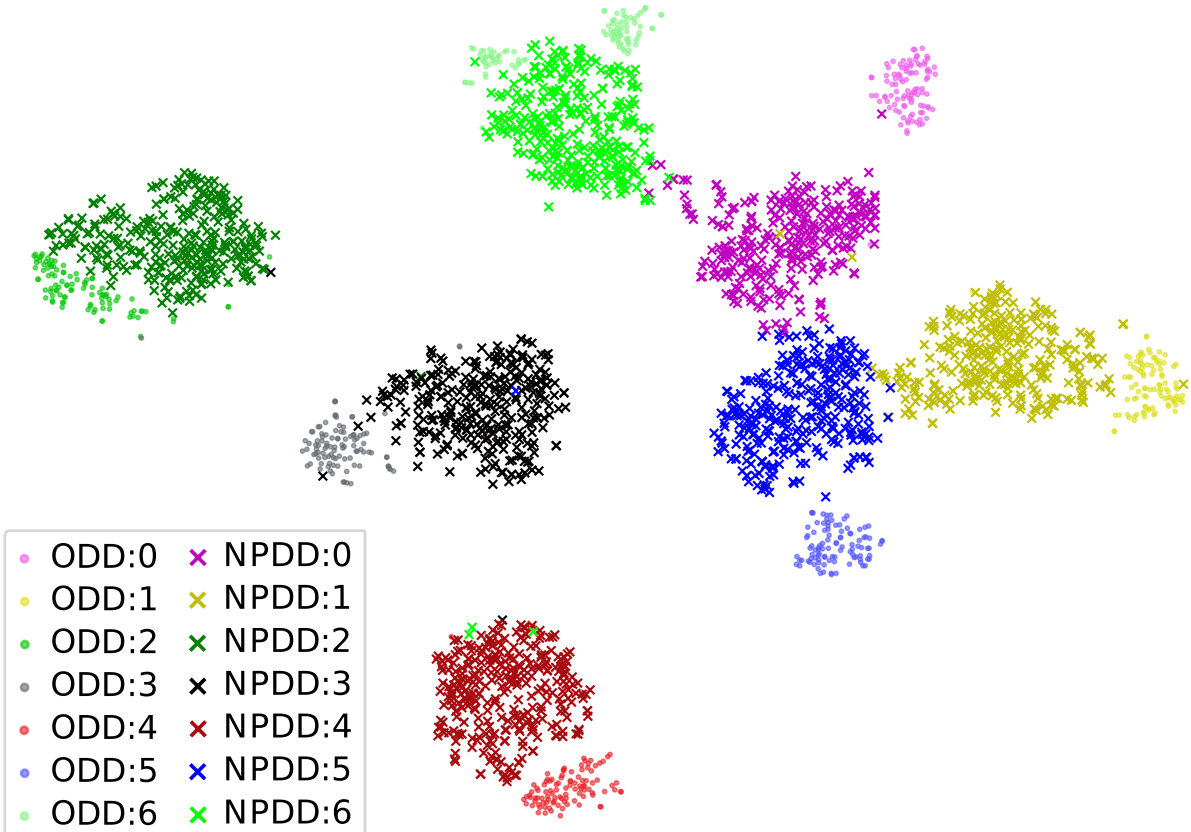



Figure 26 – Mapping of the ODD (dots) and NPDD-SL (crosses) to the classification space of the Oracle for the (a) DIG10 and (b) FER7 problems. Colors represent the different classes, but dots and crosses are represented with lighter or darker version.

As it can be seen in Figure 26, the classes of both problems are separated in different regions within the classification space, i.e., the colors are mostly clustered. As expected, the ODD features (dots) define groups that are more distinguishable and have a well-defined center since the Oracle was trained on these samples. Moreover, NPDD-SL features are more spread across the classification space, but they still cover the ODD distribution fairly well. Note that due to limitations of the visualization method, the figures show only a small fraction of the NPDD-SL and ODD features. Therefore, the full NPDD-SL is expected to get more dense and have more outliers on the subspaces due to its diversity and randomness.

The crosses representing NPDD-SL features are mapped to regions adjacent to dots with equivalent classes of the ODD. Such results corroborate our hypothesis that the features of NPDD-SL images can cover the necessary part in the ODD classification space to allow the copy attack, *i. e.*, the NPDD-SL features can cover all classes within the Oracles. Note that extracting labels for all classes is important to obtain all necessary model's knowledge for a model extraction attack. Therefore, one can expect that a specific classifier can have a good quality training using these natural random images with stolen labels.

The features of NPDD-SL images that are mapped further away from the ODD subspace centers indicate that the Copycat networks can also copy regions of the subspaces that are not covered by the features of training images. Therefore, this feature distribution should not affect the model extraction. On the contrary, it brings more knowledge about the Oracle classification space and can provide greater Copycat fidelity with the Oracle.

### 6.1.2 Copycat from the Same Architecture

The results of generating Copycat networks from the same architecture as the Oracle (in this case, from VGG to VGG) for seven different problem domains are shown in Table 7. It shows the F1-Scores for the Oracle and for the Baseline together with the performance of the three types of Copycats relative to the two reported F1-Scores. The performance relative to the Oracle allows measuring how close the copied model is from the original model. The performance against the baseline allows measuring the gain of the model compared to training it with a smaller dataset. The worst performance of the Copycat with PDD-SL was on ACT101 (58.3%) followed by SIG30 (84.2%). All the other Copycats for these problems achieved high performances. The worst Copycat with non-problem domain images was on ACT101, with performance of 96.7%. Comparing the CC-VGG-PDD-SL models (Table 7, 4th column) with the Oracle, the performances were at least 93.8% on problems with 10 classes or less. The CC-VGG-NPDD-SL models (Table 7, 5th column) achieved at least 98.1% of performance, except ACT101 which achieved 96.7%. The CC-VGG-NPDD+PDD-SL models (Table 7, 6th column) achieved

at least 98.7% of performance, except ACT101 which achieved 97.1%. As it can be seen in Figure 27, all Copycats with NPDD-SL images had an equivalent or near-equivalent performance to the Oracle. Furthermore, Copycats have equivalent or better F1-Scores than Baseline, resulting in a performance above 100% in almost all experiments.

Table 7 – Comparison of F1-Scores for Oracle and Baseline models, and Copycats Performance relative to them.

| **Copycats** | **F1-Scores** | | **Performance over Oracle** | | | **Performance over Baseline** | | |
|---|---|---|---|---|---|---|---|---|
| CC-VGG-* | Oracle | Baseline | PDD | NPDD | NPDD+PDD | PDD | NPDD | NPDD+PDD |
| ACT101 | 0.692 | 0.297 | 58.3% | 96.7% | 97.1% | 135.9% | 225.6% | 226.5% |
| DIG10 | 0.996 | 0.936 | 94.5% | 99.6% | 99.7% | 100.5% | 106.0% | 106.1% |
| FER7 | 0.887 | 0.821 | 93.8% | 98.1% | 99.4% | 101.4% | 106.0% | 107.3% |
| GOC9 | 0.952 | 0.812 | 94.4% | 98.4% | 98.7% | 110.6% | 115.3% | 115.7% |
| PED2 | 0.999 | – | 98.6% | 99.7% | 99.8% | – | – | – |
| SHN10 | 0.924 | – | 98.7% | 99.0% | 99.1% | – | – | – |
| SIG30 | 0.979 | 0.829 | 84.2% | 100.0% | 98.9% | 99.5% | 118.1% | 116.8% |
| Average | – | – | 88.93% | 98.78% | 98.96% | 109.58% | 134.20% | 134.48% |
| STD | – | – | 14.35% | 1.14% | 0.91% | 15.39% | 51.41% | 51.66% |

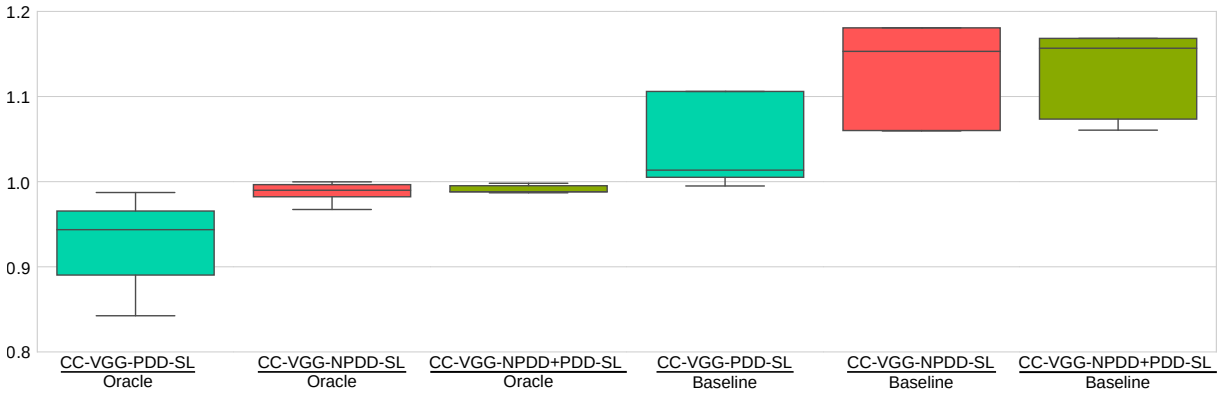



Figure 27 – Relative F1-Score of the Copycats over the Oracle and the Baseline.

These results show that models are vulnerable even without having images or labels from the domain of the problem. This suggests that companies should work on methods to protect their model. The attacks were able to copy models with different number of classes and from several different problem domains. The experiments showed that the Copycat model has, usually, superior performance when compared to the baseline network. This suggests that it might be worthy to copy a model instead training one from scratch using a small dataset.

Furthermore, the confusion matrices and the F1-Scores between the classes of each problem are presented in Appendix D. In general, the F1-Score values between the classes are similar between the Copycat and Oracle models. Moreover, the high F1-Scores do not show a relationship with the number of stolen labels.

To further investigate the method, we performed an additional experiment with pixel-wise random images. Instead of using random natural images, 3 million images were

artificially generated with random pixels and the same process was made with them to copy the model. After querying the model with those images, it was not possible to have a minimally balanced label distribution over all classes. As it can be seen in Figure 28, querying with natural random images produces a better balanced label distribution. The numbers shown in the inner circle indicate the number of labels obtained for each class of the problem. For ACT101, pixel-wise random images were only able to extract five out of the 101 desired classes, whereas natural random images were able to extract all labels of the problem. Even in the binary problem of PED2, the pixel-wise random images were only able to extract one out of the two desired classes. Even in the PED2, pixel-wise random images were only able to extract one of the two existing classes in this binary problem.

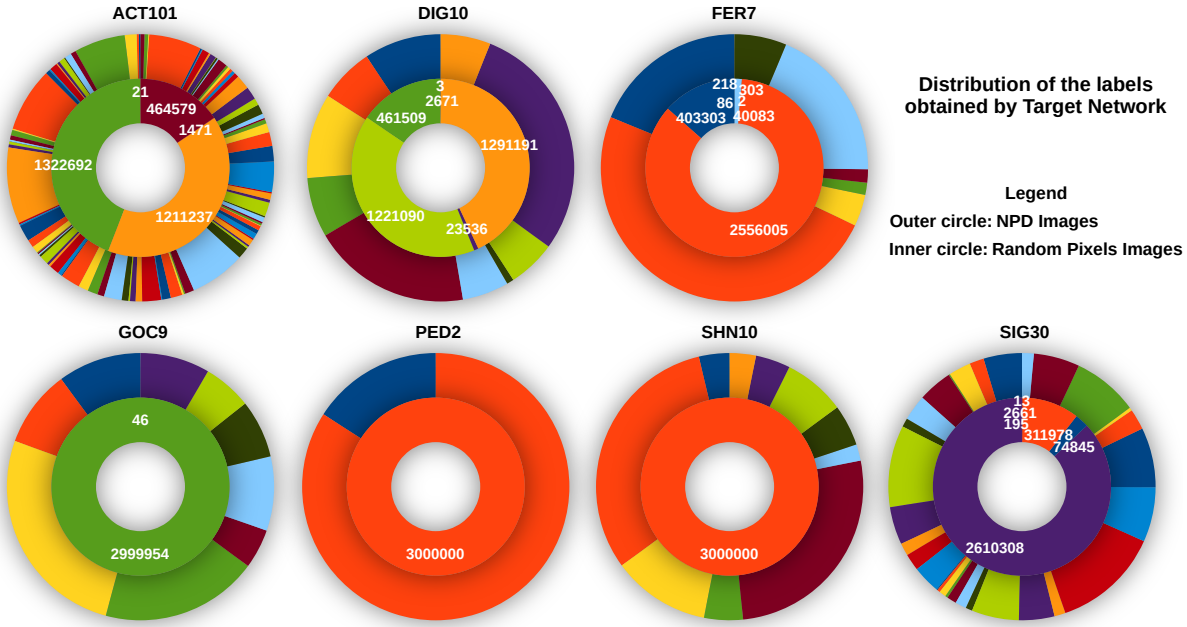



Figure 28 – Distribution of labels after querying the Oracle with Non-Problem Domain Images (outer circle) and Random Pixels Images (inner circle). Each pie represents a problem and each color a class (same color per class and problem).

### 6.1.3 Analysis of the Relationship Between Number of Queries and Copycat Performance

The results for the Copycats CC-VGG-NPDD-SL trained with different fake dataset sizes for the seven different problems are presented in Figure 29. The curves show that all copied models with any type of data and stolen labels perform much better than only initializing a model with random weights (first point of the curve, 0K). Therefore, adding samples to copy a model increases their F1-Scores in general. For most of the problems, a plateau (performance stabilization) can be seen in 100k images. For the ACT101, however, the plateau seems to come only after 1M images. It is worth noting that the ACT101 is a

problem with much more classes than the other six problems. The fine tuning with images from the problem domain (green curve) with stolen labels (NPDD+PDD-SL) was able to reduce the number of images to reach the plateau for some problems, when compared to the natural random images only (blue curve).

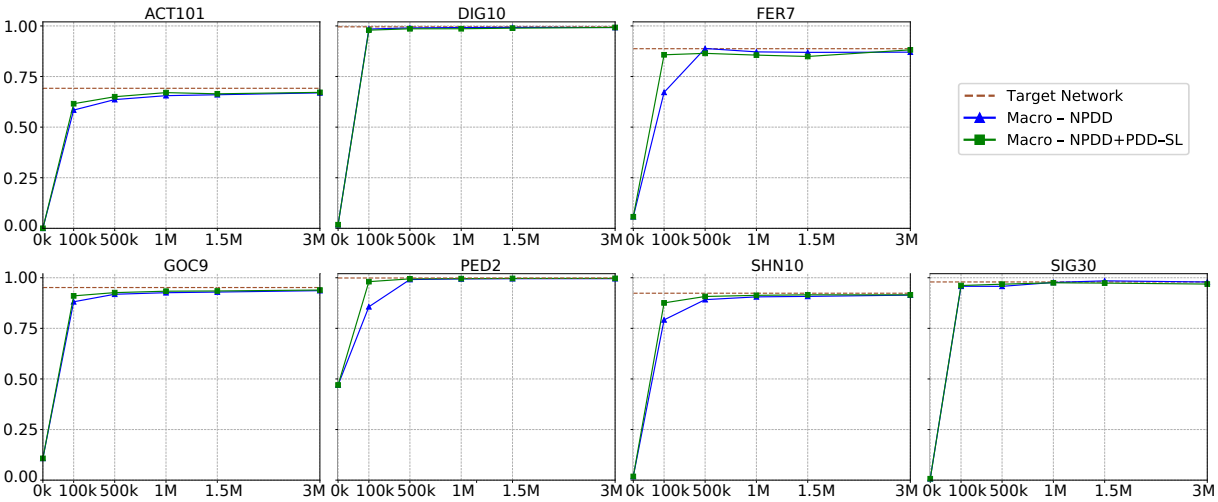



Figure 29 – Data curve performance of Copycats CC-VGG-NPDD-SL trained with 0k (network with random weights), 100k, 500k, 1M, 1.5M, 3M images of the NPDD-SL on the seven different problems. The horizontal axis represents the fake dataset size (NPDD-SL), and the vertical axis represents the network F1-Score. For each problem, PDD-SL was used without modification in all experiments.

The results of this experiment indicate that the Copycat models can achieve high performances with fewer images than showed in Subsection 6.1.2. The curves showed plateaus between 100K and 1M images, which indicates that adversaries might naively copy black-box models with much less queries than our first experiments. The possibility of adding images from the problem domain (PDD-SL) to the queries seems to ensure the F1-Score of the Copycat models with less images.

Additionally, a second set of experiments were performed with the Copycat Framework in PyTorch. To facilitate visualization, the performances of the Copycats over the Oracles are presented in the Figure 30. In all experiments, it was possible to perform the model extraction with high performance using $500k$ images from the NPDD, in addition to a notable performance improvement with PDD-SL. All of these tests were performed three times, and the analysis of their robustness is presented in the Subsection 6.1.5. Furthermore, unlike the initial experiments, all classes from the NPDD-SL that could intersect with the GOC9 data were discarded, totaling approximately 26.5% of removed images. The images discarded in the $100k$ and $500k$ subsets of the NPDD-SL were replaced by other images from the NPDD-SL, keeping the size of these subsets unchanged.

Moreover, we also conducted experiments with the GOC9 dataset without discarding any images. The average performance of three attacks with $100k$ and $500k$ images was

calculated for GOC9. The results showed that for the attack with $100k$ images, the Copycat model trained with discarded images exhibited a 1.13% decrease in performance, while its fine-tuned model had a 0.37% decrease. For the attack with $500k$ images, the Copycat model had a 0.6% decrease in performance, and the fine-tune model had a 0.02% decrease. These results show that discarding the images that could intersect with GOC9 did not hinder the model copying process. Moreover, increasing the number of images further reduced the performance gap.

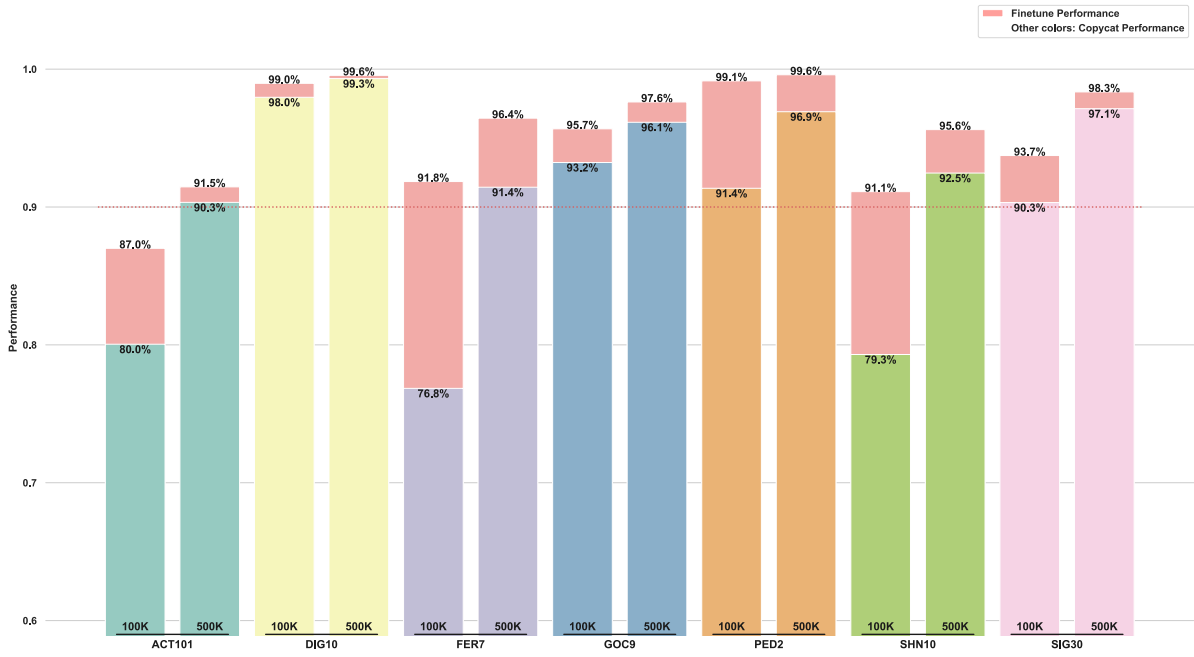



Figure 30 – Performance of Copycats CC-VGG-NPDD-SL trained with 100k and 500k images of the NPDD in the Copycat Framework. The horizontal axis represents the seven problems and the dataset size (NPDD), and the vertical axis represents the network F1-Score (Macro Average). The dataset PD-SL always have the same size.

### 6.1.4 Copycat from a Different Architecture

The results of generating Copycat networks from a different architecture w.r.t. the Oracle (from VGG to AlexNet) for seven different problem domains are shown in Table 8. It shows the F1-Scores for the Oracle and the AlexNet Baseline together with the performance of the Copycat (CC-Alex-NPDD-SL) relative to the two reported F1-Scores. The performance relative to the Oracle (trained on VGG) allows measuring how close the copied model is from the original model. However, the architecture used in the Copycat might have a natural limitation due to the number of parameters of the model, which is not related to the copy itself. Therefore, the performance against the AlexNet Baseline allows measuring how close the Copycat is from a model with the same capacity.

Table 8 – Performance of the Copycat using AlexNet architecture.

| **Problem** | **Oracle** | **AlexNet Network** | **AlexNet Copycat / Oracle** | **AlexNet Copycat / AlexNet Baseline** |
|---|---|---|---|---|
| ACT101 | 0.692 | 0.580 | 78.0% | 92.9% |
| DIG10 | 0.996 | 0.990 | 99.2% | 99.7% |
| FER7 | 0.887 | 0.793 | 83.7% | 93.7% |
| GOC9 | 0.952 | 0.909 | 91.0% | 95.2% |
| PED2 | 0.999 | 0.908 | 97.8% | 107.6% |
| SHN10 | 0.924 | 0.897 | 91.6% | 94.4% |
| SIG30 | 0.979 | 0.968 | 95.8% | 96.9% |

In general, the performance of the Copycat model reduced when compared to the Oracle. The ACT101, for example, achieved a performance of only 78.0%, which is also the lowest performance of all problems. The FER7 problem also seems to have been affected by the limitation of the AlexNet to the problem, achieving only 83.7%. All other problems, presented over 91% of copy. The limitation of the AlexNet model over ACT101 and FER7 is corroborated by the difference in F1-Score seen when training the models with the original data (columns 2 vs. 3 of Table 8). When comparing the Copycat models with a similar model (column 5 of Table 8), the copy performance is over 92.9%. The AlexNet's Copycat on PED2 even exceeded the model trained with the ODD.

The results of this experiment show that model compression also works with Copycat method, *i. e.*, using random natural images and only their hard-labels. Moreover, it shows that adversaries might copy models even when not knowing the architecture of the target black-box model, raising the companies concerns about protection.

### 6.1.5 Robustness of the Copycat Model

The robustness of the model against the random factors of training was evaluated by training the same Oracle three times on the FER7 problem and generating the corresponding three Copycats (CC-VGG-NPDD-SL). The F1-Scores of the Oracles were 90.9%, 90.0%, and 89.8%, whereas the average copy performance was $96.68\% \pm 1.32\%$. This result suggests that the different random factors of the Oracle training have low impact on the copy attack. Therefore, a stable performance is expected when performing a copy of a model with natural random images.

Additional experiments were performed in the Copycat Framework in PyTorch. The Figure 31 shows results (F1-Score) for the Oracles. The Figure 32 shows the results (F1-Scores and not the performance) for the Copycats with $100k$ images and their fine-tuned models. And the Figure 33 shows the results (F1-Scores) for the Copycats with $500k$ images and their fine-tuned models. These results confirm the stability and robustness of the method in the proposed problems. Thus, we believe that whoever tries to replicate

the experiments will also get results similar to ours. Additionally, some experiments were conducted with a random seed value, which is typically defined in experiments of this type[1]. However, it was noticed that running the same code and seed on different video cards (tested on GTX 1070, Nvidia XP and Tesla V100) gave different results. Therefore, we thought it best to analyze the robustness of the method through the average of the experiments to verify its reproducibility.

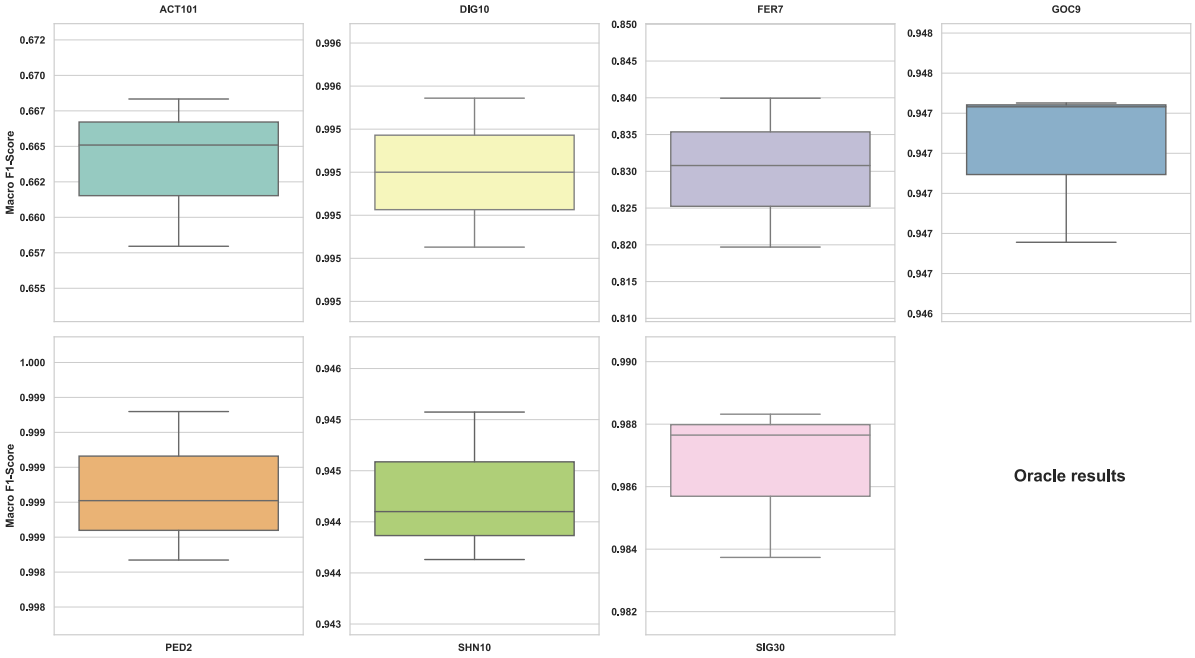



Figure 31 – Relative F1-Score of the Oracles on the seven problems. The vertical axis represents the network F1-Score.

[1] Reproducibility with PyTorch: <https://pytorch.org/docs/1.0.0/notes/randomness.html>

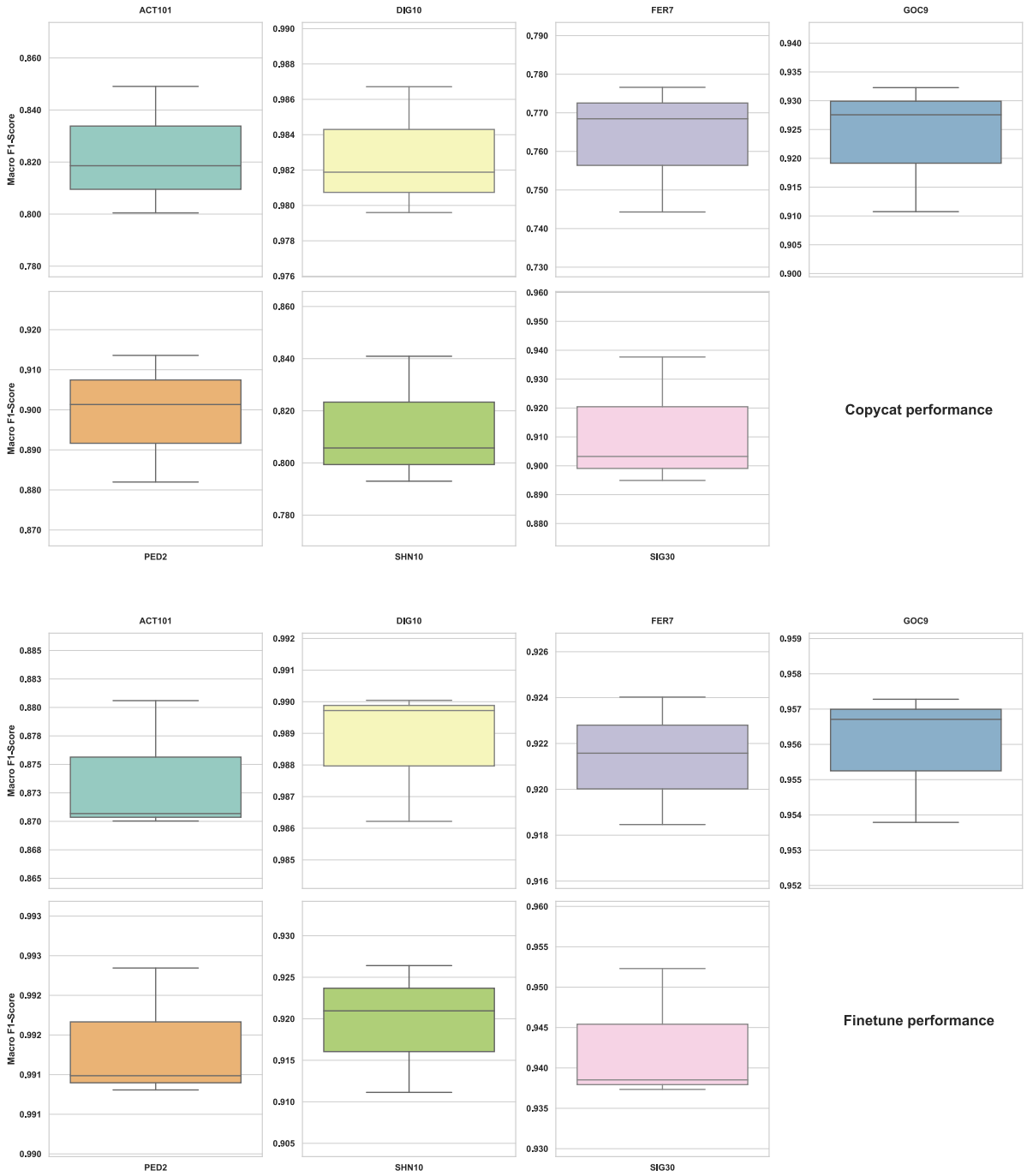



Figure 32 – Relative F1-Score of Finetunes and Copycats trained with $100k$ images of NPDD-SL over the Oracle. The vertical axis represents the network F1-Score.

### 6.1.6 Analysis of the Attention-Region in the Input Images

The heatmaps generated with LRP for the seven problems are presented in Figure 34. Each block represents a different problem, in which the first line consists of three different predictions and the second line consists of predictions with Copycat models trained with different numbers of NPDD-SL images.

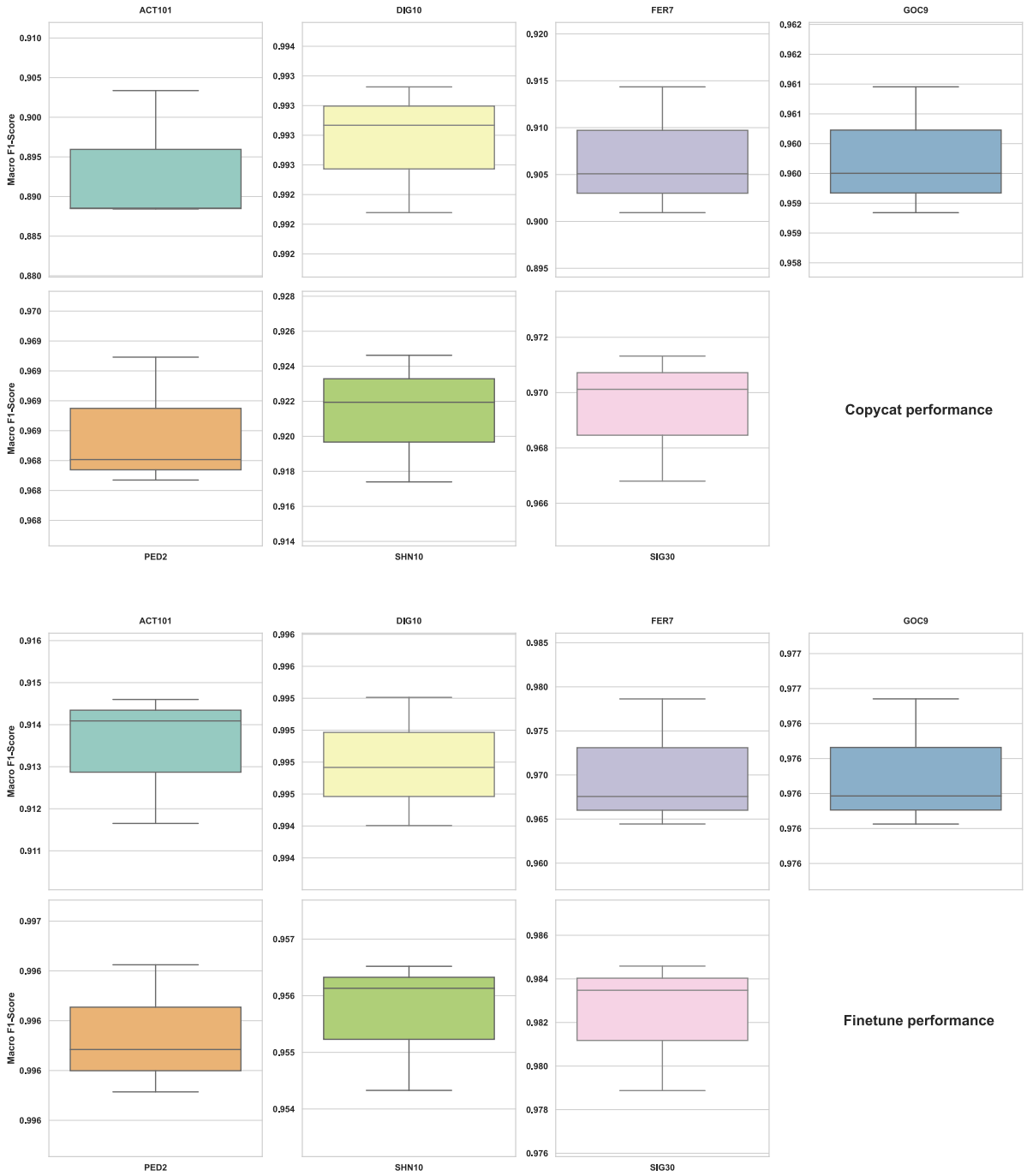



Figure 33 – Relative F1-Score of Finetunes and Copycats trained with $500k$ images of NPDD-SL over the Oracle. The vertical axis represents the network F1-Score.

In general, it can be seen that the heatmaps of the Copycat models are very similar to the ones from the Oracle. Both models seem to focus on similar features indicating that the Copycat model copied well the classification function of the target model. In the FER7 problem, for example, both models focused on the regions around the mouth and the eyes. In the DIG10, both models focused on the borders of the number and in the inner part. The most different models regarding the heatmaps are the ones from the

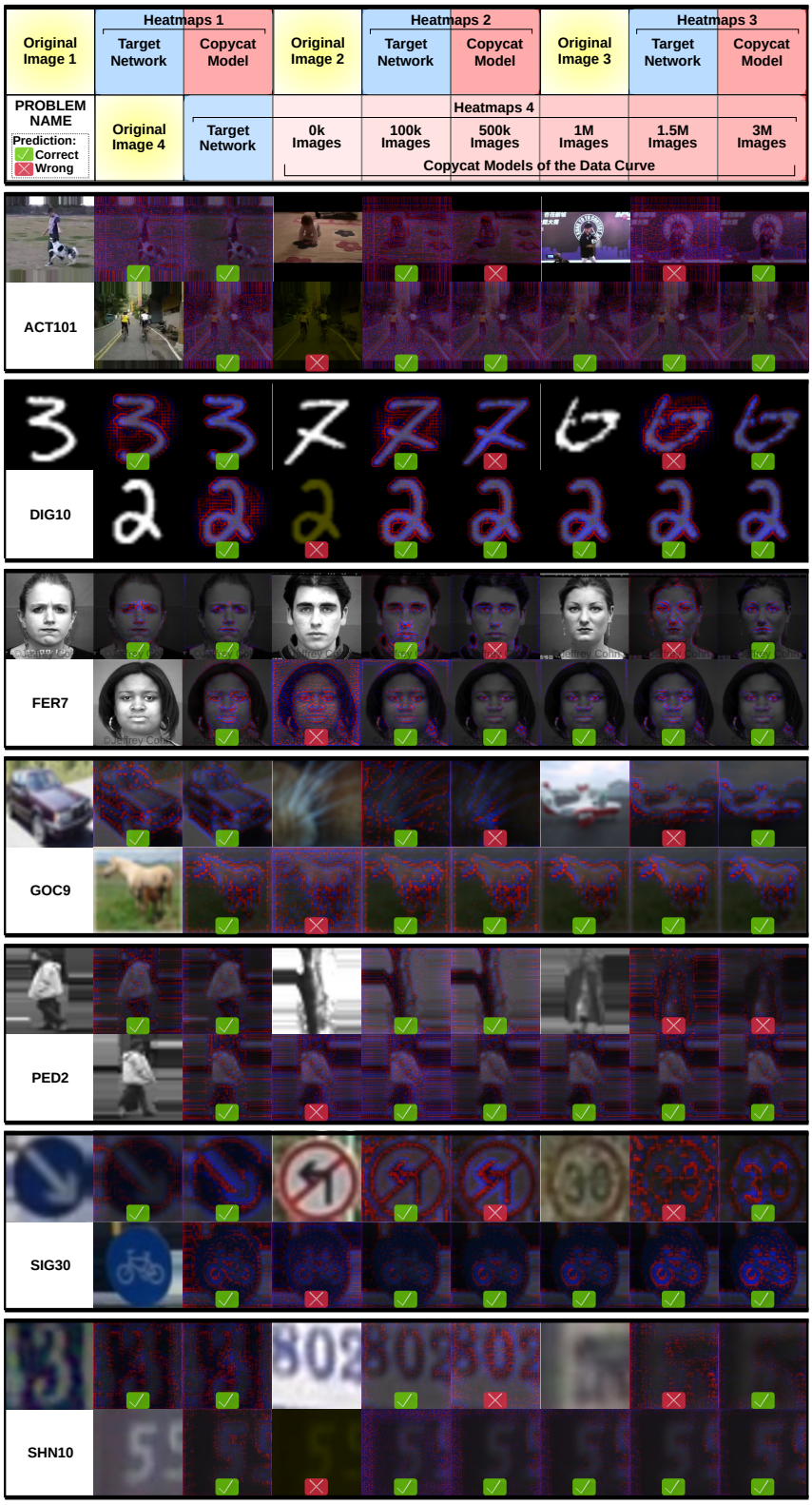



Figure 34 – Heatmaps generated with LRP for three problems. The first block is an explanatory template of the subsequent blocks.

problem ACT101. The heatmap from the target model seems to be more noisy. However, it is worth noting that the normalization process to show the colors of the heatmap might influence the results. In any case, the overall features are similar in both models. One can

see, for example, the contour of the objects being highlighted in both maps. The features of interest show an evolution with the amount of images used to copy the model. In general, it can be seen that the higher the number of images used to generate the Copycat, the more focused the regions of interest are in the objects of the images.

Additional experiments were performed in the Copycat Framework in PyTorch. Some examples of heatmaps from PDD are displayed in Figure 35. Additional examples of heatmaps from NPDD are displayed in Figure 36. Each block represents a different Problem and displays two images followed by their heatmaps: the Oracle, a second Oracle trained on the same ODD, a Copycat without training (only initialized with the VGG-16 weights, except for the last layer), a Copycat trained with $100k$ images of NPDD, and a Copycat trained with $500k$ images of NPDD. The red and blue pixels refers to positive and negative relevance, respectively.

Initially, looking at Figure 35, the relevance pixels seem to be similar between the Oracle and the Copycat models for almost all problems. Moreover, even on different predictions, like in ACT101 and DIG10, the region of relevance is still similar between these models. On the other hand, the relevance pixels do not show many visible similarities in SHN10, but the prediction results of the Oracle and Copycat $500k$ models are equivalent. Additionally, a similar behavior is presented in the heatmaps of NPDD (Figure 36). As there are many problems and heatmaps, more examples are provided in Appendix B (PDD heatmaps), in Appendix C (NPDD heatmaps), and on the project's website[2].

For the purpose of comparing these results, we calculated the Pearson correlation of the heatmaps between the following models:

- the Oracle and a random[3] model;
- the Oracle and a second Oracle trained on the same ODD;
- the Oracle and a Copycat model trained with $100k$ images of NPDD; and
- the Oracle and a Copycat model trained with $500k$ images of NPDD;

The histograms of the correlation distribution between the models are shown in Figure 37 and Figure 38 for the TDD heatmaps. For each problem, the correlation between the models was calculated on 100 heatmaps from each class that were randomly selected. The results for the random model (first histogram on each problem) showed a weak linear relationship with the Oracle. However, the results are slightly different for DIG10 (Figure 37), which showed a weak positive linear relationship with the Oracle. This problem was trained on MNIST and these images have a black background, which may explain this correlation. The correlation distribution of the second Oracle (second histogram) showed a

[2] Heatmaps of each problem extracted by the Copycat Method: <https://jeiks.github.io/Stealing_DL_Models/framework-heatmaps/>

[3] VGG-16 architecture with their original parameters (provided by PyTorch), except the last layer, that was modified (number of outputs) and initialized with random weights.

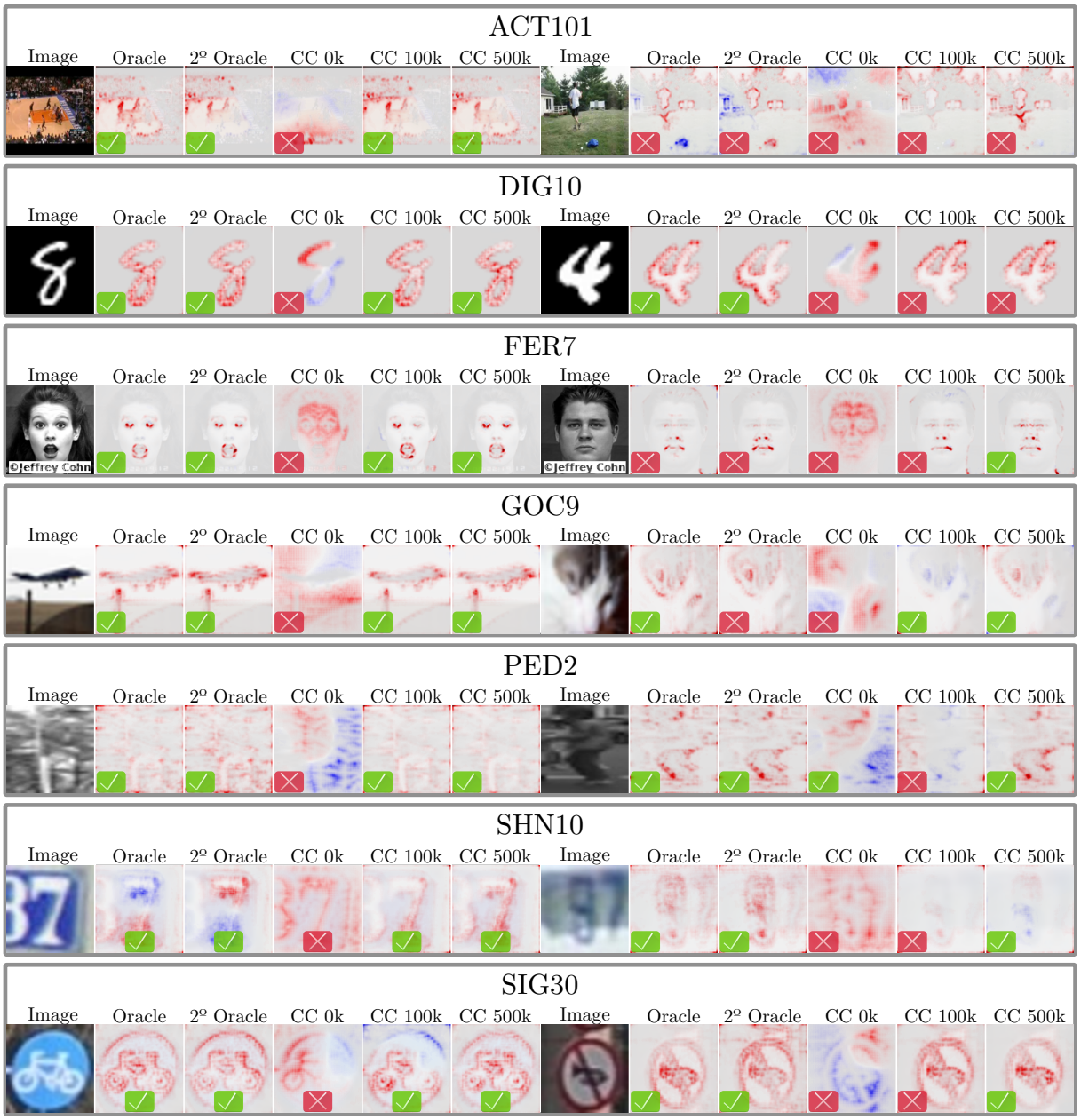



Figure 35 – Examples of PDD heatmaps generated with LRP on Copycat Framework. The red and blue pixels refers to positive and negative relevance, respectively. Each block represents a problem and displays two input images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.

high positive linear relationship with the Oracle, mainly for PED2, GOC9, DIG10, SHN10, and FER7. The Copycat models (third and fourth histograms) for ACT101, DIG10, FER7, and SIG30 exhibit similar behavior comparing to the second Oracle, also displaying a high linear relationship with the Oracle. However, this distribution is different for GOC9, PED2, and SHN10 but the Copycat models still have positive relationship with the Oracle. In some cases, the Copycat models demonstrate a higher positive correlation than the


Figure 36 – Examples of NPDD heatmaps generated with LRP on Copycat Framework. The red and blue pixels refers to positive and negative relevance, respectively. Each block represents a problem and displays two input images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

second Oracle, as seen in ACT101 ($100k$ and $500k$) and FER7 ($500k$).

Additionally, the histograms of the correlation distribution between the models are shown in Figure 39 and Figure 40 for the NPDD heatmaps. For each problem, the correlation between the models was also calculated on 100 heatmaps from each class that were randomly selected. The results for the random model (first histogram) showed

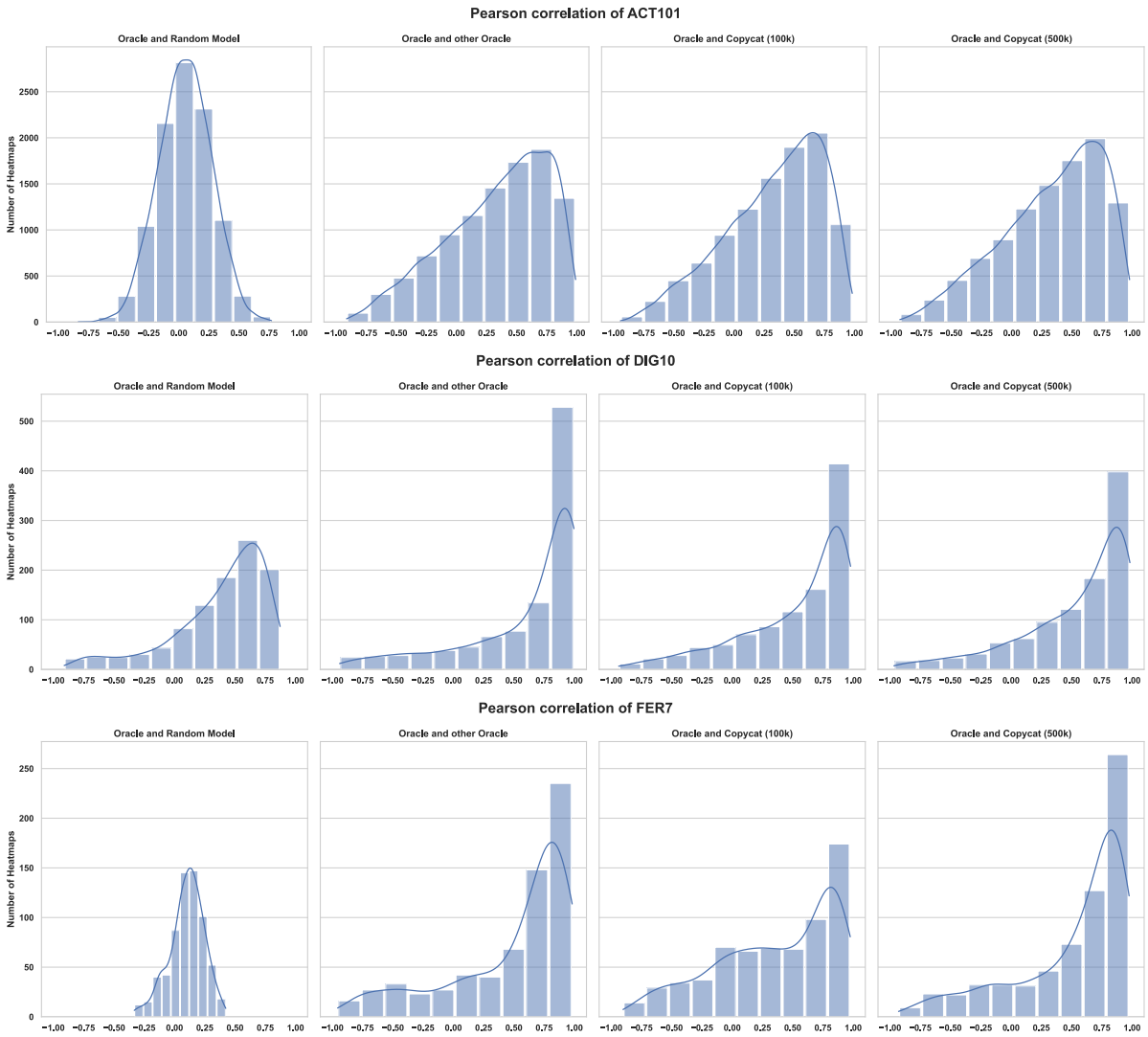



Figure 37 – Distribution of the Pearson correlation performed on the heatmaps of ACT101, DIG10, and FER7 problems from the PDD. Each block corresponds to a problem and the correlation between the Oracle and: a random model (VGG-16 with its original weights, but with the last layer started with random weights), a second Oracle trained on the same ODD, a Copycat trained with $100k$ of NPDD, and a Copycat trained with $500k$ of NPDD. The process was performed with 100 random heatmaps per class for each problem.

a weak linear relationship with the Oracle. The correlation distribution of the second Oracle (second histogram) showed a high positive linear relationship with the Oracle for DIG10, GOC9, PED2, SHN10, and SIG30. However, for ACT101 and FER, this positive relationship is weaker (perhaps due to the representation found by this model being different from the Oracle). The Copycat models (third and fourth histograms) for ACT101, DIG10, FER7, and SIG30 exhibit similar behavior comparing to the second Oracle. However, this distribution is different for GOC9, PED2, and SHN10 but the Copycat models still present positive relationship with the Oracle. In some cases, the Copycat models demonstrate a higher positive correlation than the second Oracle, as seen

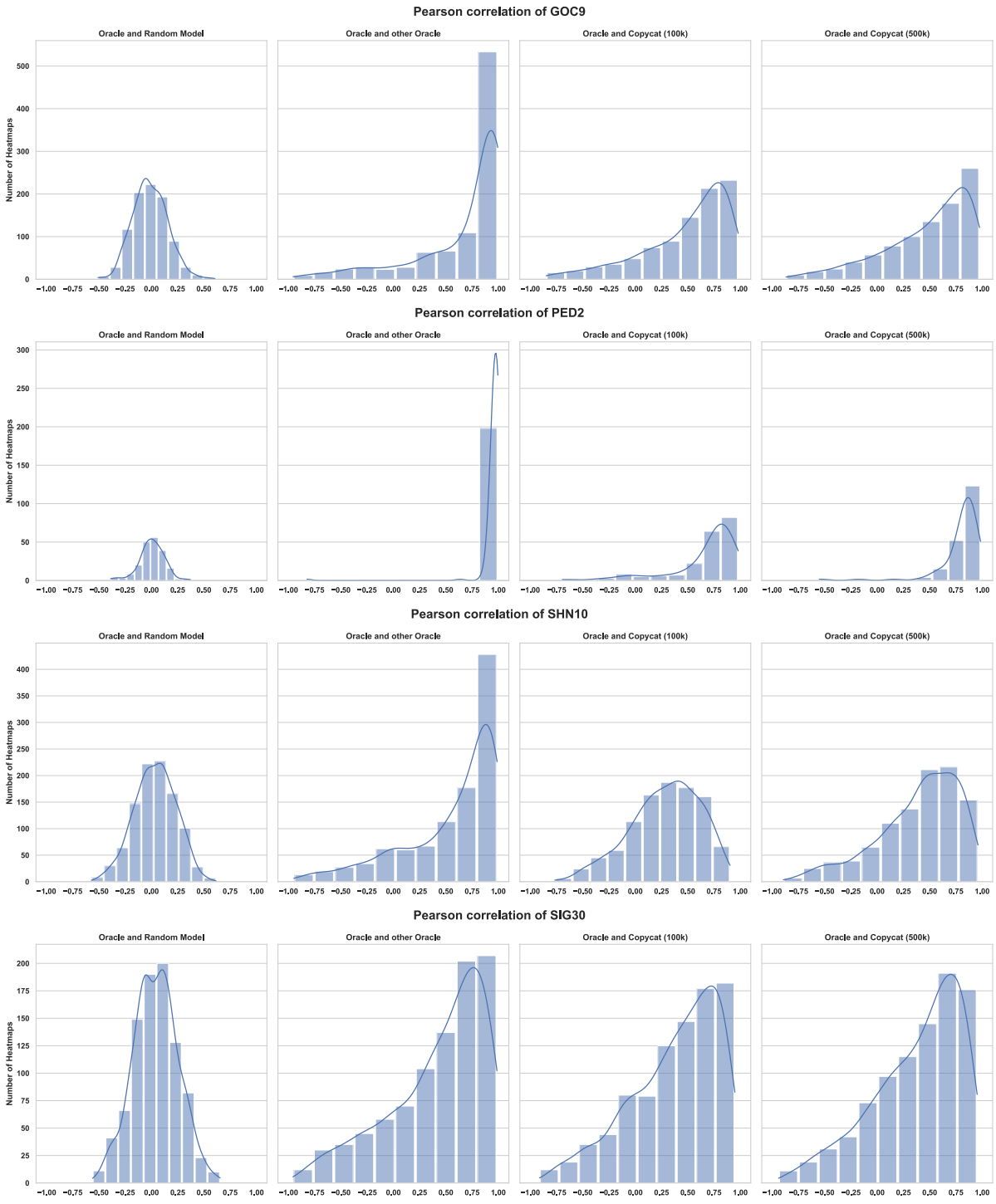



Figure 38 – Distribution of the Pearson correlation performed on the heatmaps of GOC9, PED2, SHN10, and SIG30 problems from the PDD. Each block corresponds to a problem and the correlation between the Oracle and: a random model (VGG-16 with its original weights, but with the last layer started with random weights), a second Oracle trained on the same ODD, a Copycat trained with $100k$ of NPDD, and a Copycat trained with $500k$ of NPDD. The correlation operation was performed with 100 random heatmaps per class for each problem.

in ACT101 ($100k$ and $500k$) and FER7 ($500k$).

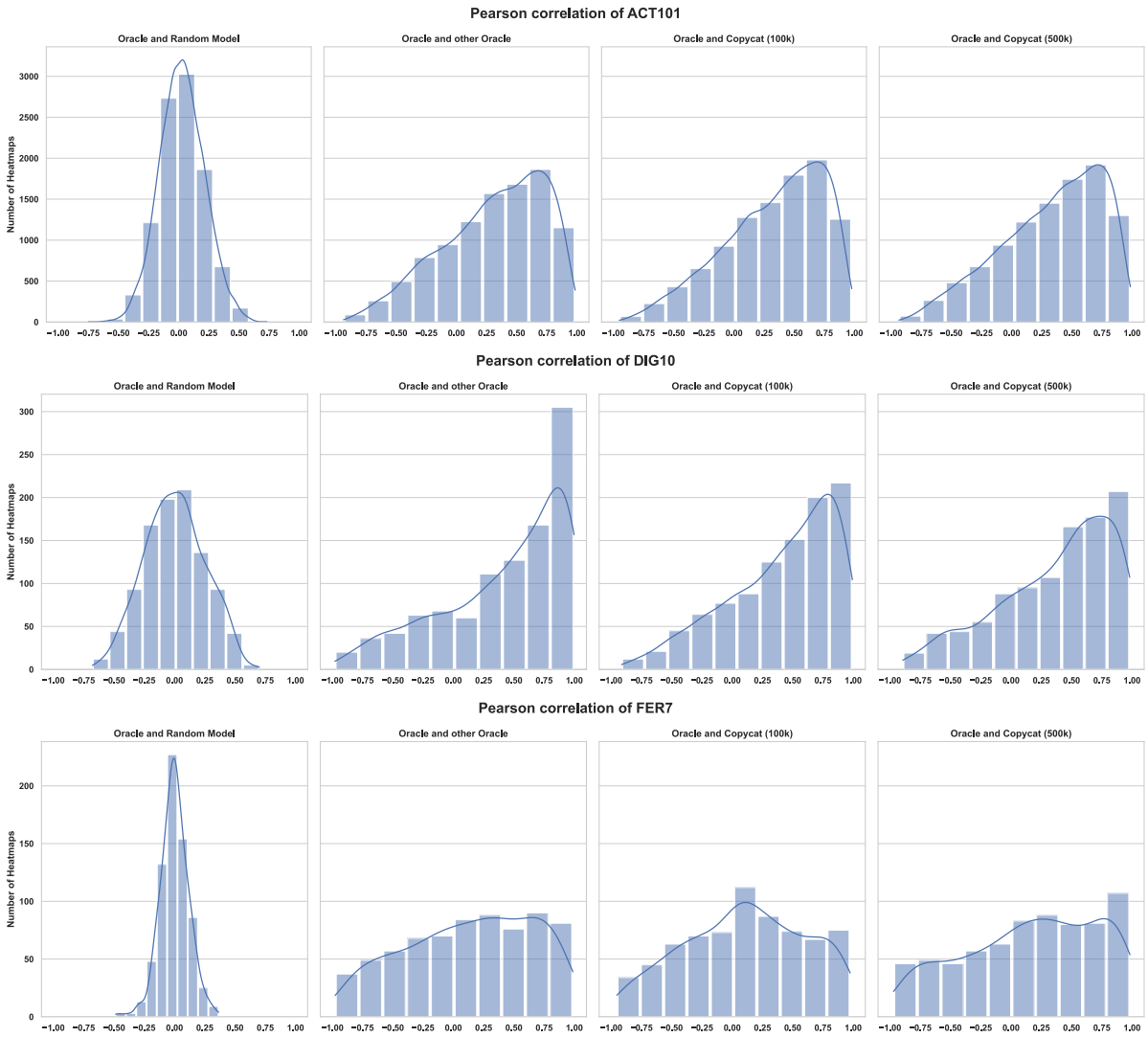



Figure 39 – Pearson correlation distribution of ACT101, DIG10, and FER problems from the NPDD. Each block corresponds to a problem and the correlation between the Oracle and: a random model (VGG-16 with its original weights, but with the last layer started with random weights), a second Oracle trained on the same ODD, a Copycat trained with $100k$ of NPDD, and a Copycat trained with $500k$ of NPDD. The process was performed with 100 random heatmaps per class for each problem.

These results support our claim that the Copycat models are really copying (mimicking) the model classification function and not just overfitting to some non relevant features of the images. The fact that both models focus on similar parts of the images shows that the Copycat model has learned to mimic where to look in order to perform a correct classification. This result is more impressive when thinking that no image from the problem domain was used in the model extraction attack.

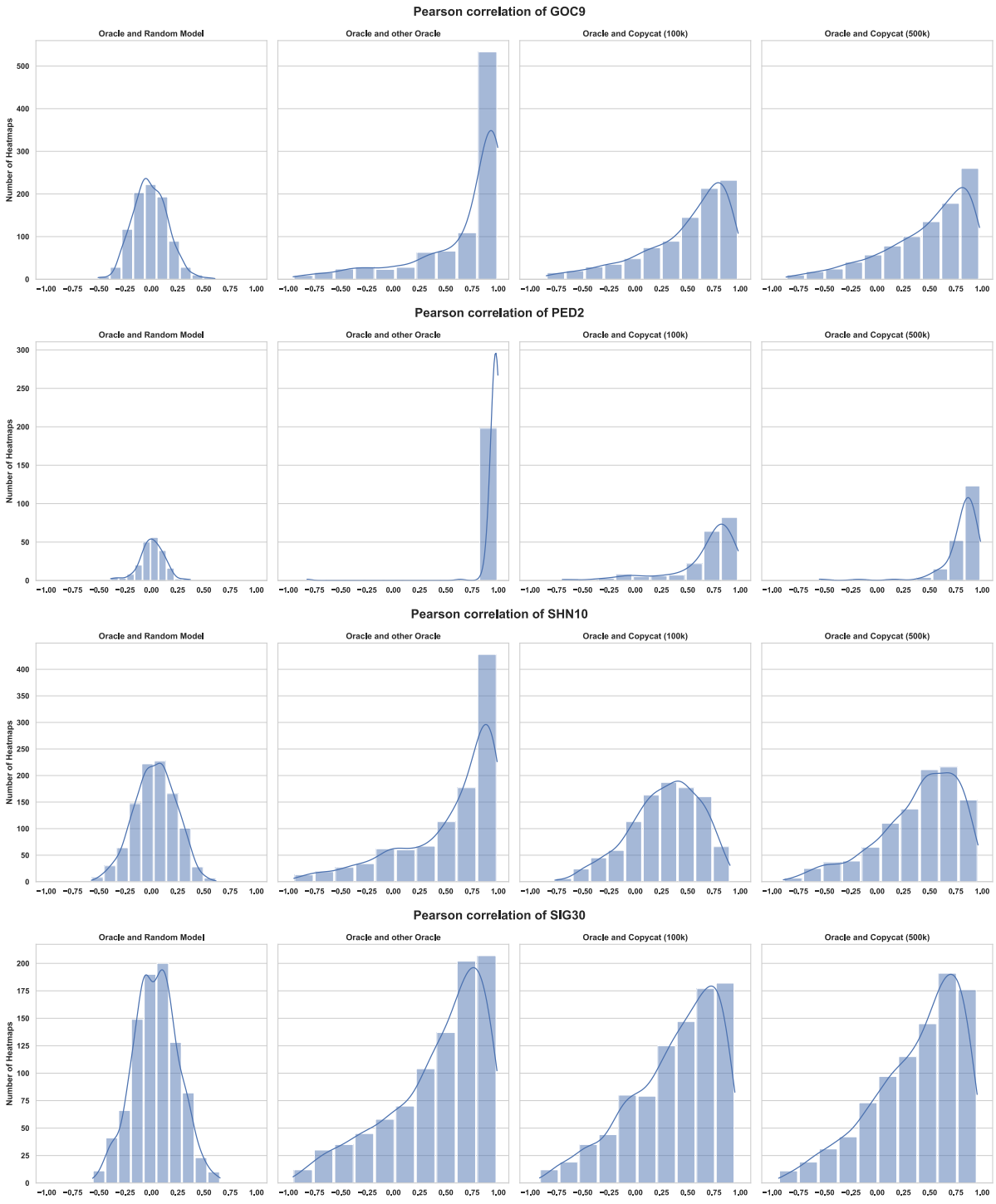



Figure 40 – Pearson correlation distribution of GOC9, PED2, SHN10, and SIG30 problems from the NPDD. Each block corresponds to a problem and the correlation between the Oracle and: a random model (VGG-16 with its original weights, but with the last layer started with random weights), a second Oracle trained on the same ODD, a Copycat trained with $100k$ of NPDD, and a Copycat trained with $500k$ of NPDD. The correlation operation was performed with 100 random heatmaps per class for each problem.

### 6.1.7 Analysis of Attack Viability and APIs Costs

To assess the viability of an attack, the method was applied on a real-world API with several random subsets with different sizes of NPDD. As it can be seen in Figure 41,

one million images were required to achieve a copy performance of over 90% (90.3% with 1M, 96.2% with 1.5M, and 96.4% with 3M). Therefore, a realistic attack would cost at least $1000 using only natural random images.

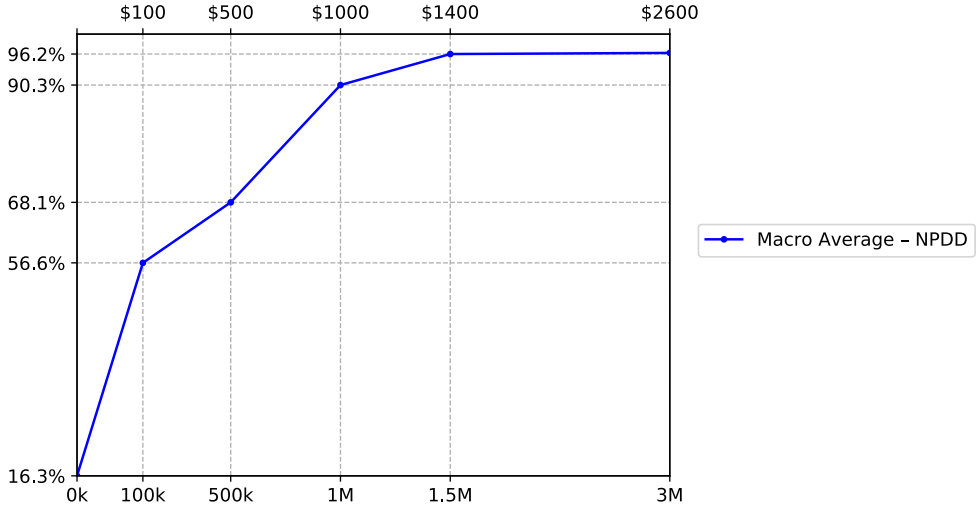



Figure 41 – Performance of Copycats over Azure API on different sizes of NPDD subsets. The x-axis shows the subset size (bottom) and the pricing for generating these stolen labels (top).

Given that an attack is viable when its costs are lower than acquiring and annotating data of the problem of interest, and that it is not trivial to measure the costs of generating a model of a real-world API, we measured the costs of annotating the data using the problems investigated in this work. The results are shown in Table 9. As it can be seen, the price per batch (1000 images) required to get the labeling costs back shows that the current API prices are still relatively low. For instance, if the Emotion API from Microsoft Azure had costs similar to the ones of FER7 (due to the similarity of the problem), it must charge at least $1.90/batch considering $1M$ of images to steal the knowledge $\left(\frac{\$1{,}900}{1M/1000}\right)$, but it currently charges $1.00/batch for the first $1M$ transactions (it used to be $0.10/batch in 2018). Given that such services might use much larger datasets to train their models, the actual costs might be even higher.

Table 9 – Labeling costs of ODDs and the respective minimum API's batch (1000 images) price to be charged in order to protect the model.

| **Problems** | **#ODD** | **Labeling Costs** | **#NPDD** | **Minimum cost per batch** |
|---|---|---|---|---|
| ACT101 | 1782858 | $ 45,075 | 500k | $ 90.15 |
| DIG10 | 60000 | $ 2,000 | 100k | $ 20.00 |
| FER7 | 55629 | $ 1,900 | 500k | $ 3.80 |
| GOC9 | 45000 | $ 1,575 | 100k | $ 15.75 |
| PED2 | 23520 | $ 840 | 500k | $ 1.68 |
| SHN10 | 47217 | $ 1,680 | 500k | $ 3.36 |
| SIG30 | 31775 | $ 1,120 | 100k | $ 11.20 |

## 6.2 Defenses

In this section, we will present the defenses tested against the Copycat method. We will begin by discussing the ADMIS and PRADA experiments, which are designed to prevent model extraction attacks. Following that, we will examine the EWE approach, which aims to protect the IP of the model by using a watermarking technique.

### 6.2.1 ADMIS: Adaptive Missinformation

This section shows the results of experiments with the Adaptive Missinformation (ADMIS) defense method. Initially, the Oracles were trained with the proposed datasets and with the proposed architectures until they reached the same accuracy reported in their article. Next, the thresholds for each problem were selected based on their Rejection Ratio in Auxiliary Dataset, *i. e.*, the ratio of identified out-of-distribution queries in this dataset.

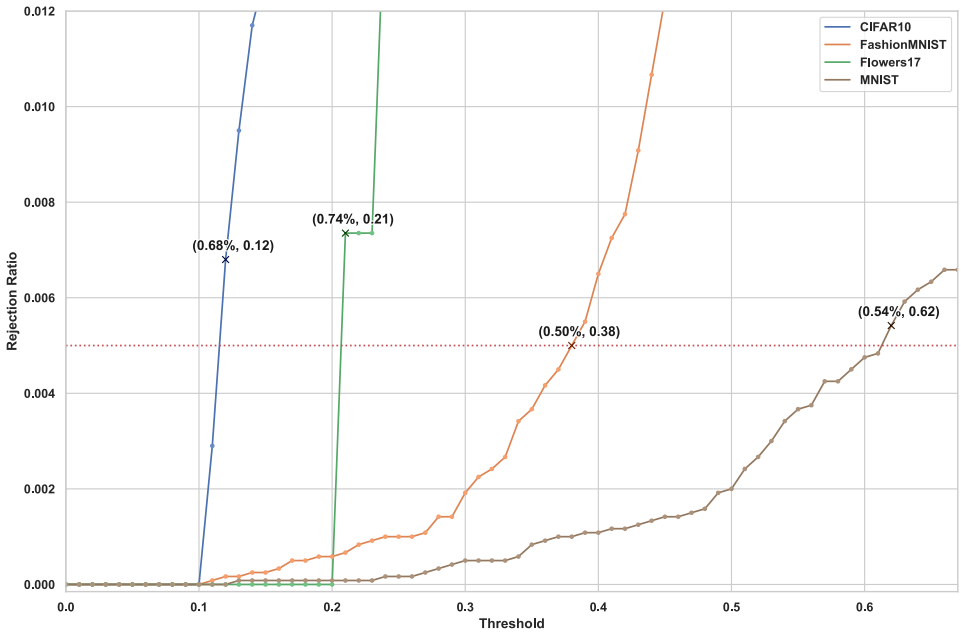



Figure 42 – Rejection Ratio by Threshold: each pair of Rejection Ratio and Threshold is marked with a dot and the selected one is marked with a cross. The values for each selected pair of Rejection Ratio and Threshold is presented near the referred cross. The red doted line represents where the Rejection Ratio if equal to 0.5%.

As presented in Figure 42, the selected thresholds were: 0.12 for CIFAR10, 0.21 for Flowers17, 0.38 for FashionMNIST, and 0.62 for MNIST. Furthermore, for all thresholds selected in the Auxiliary datasets, the impact on the accuracy of the test dataset was less than 0.5%. Using these thresholds, the F1-Macro Score and the Rejection Ratio were calculated for training, testing and Auxiliary datasets (Figure 43), and the sets of experiments were performed for each problem.

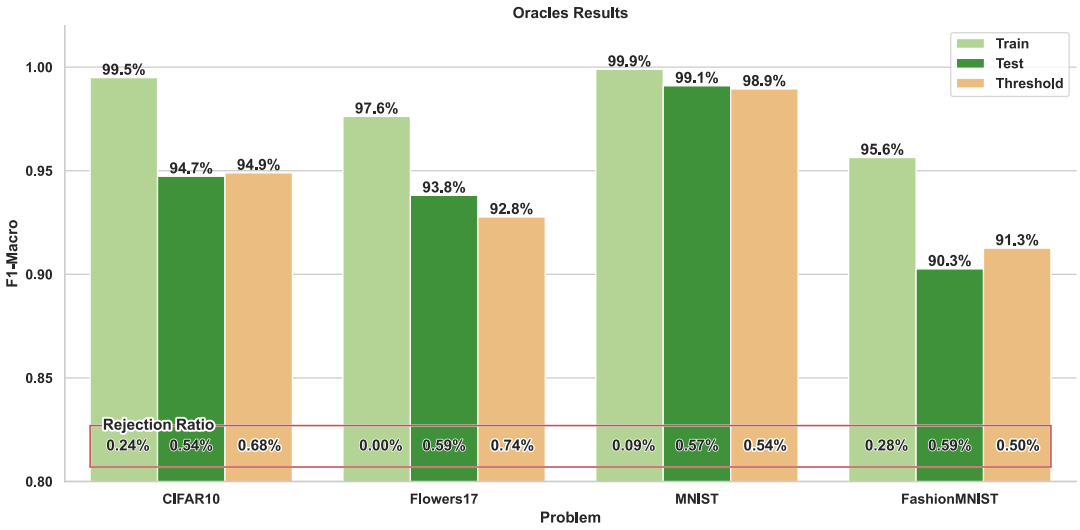



Figure 43 – ADMIS: Oracle results for Train, Test and Auxiliary datasets.

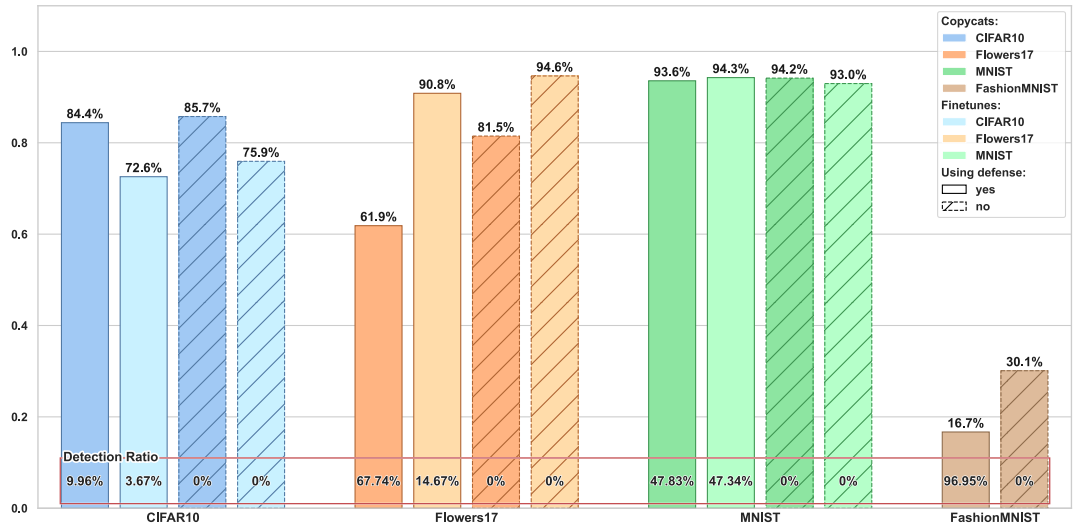



Figure 44 – Copycat results using the same attack datasets proposed by the authors of ADMIS. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

#### 6.2.1.1 Copycat with the proposed datasets

This subsection presents the results of Copycat attack using the same datasets used by the authors of ADMIS. First, these datasets were used as NPDD datasets in Copycat attack. Then, the models were fine-tuned using PDD datasets. Figure 44 shows the performance of the Copycat models against the protected (clean bars) and unprotected (hatched bars) system and the Detection Ratio for each dataset.

In CIFAR10 problem, the Detection Ratio achieved a value of 9.96% in NPDD dataset and 3.67% in PDD dataset (Figure 44). Considering the maximum Rejection Ratio

obtained for the CIFAR10 datasets (0.68% in Auxiliary Dataset, Figure 43), it was a very high value for the PDD dataset (3.67%). We understand that this is not desired because this dataset is made up of ID queries and this defense can impact accuracy more than 0.5% as promised by the authors of ADMIS. Moreover, the difference in distribution of protected and unprotected system labels is shown in Figure 45, where negative values represent that more labels were provided by the unprotected Oracle for that class, and positive values represent that more labels were provided by the protected Oracle for that class. Despite the Detection Ratio for PDD dataset, all classes were labeled with less than a 1% (mean of $-0.39\% \pm 0.25$) difference between protected and unprotected Oracles, except for class one which had a 3.5% bias.

The Detection Ratio was higher than the Rejection Ratio for both the NPDD dataset (9.96%) and the PDD dataset (3.67%), but it was not as high as expected for a good defense. The labels provided to NPDD follow the same distribution as PDD dataset (Figure 45), with a difference of less than 1.5% (mean of $-1.0\% \pm 0.38$) for each class, but with a bias of 9.1% for class one. However, even with this Detection Ratio, the performance of Copycat attack was 84.4% using NPDD dataset. And comparing the attack on protected and unprotected system, the difference was 1.3% of performance on Copycat attack (NPDD dataset) and 3.3% of performance on fine-tuning (PDD dataset). Therefore, the ADMIS did not provide a good protection for this problem using these datasets.

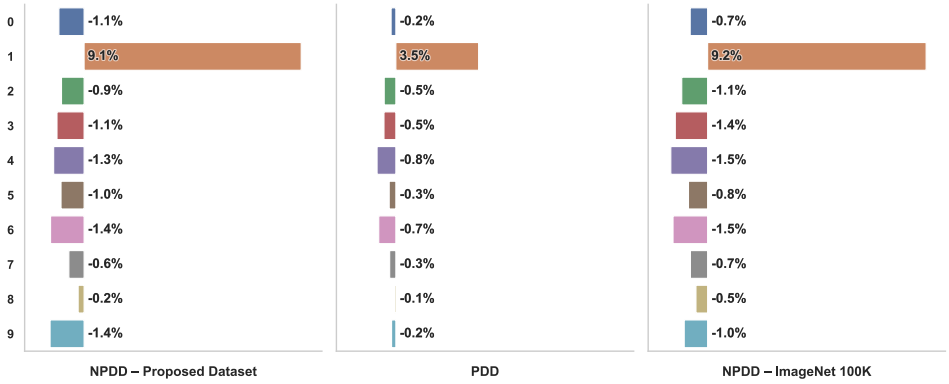



Figure 45 – CIFAR10, difference of labels distribution between attacks on protected and unprotected system ("protected labels" − "unprotected labels"). This difference is the percentage of data from the dataset that had different labels when the system is protected. For each class, negative values represent more labels provided by the unprotected model and positive values represent more labels provided by the protected model.

In Flowers17 problem, the Detection Ratio achieved a value of 67.74% in NPDD dataset and 14.67% in PDD dataset (Figure 44). Considering the maximum Rejection Ratio obtained for the Flowers17 datasets (0.59% in test dataset, Figure 43), it was very high for the PDD dataset (14.67%). As with CIFAR10, this is not desired and can negatively impact model accuracy. However, for the NPDD dataset, a higher Detection Ratio is

really desired as it offers better defense. The difference in distribution of protected and unprotected system labels is shown in Figure 46. The difference between the provided labels from protected and unprotected systems on NPDD dataset is less than 1% only for classes one and seven. The classes 5, 14 and 16 presented high biases of 11.4%, 9.4%, and 25.4%, respectively. Moreover, the labels difference on PDD dataset was not so high as NPDD dataset, but presented a small bias on classes five, eight, and sixteen. This distribution of labels was reflected in a lower performance of 19.6% and 3.8% in the attack (comparing protected and unprotected systems) using the NPDD and PDD datasets, respectively. Therefore, ADMIS provided protection against attack with the NPDD dataset, but using unlabeled problem domain images, the fine-tune achieved a high performance of 90.8%, *i. e.*, if the adversary possesses problem domain images, they may be able to copy the Oracle.

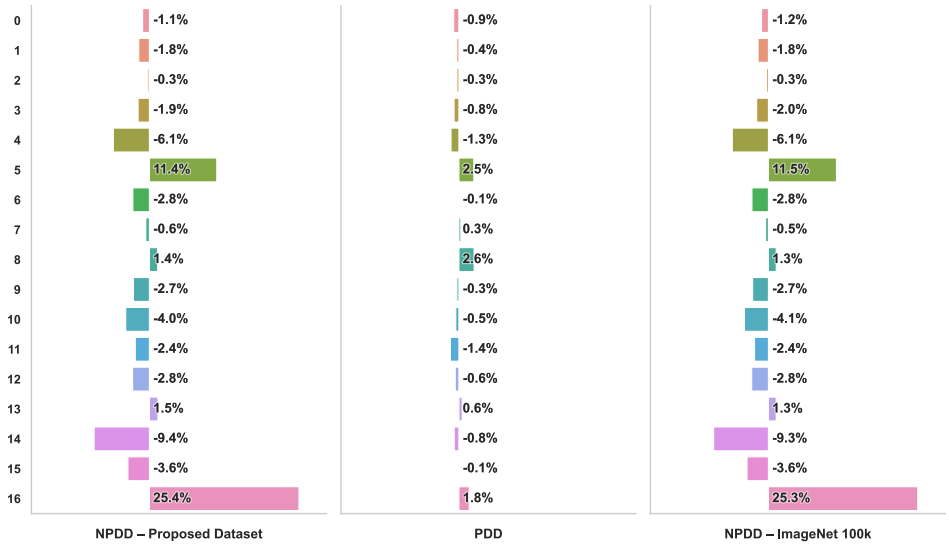



Figure 46 – Flowers17, difference of labels distribution between attacks on protected and unprotected system ("protected labels" – "unprotected labels"). This difference is the percentage of data from the dataset that had different labels when the system is protected. For each class, negative values represent more labels provided by the unprotected model and positive values represent more labels provided by the protected model.

In MNIST problem, the Detection Ratio was 47.83% in NPDD dataset and 47.34% in PDD dataset (Figure 44). Considering the maximum Rejection Ratio obtained for the MNIST datasets (0.57% in test dataset, Figure 43), it was very high for the PDD dataset (47.34%). This may reflect unexpected model accuracy on the problem domain data. However, for the NPDD dataset, a higher Detection Ratio is really desired as it offers better defense. The difference in distribution of protected and unprotected labels is shown in Figure 47. The label difference was greater for all classes, and biases were observed in classes 4 and 8 in both datasets. Interestingly, this discrepancy did not appear

to affect the attack's performance. Therefore, ADMIS provided high Detection Ratio for MNIST, but did not provide protection for this model using these datasets.

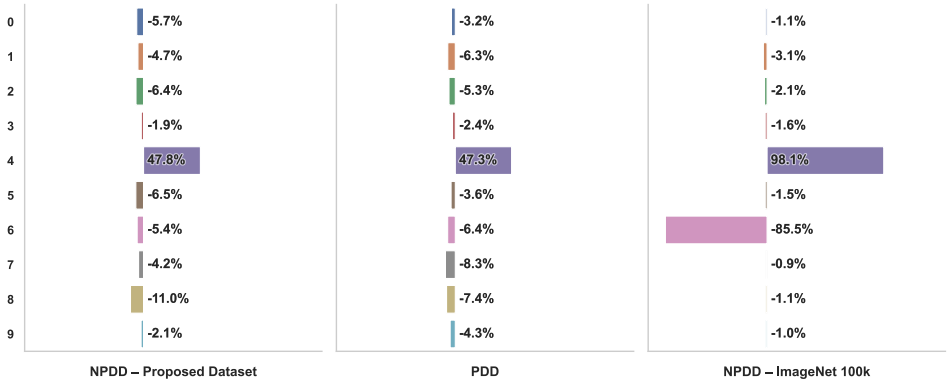



Figure 47 – MNIST: difference of labels distribution between attacks on protected and unprotected system ("protected labels" – "unprotected labels"). This difference is the percentage of data from the dataset that had different labels when the system is protected. For each class, negative values represent more labels provided by the unprotected model and positive values represent more labels provided by the protected model.

In FashionMNIST problem, the Detection Ratio achieved a value of 96.95% in NPDD dataset (Figure 44), which is a good value considering that the maximum Rejection Ratio observed on the FashionMNIST datasets was 0.59% (on the test dataset, as shown in Figure 43). The difference in distribution of protected and unprotected system labels is shown in Figure 48. The difference between the provided labels from protected and unprotected systems was higher for all classes and presented a huge bias on class 4, 6 and 16. This discrepancy was reflected in the attack's performance. Therefore, ADMIS provided high Detection Ratio for FashionMNIST, but even the unprotected model could not be copied by the Copycat with the NPDD. This may have been caused by the complexity of the problem data or the small number of images used in the extraction. However, experimental tests to further investigate these causes will be left for future work.

#### 6.2.1.2 Copycat with the usual NPDD dataset

This subsection presents the results of Copycat attacks using the NPDD dataset already used in the Copycat's experiments. The attacks were performed on different sizes of the NPDD dataset, using $100k$, $300k$, and $500k$ images. The PDD datasets to fine-tune these models were the same PDD datasets used in the last subsection. The Figure 49, Figure 50 and Figure 51 show the performance obtained by the Copycats against the protected (clean bars) and unprotected (hatched bars) systems and the Detection Ratio for each dataset.

In CIFAR10 problem, the Detection Ratio achieved a value of 9.83±0.05% for NPDD datasets. Figure 45 shows the difference in label distributions between the protected and

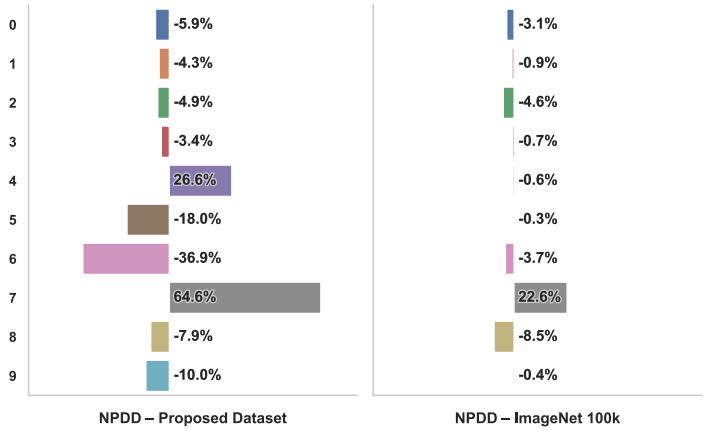



Figure 48 – FashionMNIST: difference of labels distribution between attacks on protected and unprotected system ("protected labels" – "unprotected labels"). This difference is the percentage of data from the dataset that had different labels when the system is protected. For each class, negative values represent more labels provided by the unprotected model and positive values represent more labels provided by the protected model.

unprotected systems. The label differences observed in this experiment had similar values and biases as those seen in the authors' proposed NPDD dataset (subsubsection 6.2.1.1). When comparing performance across different dataset sizes, it was observed that increasing the number of images improved the attack's performance (80.4%, 85%, and 87.5% for NPDD datasets, and 83%, 88%, and 89.2% for fine-tuning with PDD dataset). Unlike in the first experiment set, fine-tuning in this experiment provided improvements in the performance of the attacks. Using $500k$ images, it was possible to achieve a better performance (89.2%) than the first experiment set (subsubsection 6.2.1.1). Therefore, the ADMIS did not provide a good protection for this problem using these datasets.

In Flowers17 problem, the Detection Ratio achieved a value of $67.4\% \pm 0.08$ for NPDD datasets. The difference of extracted labels distribution from protected and unprotected system is shown in Figure 46. The label differences observed in this experiment had similar values and biases as those seen in the authors' proposed NPDD dataset. When comparing performance across different dataset sizes, it is observed that increasing the number of images improved the attack performance (67.9%, 73.3%, and 77.5% for NPDD datasets, and 91.6%, 90.7%, and 91.6% for fine-tuning with PDD dataset). However, NPDD achieved a lower performance on protected system and PDD achieved similar results comparing protected and unprotected system. While the attacks with NPDD dataset presented a lower performance comparing to unprotected system, it was possible to achieve a performance $> 90\%$ in all attacks after fine-tuning these models. Therefore, ADMIS provided protection for this problem against the extraction attack with the NPDD dataset. However, if the adversary possesses problem domain images, they may be able to copy the Oracle using the Copycat method.

In MNIST problem, the Detection Ratio achieved a value of $98.1 \pm 0.04$ for NPDD datasets. The difference of extracted labels distribution from protected and unprotected system is shown in Figure 47. The distribution of labels had different biases compared to the experiment with the dataset proposed by the authors. In this experiments, the biases were at class 4 and 6 instead of classes 4 and 8. As ADMIS defense provided a very high Detection Ratio value for NPDD datasets, and it was reflected in the attack performance. The attack with the NPDD dataset performed poorly on the protected system and the fine-tuning performed slightly worse than the attack on the unprotected system. In this set of experiments, the PDD dataset cannot improve the attack performance as happened in the first set of experiments (subsubsection 6.2.1.1). Although ADMIS provides protection against the attack with NPDD for this problem, the fine-tuning approach still achieved a relatively high attack success rate of 87.2% when using the NPDD dataset with $500k$ images (Figure 51).

In FashionMNIST problem, the Detection Ratio achieved a value of $23.17\% \pm 0.05$ for NPDD datasets. The difference of extracted labels distribution from protected and unprotected system is shown in Figure 48. The Detection Ratio in this experiment (23%) was notably lower than in the first set of experiments (96.9%). The distribution of labels in this experiment was also markedly different, as there were no biases observed in classes 4, 5, 6, and 9. This difference may be reflecting the higher performance of this attack in relation to the first set of experiments. Furthermore, unlike the first set of experiments, in this experiment the attack on the protected and unprotected systems had a smaller difference between them (14% to $5.3\% \pm 2.9$). The attack had higher performance by using more images, but both attacks on protected and unprotected systems did not reach high performance ($> 90\%$). Therefore, while ADMIS was effective in providing protection for this problem, it was not possible to measure its performance using problem domain data. Moreover, comparing the results of this experiment with the previous MNIST experiments suggests that fine-tuning with a problem domain dataset may lead to improved performance for this attack.

#### 6.2.1.3 Copycat of CIFAR10 using only out-of-distribution data

The authors of the OE technique indicate the use of out-of-distribution datasets, but the authors of ADMIS used ImageNet as OE dataset without removing the classes that may intersect with CIFAR10. To address this issue, we conducted new experiments using only ImageNet images that do not intersect with CIFAR10. Thus, the ImageNet composed by out-of-distribution images was used as OE dataset and also in the model extraction attack as NPDD. This section compares the results of CIFAR10 (OOD) and the CIFAR10 experiments.

After training the CIFAR10 (OOD) system with the new OE dataset, the results

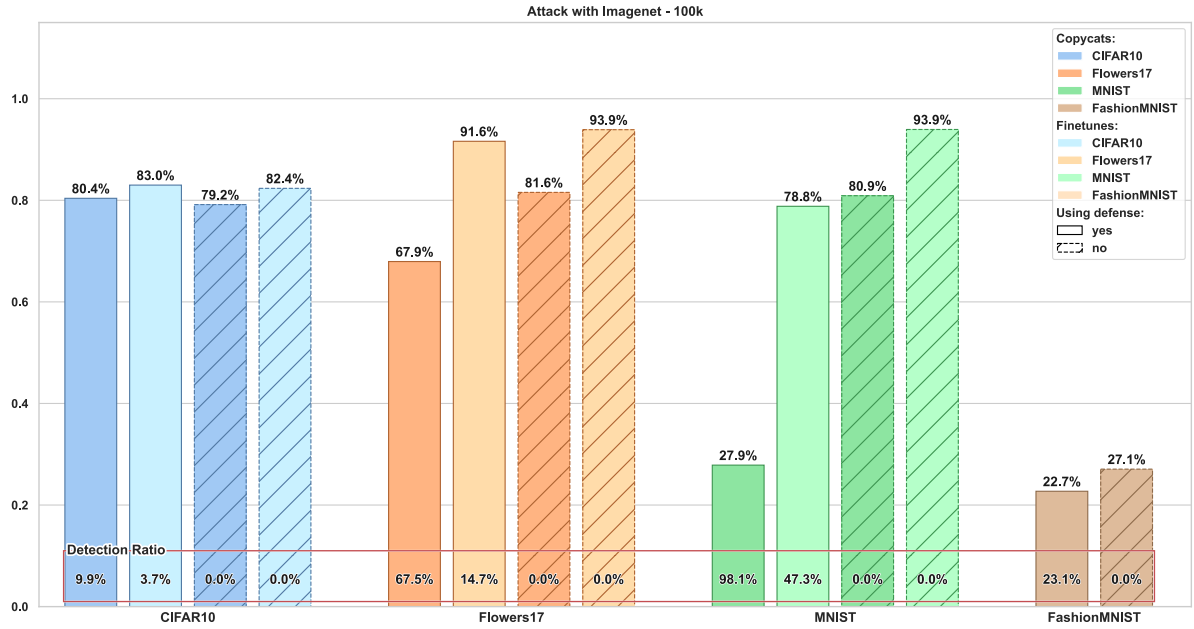



Figure 49 – Copycat results using the 100$k$ images from ImageNet as NPDD dataset. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

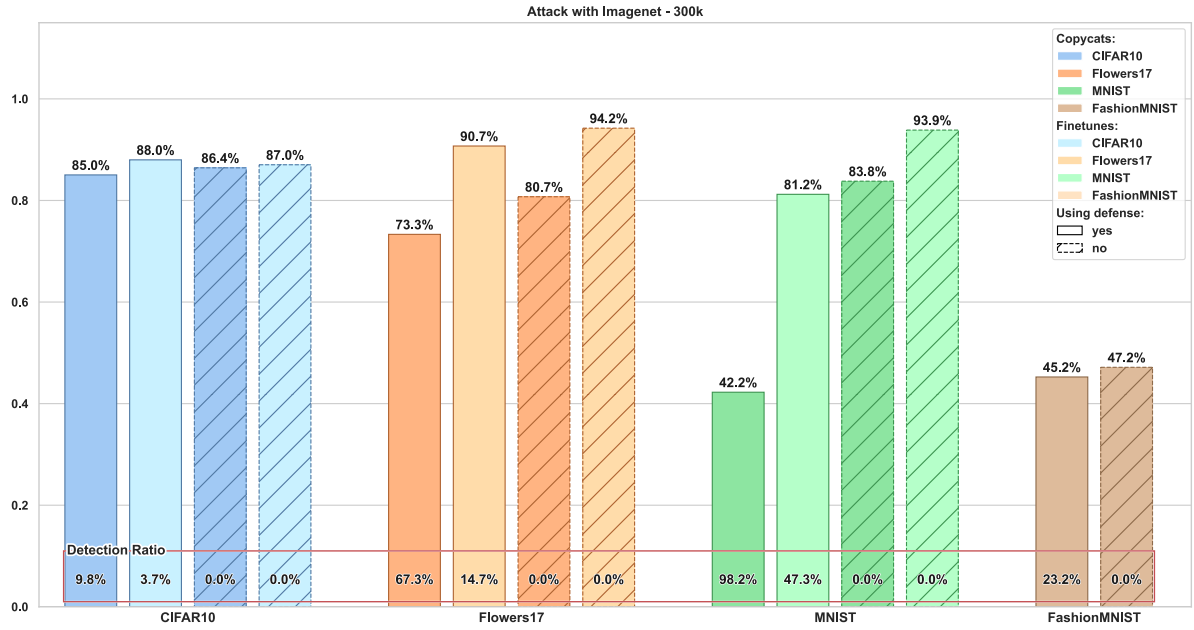



Figure 50 – Copycat results using the 300$k$ images from ImageNet as NPDD dataset. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

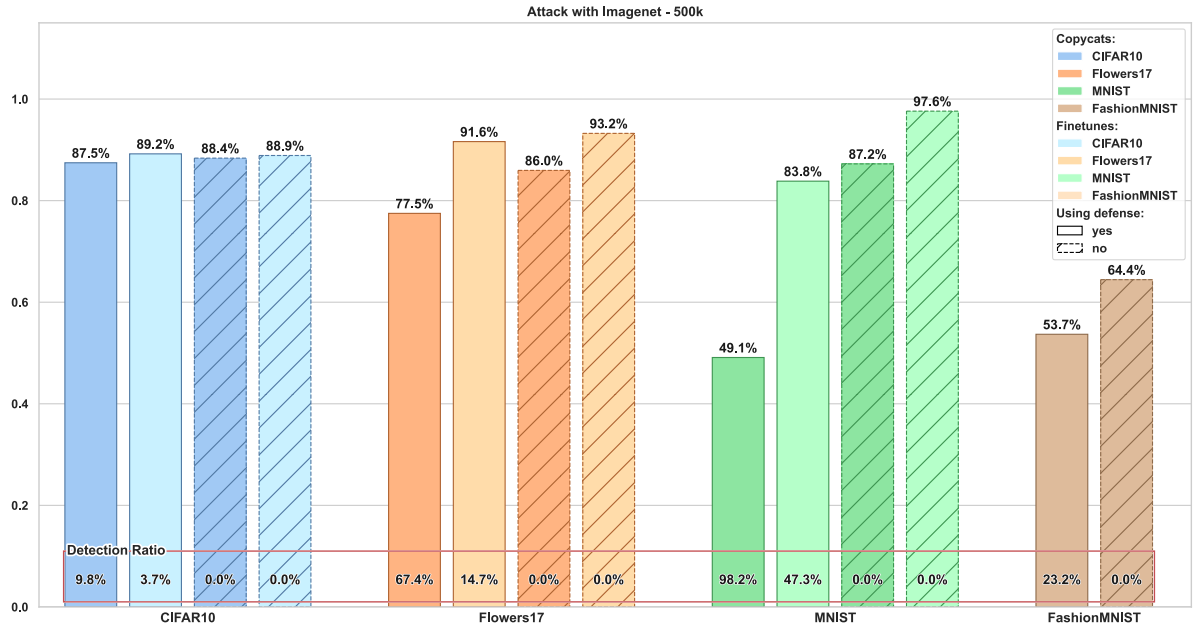



Figure 51 – Copycat results using the 500$k$ images from ImageNet as NPDD dataset. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

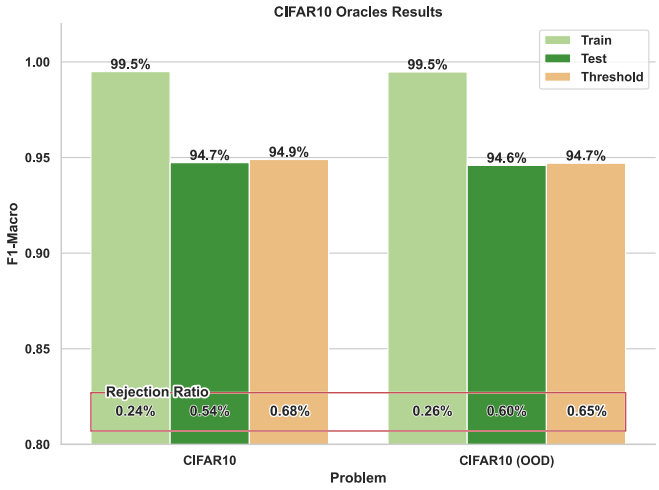



Figure 52 – ADMIS: CIFAR10's Oracles results for Train, Test and Auxiliary datasets.

presented in the Figure 52 were obtained. Comparing results with the old CIFAR10 experiments, the F1-Score and the Rejection Ratio were quite identical. We calculated the absolute differences in Rejection Ratio and F1-Scores between the CIFAR10 (OOD) and CIFAR10 datasets in the Auxiliary Dataset. To further analyze the variability of the results, we computed the variances of these differences. Our findings showed that the variance in Rejection Ratio was $5.33 \times 10^{-6}$, while the variance in F1-Macro was $7.98 \times 10^{-6}$. Moreover, the differences on Rejection Ratios in the selected threshold are very small ($< 0.1\%$). Therefore, using ImageNet OOD as OE dataset did not provide

relevant differences in these experiments.

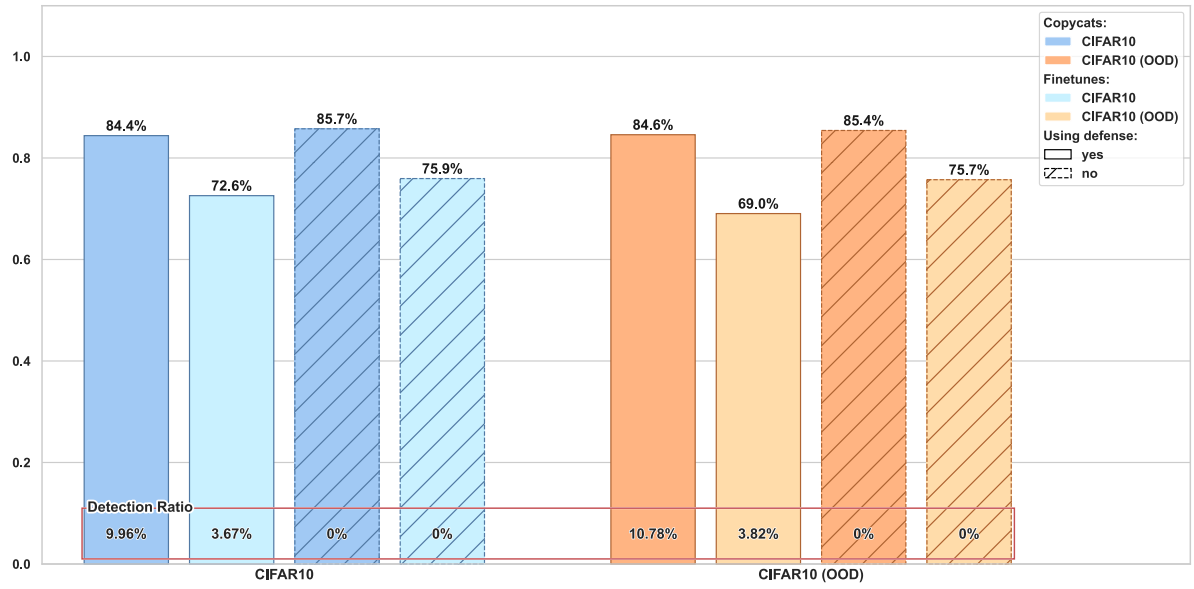



Figure 53 – Copycat results using the same attack datasets proposed by the authors of ADMIS. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

Figure 53 shows the results using the same datasets proposed by the authors of ADMIS. The results between the two models, CIFAR10 (OOD) and CIFAR10, are similar and the biggest difference was 3.6% of performance between the fine-tuned models. The differences between the Detection Ratios were also small ($< 0.83\%$). Therefore, using ImageNet OOD as the OE dataset did not significantly impact the performance of the attack.

Unlike the last results, attacks using the NPDD composed only of out-of-distribution images showed bigger differences in the experiments (Figure 54, Figure 55, and Figure 56). CIFAR10 (OOD) attacks had better results, with Copycats performing 5.4% better in attacks with $100k$ images, 3.2% in attacks with $300k$ images, and 1.3% in attacks with $500k$ images. However, it should be noted that the performance differences between CIFAR10 (OOD) and CIFAR10 decrease with an increase in the number of attack images. The fine-tuned models performed 1.6% better in attacks with $100k$ images but showed a decline of $-0.6\%$ and $-1.2\%$ for attacks with $300k$ and $500k$ images, respectively. Therefore, the use of ImageNet OOD as the OE dataset resulted in a weaker defense in these experiments.

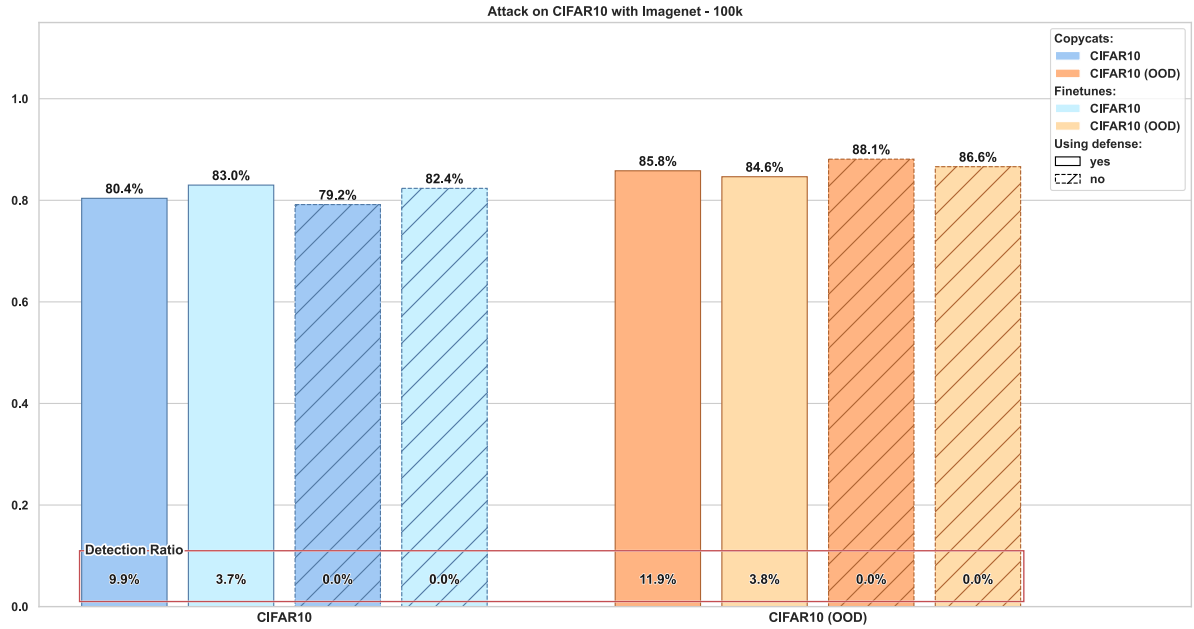



Figure 54 – Copycat results using the 100$k$ ImageNet images and using 100$k$ OOD ImageNet images as NPDD datasets. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

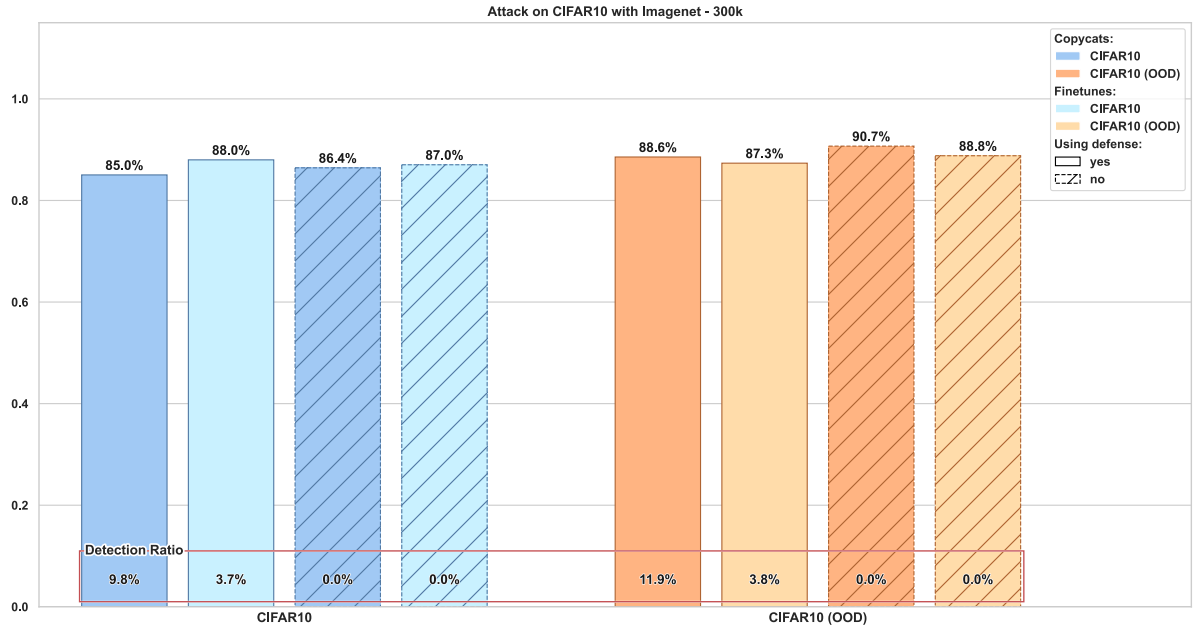



Figure 55 – Copycat results using the 300$k$ ImageNet images and using 300$k$ OOD ImageNet images as NPDD datasets. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

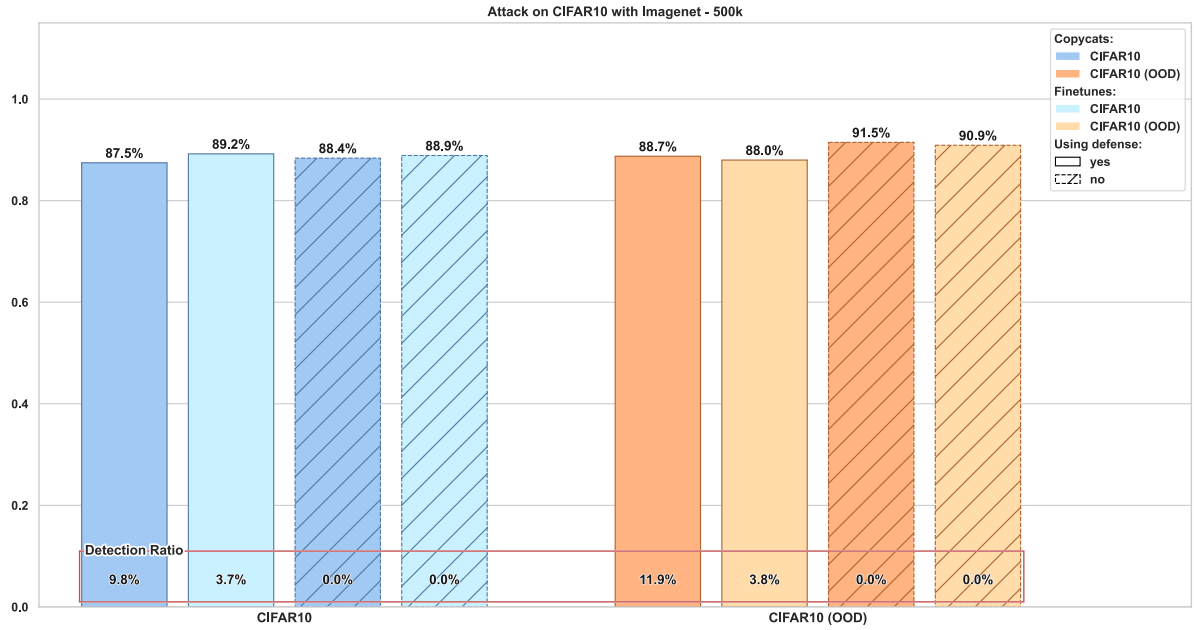



Figure 56 – Copycat results using the 500$k$ ImageNet images and using 500$k$ OOD ImageNet images as NPDD datasets. Dark colors represent the Copycat performance and light colors represent the fine-tuned model performance. The straight line is used to represent the protected systems and the dashed line is used as the edge to represent unprotected systems.

## 6.2.2 PRADA

This section presents the results of experiments with the PRADA defense method. Firstly, we show the results for MNIST on both the small architecture and the VGG-16 architecture. Next, we present the results for GTSRB on both the small architecture and the VGG-16 architecture. Then, we show the results for GOC9 on the VGG-16 architecture. Finally, we discuss about the detection performed by the PRADA defense method and its impact on the Copycat attack.

### 6.2.2.1 MNIST on Small Architecture

This subsection presents the experiments results for the small model (Table 6, first column) trained with MNIST dataset. Figure 57 shows the Statistical Test Value for False Positive (FP) Rates (a,b) and for Detection Rates (c). The smallest statistical value is marked as a cross for each dataset, which represents the first false positive occurrence in the system (a,b) or the first detected attacks (c) when increasing the Statistical test value.

These results were analyzed as this example: looking at TDD and PDD (USPS) on sorted datasets (Figure 57 (a)), the false positives occurs at the statistical test value of 0.72 and 0.98, so the threshold should be 0.71 (one-hundredth less than the lowest statistical value). However, looking the values of Figure 57 (c), the detection of the attack occurs only when statistical test value is bigger than 0.90, thus the attack is not detected

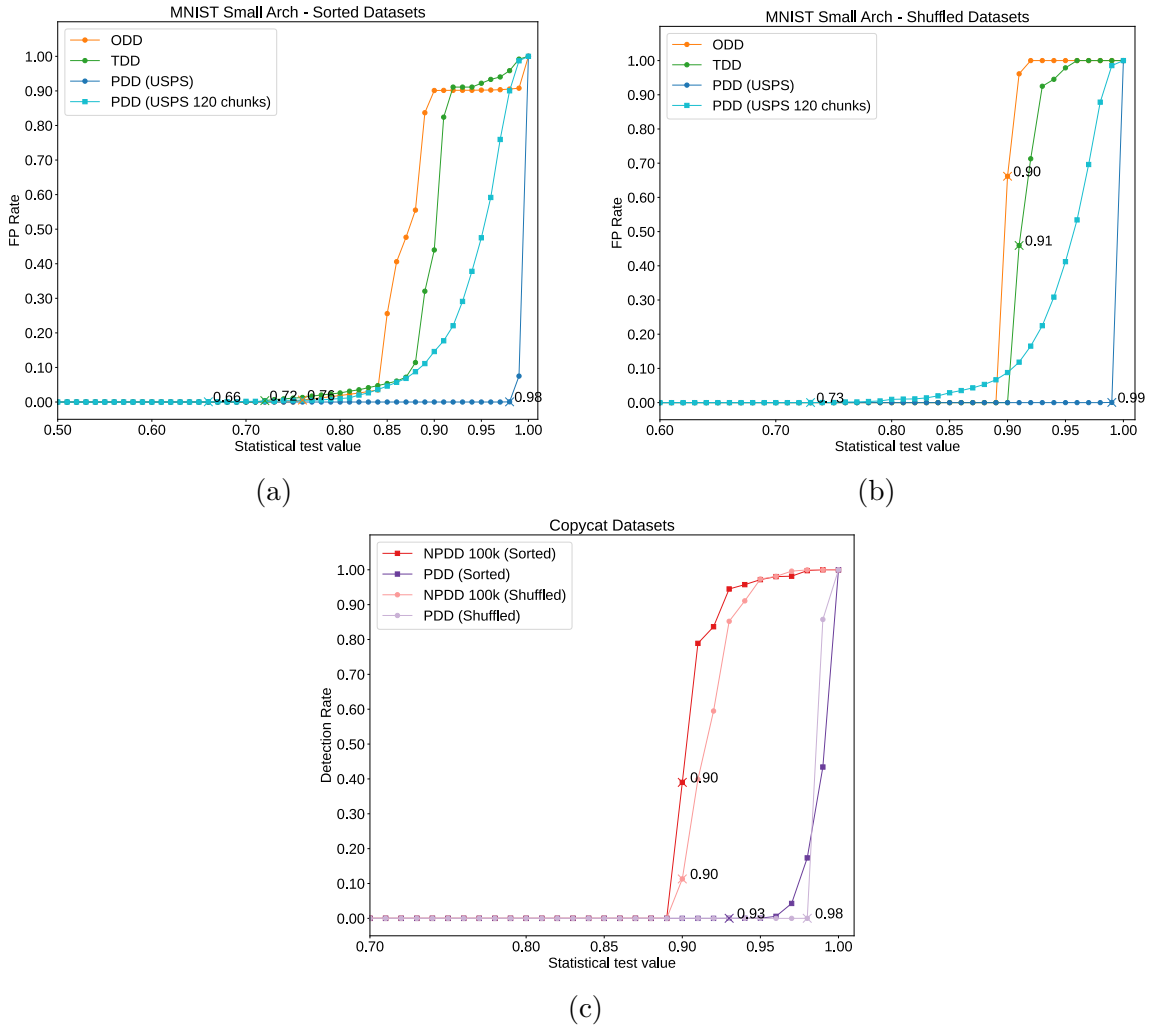



Figure 57 – FP (False Positive) Rate (a,b) and Detection Rate (c) for Statistical test value obtained with the model trained with MNIST dataset on the small architecture.

using this threshold.

In order to facilitate comparisons between the results, the Table 10 has been filled in with the thresholds for each dataset (first and second columns) and the attack detection rate (columns 4-7) for each threshold (third column). Additionally, the dataset groups discussed in the Subsection 5.7.2 make up the second part of the table (Selecting Threshold Value), where each column refers to the thresholds of each group. Moreover, the selected threshold (i.e., lowest value) for each group is highlighted in green. For the group O2, for example, the thresholds are 0.71 (TDD) and 0.97 (USPS) and the selected threshold is 0.71. Then following this row with 0.71 highlighted, we can see that any attack query can be detected (Table 10, columns 4-7). As the training data (ODD) were not used in the original article to select the threshold, it is presented in the table only to verify if there would be changes with its use.

As can be observed in Table 10, for groups O1, O2, O3 and O4, the threshold

value selected is 0.71, which does not detect Copycat queries (NPDD and PDD columns). Therefore, it does not provide attack detection for the model. On the other hand, for groups S1 and S2, the selected threshold value of 0.90 can detect 39% of Sorted NPDD dataset and 11.31% of Shuffled NPDD dataset, but cannot detect Copycat PDD dataset. In addition to these results, when the USPS chunks are used to select the threshold for groups S3 (as suggested by the authors of PRADA) and S4, the Copycat attack is not detected. Furthermore, even if the OD thresholds were considered in S1 and S2 groups, the attack would also not be detected.

Table 10 – MNIST dataset trained on the Small architecture. The Threshold column presents the threshold to avoid false positives. The NPDD and PDD columns show the rate of detected attack queries. Each column of "Selecting Threshold Value" section refers to a group (presented in the Subsection 5.7.2) of datasets used to select the PRADA's threshold. The highlighted value on these columns refers to the threshold selected for the group. The OD thresholds were presented only for analysis.

| **MNIST Small Arch** | | **Threshold** | **NPDD** | | **PDD** | | **Selecting Threshold Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Sorted** | **Shuffled** | **Sorted** | **Shuffled** | **O1** | **O2** | **O3** | **O4** | **S1** | **S2** | **S3** | **S4** |
| ODD | Ordered | 0.75 | 0% | 0% | 0% | 0% | 0.75 | 0.75 | 0.75 | 0.75 | | | | |
| | Shuffled | 0.89 | 0% | 0% | 0% | 0% | | | | | 0.89 | 0.89 | 0.89 | 0.89 |
| TDD | Ordered | 0.71 | 0% | 0% | 0% | 0% | 0.71 | 0.71 | 0.71 | 0.71 | | | | |
| | Shuffled | 0.90 | 39.00% | 11.31% | 0% | 0% | | | | | 0.90 | 0.90 | 0.90 | 0.90 |
| USPS | Ordered | 0.97 | 98.16% | 99.61% | 4.28% | 0% | | 0.97 | | 0.97 | | | | |
| | Shuffled | 0.98 | 99.75% | 99.96% | 17.00% | 0% | | | | | | 0.98 | | 0.98 |
| USPS (chunks) | Ordered | 0.73 | 0% | 0% | 0% | 0% | | | 0.73 | 0.73 | | | | |
| | Shuffled | 0.65 | 0% | 0% | 0% | 0% | | | | | | | 0.65 | 0.65 |

#### 6.2.2.2 MNIST on VGG16 Architecture

This subsection presents the experiments results for the VGG16 model trained with MNIST dataset. Figure 58 shows the Statistical Test Value for False Positive (FP) Rates (a,b) and for Detection Rates (c). The smallest statistical value is marked as a cross for each dataset, which represents the first false positive occurrence in the system (a,b) or the first detected attacks (c) when increasing the Statistical test value.

In order to facilitate comparisons between the results, the Table 11 has been filled in with the thresholds for each dataset and the attack detection rate for each threshold. Additionally, the dataset groups discussed in the Subsection 5.7.2 make up the second part of the table (Selecting Threshold Value), where each column refers to the thresholds of each group. Moreover, the selected threshold (i.e., lowest value) for each group is highlighted in green. For the group O2, for example, the thresholds are 0.94 (TDD) and 0.92 (USPS) and the selected threshold is 0.92. Then following this row with 0.92 highlighted, we can see that 99.44% of Sorted NPDD is detected and all queries of Shuffled NPDD dataset are detected, but any query from PDD dataset can be detected (Table 11, columns 4-7). As the training data (ODD) were not used in the original article to select the threshold, it is

presented in the table only to verify if there would be changes with its use.

As can be observed in Table 11, for groups O1 and O2, all queries from Shuffled NPDD dataset can be detected and more than 99% queries of Sorted NPDD dataset can be detected. And for S1 and S2, the thresholds can detect all queries of NPDD dataset and can detect 17.60% and 8.98% of Sorted PDD dataset, respectively, but cannot detect Shuffled PDD dataset. On the other hand, when it is used the USPS chunks, as suggested by the authors of PRADA, the threshold for groups O3, O4, S3 and S4 cannot detect any attack query. Additionally, the atack is also not detected when ODD thresholds are considered.

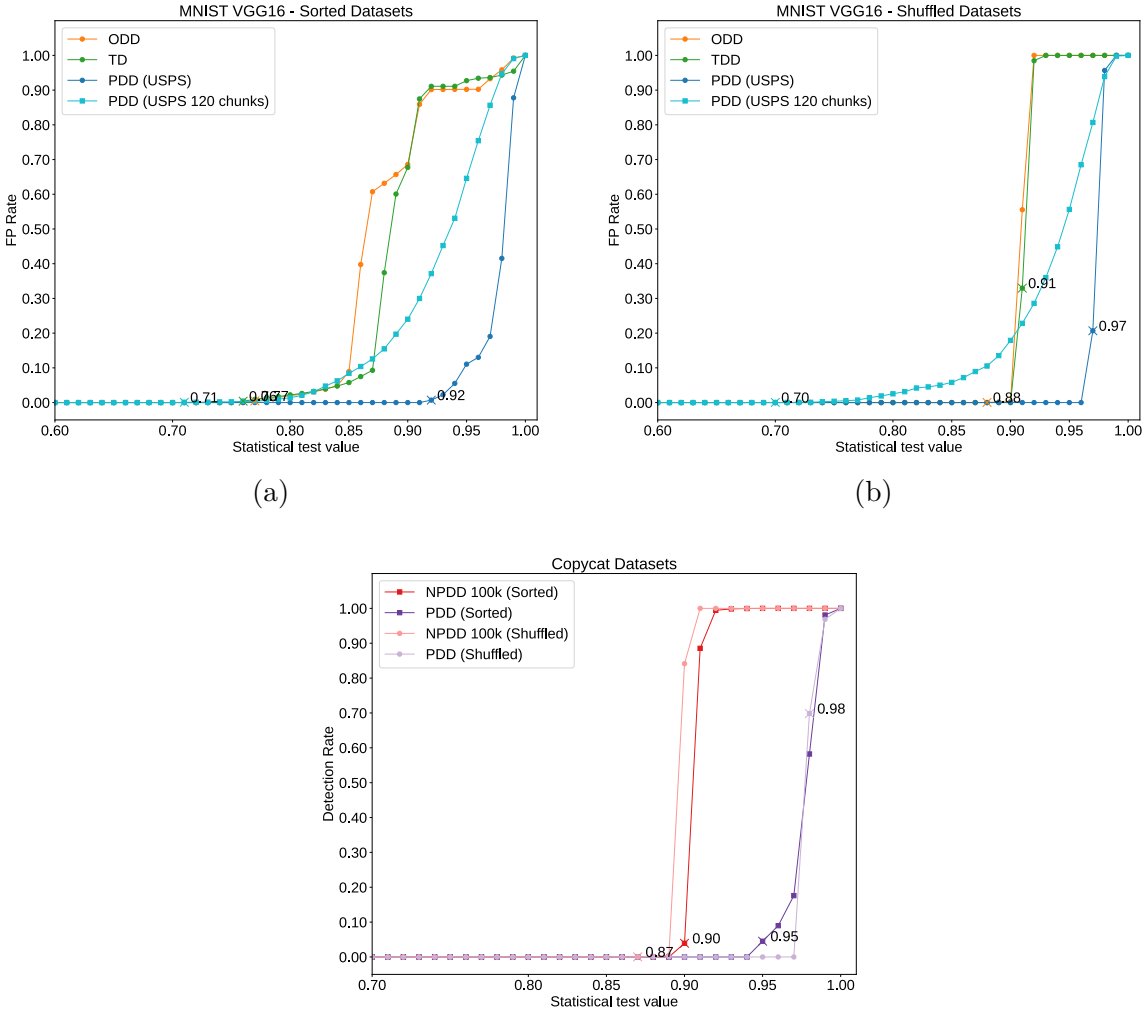



Figure 58 – FP (False Positive) Rate (a,b) and Detection Rate (c) for Statistical test value obtained with the model trained with MNIST dataset on VGG16 architecture.

Table 11 – MNIST Dataset trained on VGG-16. The Threshold column presents the threshold to avoid false positives. The NPDD and PDD columns show the rate of detected attack queries. Each column of "Selecting Threshold Value" section refers to a group (presented in the Subsection 5.7.2) of datasets used to select the PRADA's threshold. The highlighted value on these columns refers to the threshold selected for the group. The OD thresholds were presented only for analysis.

| **MNIST VGG-16 Arch** | | **Threshold** | **NPDD** | | **PDD** | | **Selecting Threshold Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **Sorted** | **Shuffled** | **Sorted** | **Shuffled** | **O1** | **O2** | **O3** | **O4** | **S1** | **S2** | **S3** | **S4** |
| ODD | Ordered | 0.76 | 0% | 0% | 0% | 0% | 0.76 | 0.76 | 0.76 | 0.76 | | | | |
| | Shuffled | 0.87 | 0% | 0% | 0% | 0% | | | | | 0.87 | 0.87 | 0.87 | 0.87 |
| TDD | Ordered | 0.94 | 99.96% | 100% | 0% | 0% | 0.94 | 0.94 | 0.94 | 0.94 | | | | |
| | Shuffled | 0.97 | 100% | 100% | 17.60% | 0% | | | | | 0.97 | 0.97 | 0.97 | 0.97 |
| USPS | Ordered | 0.92 | 99.44% | 100% | 0% | 0% | | 0.92 | | 0.92 | | | | |
| | Shuffled | 0.96 | 100% | 100% | 8.98% | 0% | | | | | | 0.96 | | 0.96 |
| USPS (chunks) | Ordered | 0.69 | 0% | 0% | 0% | 0% | | | 0.69 | 0.69 | | | | |
| | Shuffled | 0.70 | 0% | 0% | 0% | 0% | | | | | | | 0.70 | 0.70 |

#### 6.2.2.3 GTSRB on Small Architecture

This subsection presents the experiments results for the small model (Table 6, second column) trained with GTSRB dataset. The Figure 59 shows the Statistical Test Value for False Positive (FP) Rates (a,b) and for Detection Rates (c). The smallest statistical value is marked as a cross for each dataset, which represents the first false positive occurrence in the system (a,b) or the first detected attacks (c) when increasing the Statistical test value.

In order to facilitate comparisons between the results, the Table 12 has been filled in with the thresholds for each dataset and the attack detection rate for each threshold. Additionally, the dataset groups discussed in the Subsection 5.7.2 make up the second part of the table (Selecting Threshold Value), where each column refers to the thresholds of each group. Moreover, the selected threshold (i.e., lowest value) for each group is highlighted in green. For the group O2, for example, the thresholds are 0.75 (TDD) and 0.82 (BTS). Then following this row with 0.75 highlighted, we can see that any attack query can be detected but only 0.01% of the Sorted PDD dataset (Table 12, columns 4-7). As the training data (ODD) were not used in the original article to select the threshold, it is presented in the table only to verify if there would be changes with its use.

As can be observed in Table 12, for groups O1, O2, O3 and O4, the threshold is 0.75 and for the following groups S1, S2, S3 and S4, the threshold is 0.74 which both only detect 0.01% of Sorted PDD dataset, therefore, it does not provide attack protection for the model. In group S2 and S4, the threshold for BTS if 0.93, which detects more than 90% of NPDD queries. However, this threshold also detects more than 99% of PDD queries as an attack. Moreover, when it is used the USPS chunks, as suggested by the authors of PRADA, the threshold for groups O3, O4, S3 and S4 cannot detect any attack query.

Additionally, the atack is also not detected when ODD thresholds are considered.

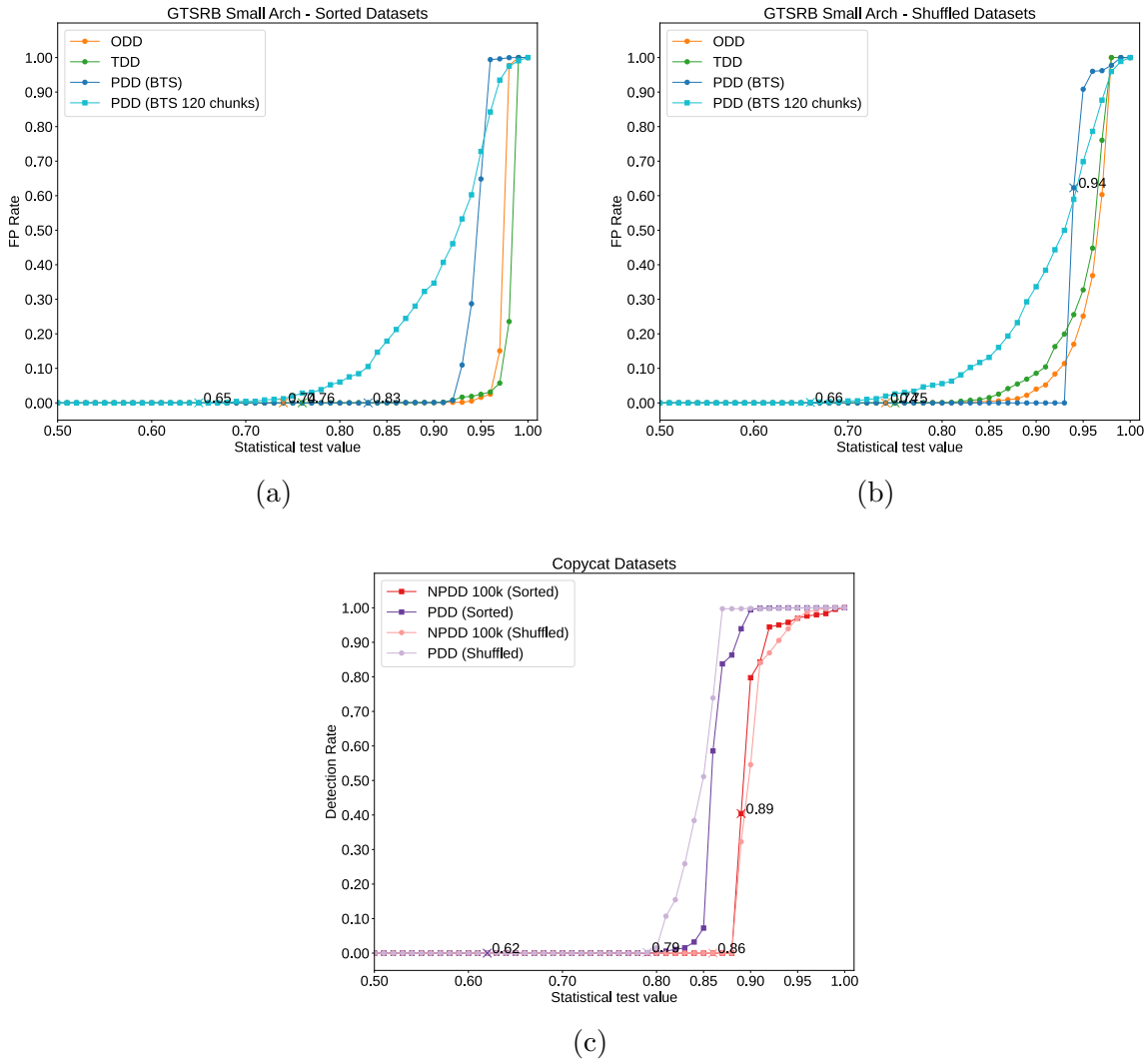



Figure 59 – FP (False Positive) Rate (a,b) and Detection Rate (c) for Statistical test value obtained with the model trained with GTSRB dataset on the small architecture.

#### 6.2.2.4 GTSRB on VGG16 Architecture

This subsection presents the experiments results for the VGG16 model trained with GTSRB dataset. The Figure 60 shows the Statistical Test Value for False Positive (FP) Rates (a,b) and for Detection Rates (c). The smallest statistical value is marked as a cross for each dataset, which represents the first false positive occurrence in the system (a,b) or the first detected attacks (c) when increasing the Statistical test value.

In order to facilitate comparisons between the results, the Table 13 has been filled in with the thresholds for each dataset and the attack detection rate for each threshold. Additionally, the dataset groups discussed in the Subsection 5.7.2 make up the second part of the table (Selecting Threshold Value), where each column refers to the thresholds of each

Table 12 – GTSRB Dataset trained on the Small architecture. The Threshold column presents the threshold to avoid false positives. The NPDD and PDD columns show the rate of detected attack queries. Each column of "Selecting Threshold Value" section refers to a group (presented in the Subsection 5.7.2) of datasets used to select the PRADA's threshold. The highlighted value on these columns refers to the threshold selected for the group. The OD thresholds were presented only for analysis.

| GTSRB Small Arch | | Threshold | NPDD | | PDD | | Selecting Threshold Value | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sorted | Shuffled | Sorted | Shuffled | O1 | O2 | O3 | O4 | S1 | S2 | S3 | S4 |
| ODD | Ordered | 0.73 | 0% | 0% | 0.01% | 0% | 0.73 | 0.73 | 0.73 | 0.73 | | | | |
| | Shuffled | 0.73 | 0% | 0% | 0.01% | 0% | | | | | 0.73 | 0.73 | 0.73 | 0.73 |
| TDD | Ordered | 0.75 | 0% | 0% | 0.01% | 0% | 0.75 | 0.75 | 0.75 | 0.75 | | | | |
| | Shuffled | 0.74 | 0% | 0% | 0.01% | 0% | | | | | 0.74 | 0.74 | 0.74 | 0.74 |
| BTS | Ordered | 0.82 | 0% | 0% | 1.09% | 15.46% | | 0.82 | | 0.82 | | | | |
| | Shuffled | 0.93 | 95.03% | 90.54% | 99.96% | 99.88% | | | | | | 0.93 | | 0.93 |
| BTS (chunks) | Ordered | 0.82 | 0% | 0% | 1.09% | 15.46% | | | 0.82 | 0.82 | | | | |
| | Shuffled | 0.93 | 95.03% | 90.54% | 99.96% | 99.88% | | | | | | | 0.93 | 0.93 |

group. Moreover, the selected threshold (i.e., lowest value) for each group is highlighted in green. For the group 03, for example, the thresholds are 0.93 (TDD) and 0.65 (BTS) and the selected threshold is 0.65. Then following this row with 0.65 highlighted, we can see that any Copycat query can be detected (Table 13, columns 4-7). As the training data (ODD) were not used in the original article to select the threshold, it is presented in the table only to verify if there would be changes with its use.

As can be observed in Table 13, for groups O1 and O2, almost all Copycat queries can be detected, including NPDD and PDD. On the other hand, with the thresholds of the other groups, including the groups that use BTS (chunks), which was suggested by the authors of PRADA, the attack is not detected. Furthermore, the ODD thresholds would not provide differences in the system behavior.

Table 13 – GTSRB Dataset trained on VGG-16. The Threshold column presents the threshold to avoid false positives. The NPDD and PDD columns show the rate of detected attack queries. Each column of "Selecting Threshold Value" section refers to a group (presented in the Subsection 5.7.2) of datasets used to select the PRADA's threshold. The highlighted value on these columns refers to the threshold selected for the group. The OD thresholds were presented only for analysis.

| GTSRB VGG-16 Arch | | Threshold | NPDD | | PDD | | Selecting Threshold Value | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sorted | Shuffled | Sorted | Shuffled | O1 | O2 | O3 | O4 | S1 | S2 | S3 | S4 |
| ODD | Ordered | 0.93 | 98.42% | 100% | 98.73% | 100% | 0.93 | 0.93 | 0.93 | 0.93 | | | | |
| | Shuffled | 0.86 | 0% | 0.12% | 0% | 7.95% | | | | | 0.86 | 0.86 | 0.86 | 0.86 |
| TDD | Ordered | 0.93 | 98.42% | 100% | 98.73% | 100% | 0.93 | 0.93 | 0.93 | 0.93 | | | | |
| | Shuffled | 0.80 | 0% | 0% | 0% | 0% | | | | | 0.80 | 0.80 | 0.80 | 0.80 |
| BTS | Ordered | 0.93 | 98.42% | 100% | 98.73% | 100% | | 0.93 | | 0.93 | | | | |
| | Shuffled | 0.93 | 99.42% | 100% | 98.73% | 100% | | | | | | 0.93 | | 0.93 |
| BTS (chunks) | Ordered | 0.65 | 0% | 0% | 0% | 0% | | | 0.65 | 0.65 | | | | |
| | Shuffled | 0.59 | 0% | 0% | 0% | 0% | | | | | | | 0.59 | 0.59 |

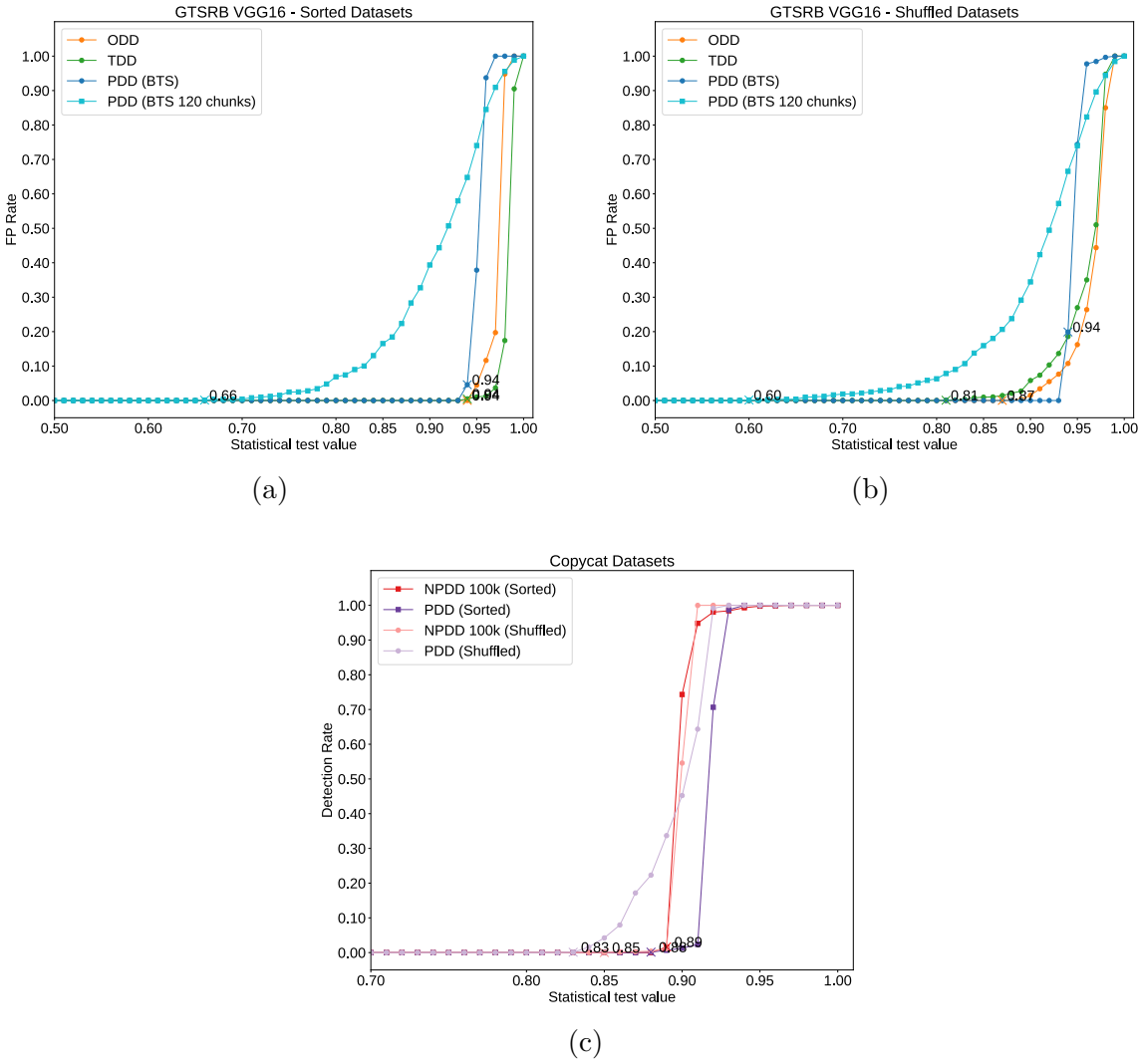



Figure 60 – FP (False Positive) Rate (a,b) and Detection Rate (c) for Statistical test value obtained with the model trained with GTSRB dataset on VGG16 architecture.

### 6.2.2.5 GOC9

This section presents an extension of the experiments on the GOC9 problem, which was already part of the Copycat experiments (Subsection 5.2.4). The problem consists of a more complex dataset trained on the VGG-16 model. In these experiments, we also aimed to deceive the PRADA defense system. To achieve this, we added a small set of problem domain images to the NPDD dataset. This small group, comprising 6.36% of the images, contains 5 categories of problem domain images. This new dataset is referred to as NPDD (5 cat.D). When the dataset is sorted, the first images are from the problem domain. In contrast, when the dataset is shuffled, the images have no specific order. We believe that this dataset can deceive the defense system because these problem domain images can maintain the normal distribution of queries made by the adversary when using the NPDD dataset.

Figure 60 shows the False Positive (FP) (a,b) and Detection Rates (c,d) for Statistical Test Value. The NPDD (5 cat.D) is represented on Figure 60 (d). The smallest statistical value is marked as a cross for each dataset, which represents the first false positives in the system (a,b) or the first detected attacks (c,d) when increasing the Statistical test value.

In order to facilitate comparisons between the results, the Table 14 has been filled in with the thresholds for each dataset and the attack detection rate for each threshold. Additionally, the dataset groups discussed in the Subsection 5.7.2 make up the second part of the table (Selecting Threshold Value), where each column refers to the thresholds of each group. Moreover, the selected threshold (i.e., lowest value) for each group is highlighted in green. For the group 03, for example, the thresholds are 0.84 (TDD) and 0.55 (BTS) and the selected threshold is 0.55. Then following this row with 0.55 highlighted, we can see that any Copycat query can be detected (Table 13, columns 4-7). As the training data (ODD) were not used in the original article to select the threshold, it is presented in the table only to verify if there would be changes with its use.

As can be observed in Table 14, for groups O1, O2, and S2, 7.01% (Sorted) and 18.33% (Shuffled) of NPDD queries were detected, but on NPDD (5 cat.D) these rates decrease to 0% and 0.03%, respectively. The PDD queries are detected 0.7% only when its images are shuffled. The detection also occurs on group S1, with 30.2% (sorted) and 60.76% (shuffled) for NPDD. However, these rates decrease to 0% (sorted) and 0.14% on NPDD (5 cat.D). On the other hand, on O3, O4, S3 (as suggested by the PRADA article) and S4, the Copycat attack is not detected. Moreover, if ODD thresholds were considered, the detection rate would decrease for O1, O2, O3, and O4 thresholds, but the system behavior would remain the same for S1,S2,S3,S4 thresholds.

Table 14 – GOC9 problem trained on VGG-16. The Threshold column presents the threshold to avoid false positives. The NPDD, the NPDD (5 cat.D), and PDD columns show the rate of detected attack queries. Each column of "Selecting Threshold Value" section refers to a group (presented in the Subsection 5.7.2) of datasets used to select the PRADA's threshold. The highlighted value on these columns refers to the threshold selected for the group. The OD thresholds were presented only for analysis.

| GOC9 VGG-16 Arch | | Threshold | NPDD | | NPDD (5 cat.D) | | PDD | | Selecting Threshold Value | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Sorted | Shuffled | Sorted | Shuffled | Sorted | Shuffled | O1 | O2 | O3 | O4 | S1 | S2 | S3 | S4 |
| ODD | Ordered | 0.81 | 0% | 0.78% | 0% | 0% | 0% | 0% | 0.81 | 0.81 | 0.81 | 0.81 | | | | |
| | Shuffled | 0.88 | 99.93% | 100% | 0.05% | 1.47% | 38.15% | 9.96% | | | | | 0.88 | 0.88 | 0.88 | 0.88 |
| TDD | Ordered | 0.84 | 7.01% | 18.33% | 0% | 0.03% | 0% | 0.7% | 0.84 | 0.84 | 0.84 | 0.84 | | | | |
| | Shuffled | 0.85 | 30.2% | 60.76% | 0% | 0.14% | 6.10% | 1.57% | | | | | 0.85 | 0.85 | 0.85 | 0.85 |
| STL | Ordered | 0.84 | 7.01% | 18.33% | 0% | 0.03% | 0% | 0.7% | | 0.84 | | 0.84 | | | | |
| | Shuffled | 0.84 | 7.01% | 18.33% | 0% | 0.03% | 0% | 0.7% | | | | | | 0.84 | | 0.84 |
| STL (chunks) | Ordered | 0.55 | 0% | 0% | 0% | 0% | 0% | 0% | | | 0.55 | 0.55 | | | | |
| | Shuffled | 0.53 | 0% | 0% | 0% | 0% | 0% | 0% | | | | | | | 0.53 | 0.53 |

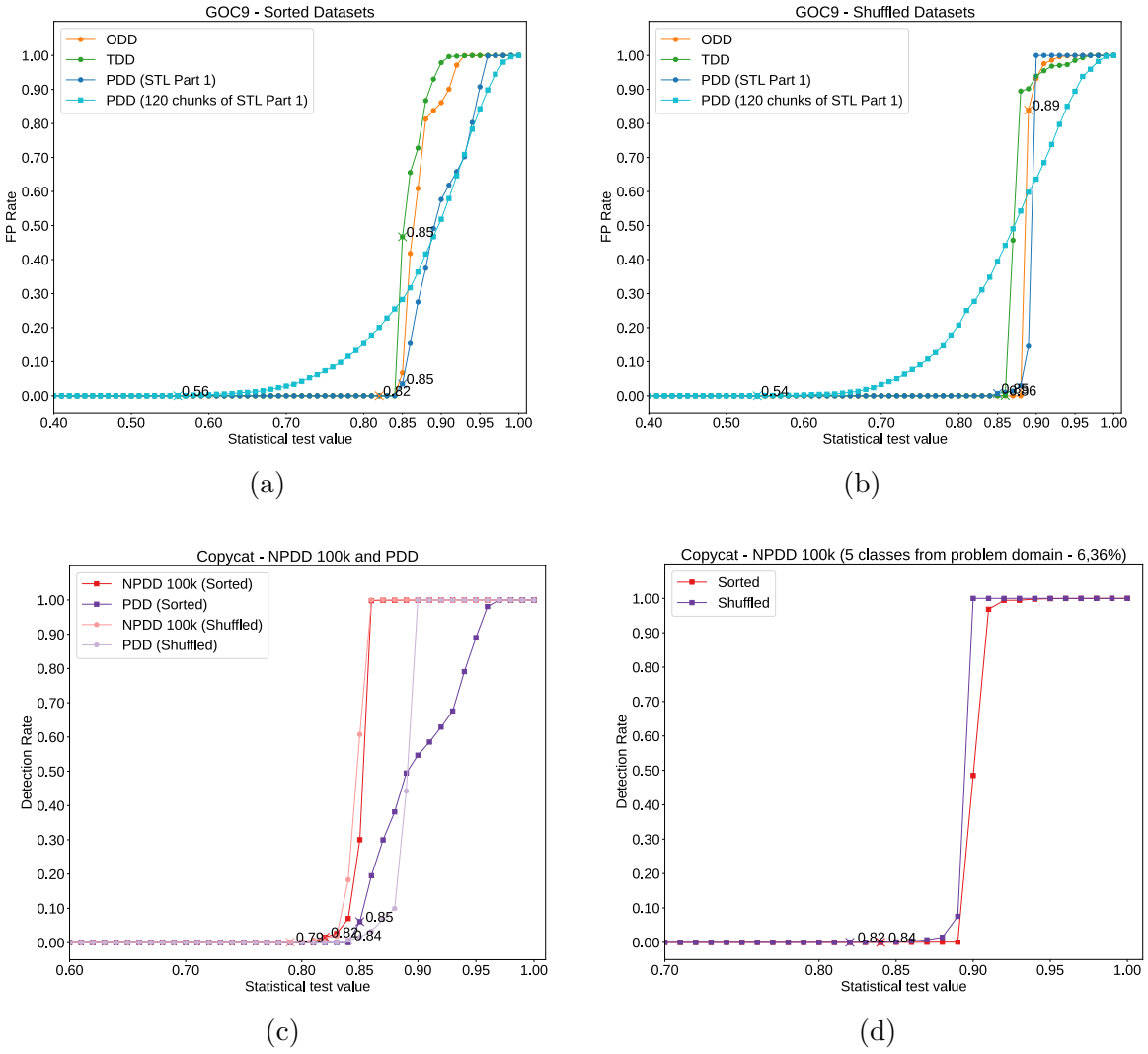



Figure 61 – FP (False Positive) Rate (a,b) and Detection Rate (c,d) for Statistical test value obtained with the GOC9 problem.

#### 6.2.2.6 Discussion

Based on the experiments, this method cannot be used to defend a model against a Copycat attack while avoiding false positives. It happens because selecting the thresholds as defined in the main paper (group S3 on the Table 10, Table 11, Table 12, Table 13, and Table 14) does not work to avoid attacks, i.e, to detect NPDD and PDD queries. When selecting other groups, like O2 and S2 in MNIST on VGG16 (Table 11), or O2 in GTSRB on VGG16 (Table 13), or O2 and S1 in GOC9 (Table 14), the Copycat NPDD queries can be detected, but PDD queries are also detected (that is not expected, as they are part of the same problem domain). Furthermore, these groups do not provide good thresholds for other models. Additionally, the threshold 0.93 of group O2 in GTSRB on VGG16 (Table 13) generates false positives for the same TD queries when they are shuffled (Figure 60 (b)).

Therefore, there is no rule or common method to define the correct threshold for each problem. Additionally, the PRADA method is computationally expensive. Since it stores all processed queries in computer memory to calculate distances for new images, it requires significant computational resources to operate efficiently. As a result, the system can become quite slow when processing larger batches of data. For instance, it takes 6 hours and 40 minutes to process $100k$ queries using a machine with an i5 processor and 62 GB of RAM. This limits the method's suitability for an analysis on huge batches of queries.

### 6.2.3 EWE: Entangled Watermarks

The experiment results for the Oracles are shown in Figure 62, which displays the F1-Scores of the Oracles. Additionally on the Figure, the Entangled Watermark box displays the identified watermarked rate for watermarked models and the false positive rate for clean models. In the experiments, the CIFAR10 has a small identified watermarked rate of 15.3(±9.90)% and the false positive rate close to zero, so the watermark success rate was 15.3(±9.9)%. Additionally, the MNIST model detected almost all watermarked data (identified watermarked rate) and the false positive rate also close to zero, so the watermark success rate was 99.9(±0.1)%. Moreover, the FashionMNIST also presented a high rate of identified watermarks (100(±0.01)%) but its false positive rate was 28.1(±12.17)% that decreased the watermarked success rate to 71.9(±12.17)%. To make more readable, these results are presented in Table 15.

Table 15 – Watermarked success rates of the Oracles.

| **Problem** | **Identified watermarked rate** | **False positive rate** | **Watermark success rate** |
|---|---|---|---|
| CIFAR10 | 15.3(±9.90)% | 0.0(±0.01)% | 15.3(±9.90)% |
| MNIST | 100.0(±0.02)% | 0.1(±0.08)% | 99.9(±0.1)% |
| FashionMNIST | 100.0(±0.01)% | 28.1(±12.17)% | 71.9(±12.17)% |

The results of the Copycat experiments are shown in Figure 63. As can be seen in the Entangled Watermark box, the watermark detection persists even after the Copycat attack. Furthermore, the Copycat rates are very similar to the rates obtained in Oracle results (Figure 62). In CIFAR10, the watermark success rate was 16.51(±2.74)% for Copycat model and 20.05(±0.88)% for its fine-tuned model. Note that fine-tuning increased the watermark success rate. In MNIST, the watermark success rate was 98.73(±1.05)% for Copycat model and 99.37(±0.28)% for its fine-tuned model. And for FashionMNIST, the watermarked success rate was 70.96(±16.43)% for Copycat model. To make more readable, these results are presented in Table 16 and Table 17. Therefore, the EWE method was able to defend the Intellectual Property of the tested models against the Copycat method.

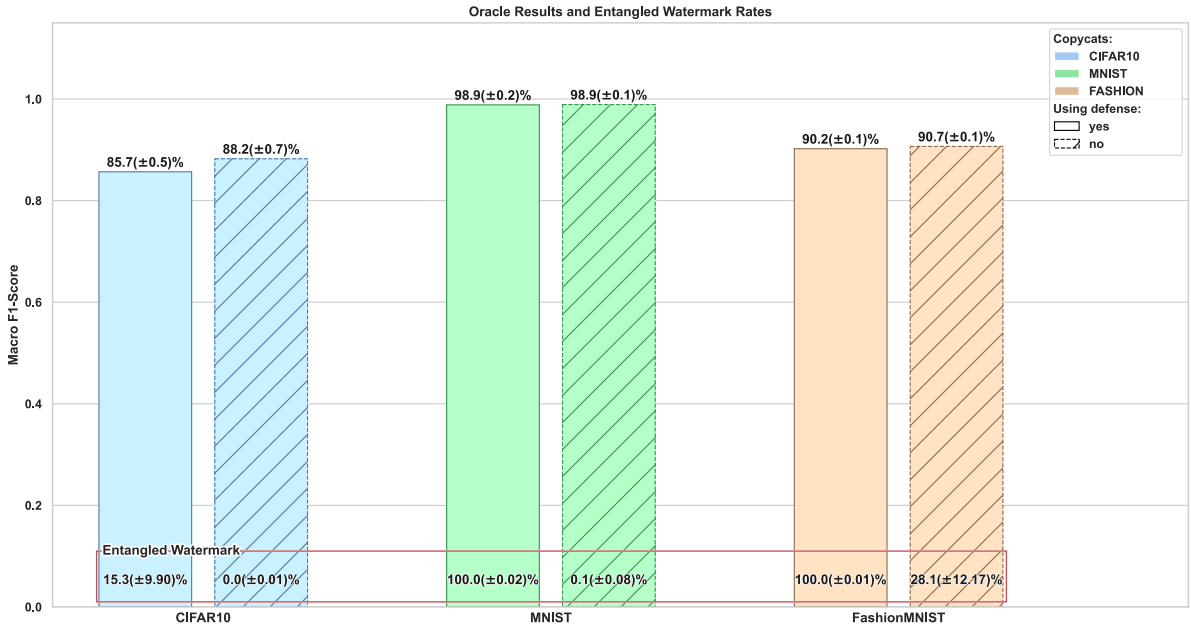



Figure 62 – EWE: Oracles results. The straight line as the edge is used in watermarked models and the dashed line is used in clean models. The red box shows the identified watermarked rate for watermarked models and the false positive rate for clean models.

Table 16 – Watermarked success rates of the Copycat models.

| **Problem** | **Identified watermarked rate** | **False positive rate** | **Watermark success rate** |
|---|---|---|---|
| CIFAR10 | 16.8(±2.6)% | 0.3(±0.2)% | 16.51(±2.74)% |
| MNIST | 99.6(±0.2)% | 0.9(±0.9)% | 98.73(±1.05)% |
| FashionMNIST | 99.5(±0.4)% | 28.5(±16.8)% | 70.96(±16.43)% |

Table 17 – Watermarked success rates of the Copycat models after fine-tuning process.

| **Problem** | **Identified watermarked rate** | **False positive rate** | **Watermark success rate** |
|---|---|---|---|
| CIFAR10 | 20.5(±0.7)% | 0.5(±0.3)% | 20.05(±0.88)% |
| MNIST | 99.7(±0.2)% | 0.3(±0.5)% | 99.37(±0.28)% |

Additionally, we also analyzed the discrepancy in stolen labels distribution provided by the watermarked model and the clean model. It is important because, as the authors stated in their paper, knowledge of the trigger watermarked class can be exploited by an adversary to reduce the robustness of EWE. The results are displayed in Figure 64, where negative values indicate a higher number of labels provided by the clean model and positive values indicate a higher number of labels provided by the watermarked model. In the three problems, the largest positive value corresponds to the trigger target class.

In CIFAR10, the trigger target class is 0 (zero) and it shows a difference of 31.8%

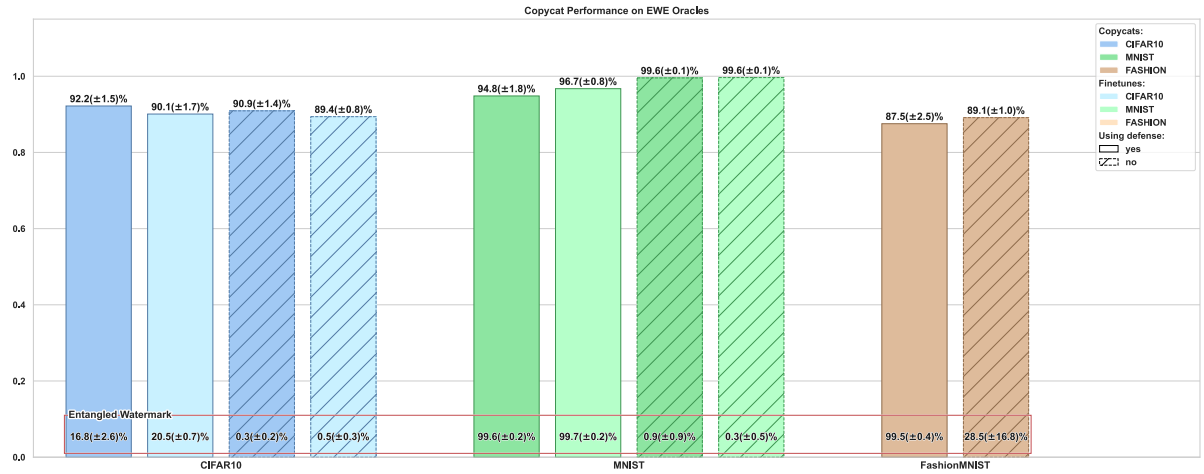



Figure 63 – Copycat results for EWE. The dark colors are the performance of the Copycat models and the light colors are the performance after fine-tuning with PDD dataset. The straight line as the edge is used in watermarked models and the dashed line is used in clean models. The red box shows the identified watermarked rate for watermarked models and the false positive rate for clean models.

in the number of labels when compared to the extracted labels from the clean model. The extracted labels from CIFAR10 watermarked model for this class comprise 44% of the NPDD dataset. Additionally, in MNIST, the trigger target class is 7 and it shows a difference of 96.5% in the number of labels when compared to the extracted labels from the clean model. The extracted labels from MNIST watermarked model for this class comprise 99% of the NPDD dataset. The number of labels in CIFAR10 and MNIST exhibited similar behavior, providing insight into the trigger target class.

In addition, the FashionMNIST trigger target class is 0 and it shows a difference of 27.3% in the number of labels when compared to the extracted labels from the clean model. However, the amount of labels extracted from the FashionMNIST watermark model for this class is not the greatest and it has 41.7% of the NPDD dataset. In contrast, the amount of extracted labels for the trigger source class comprise 55.5% of the NPDD dataset for the watermarked model, and 69.8% for the clean model. As can be seen, this problem did not show the same behavior as the other problems.

The authors of EWE carried out experiments to measure the effectiveness of their method and conducted an analysis to evaluate the capability of Neural Cleanse (WANG et al., 2019) in detecting EWE. This method tries to perturb data from each class of a dataset in order to misclassify them. Next, the class that was missclassified by significantly smaller perturbations is identified as the infected class. In their experiments, they concluded that Neural Cleanse is unable to identify their watermark. However, in our experiments on CIFAR10 and MNIST, we found disparities in label distribution that reveal the trigger target class. Thereby, we conducted experiments by decreasing the number of samples

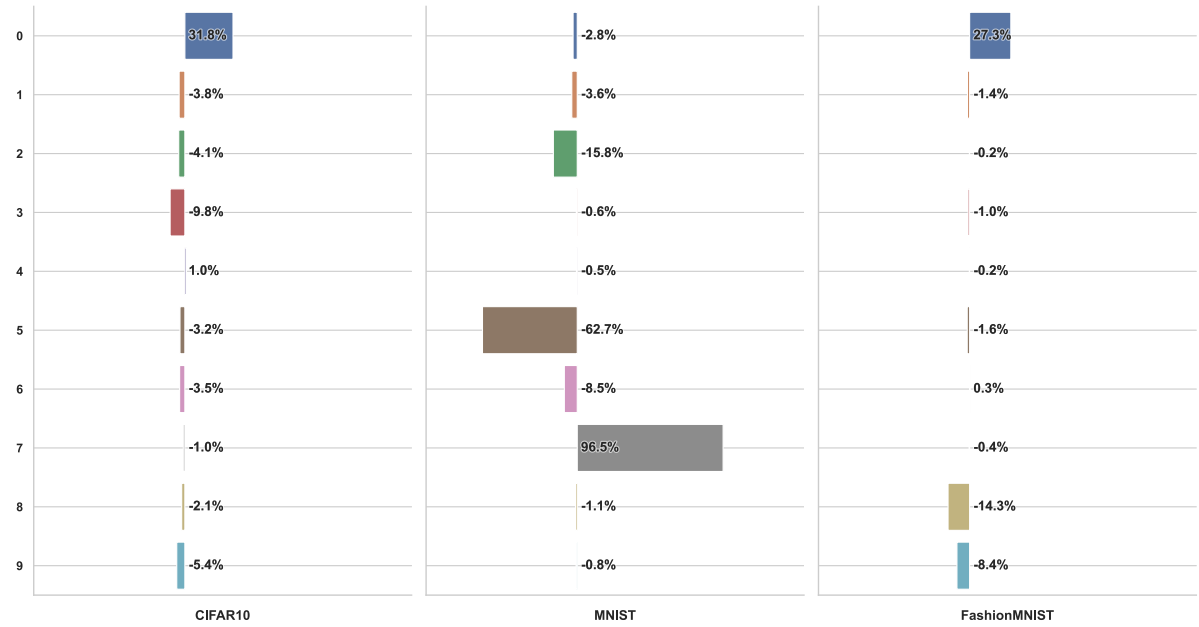



Figure 64 – EWE, difference of labels distribution between attacks on watermarked and clean models. This difference is the percentage of data from the dataset that had different labels when the system is protected. For each class, negative values represent more labels provided by the clean model and positive values represent more labels provided by the watermarked model.

in the trigger target class, but the watermark remained present in the Copycat model. Perhaps this results may be attributed to the fact that these watermarked models were trained on two different datasets and also to entangle the watermarks, *i. e.*, the images from other classes may also have features that can identify the watermark data. It did not provided a definitive conclusion, but it may suggest a research direction for detecting the trigger target class used in the EWE.

## 6.3 Limitations

One of the expected limitations is the performance limit when copying from a big architecture to a smaller one. All the other copy attack methods, as well as techniques such as model distillation, suffer from this limitation too. A limitation more specific to our proposed approach is that it needs samples with stolen labels for all classes to train a Copycat model, but one cannot ensure that the Oracle will cover all classes of interest when labeling queries with random natural images. In this work, however, it is shown that the proposed approach suggests to be successful in a wide range of problems, obtaining enough samples with stolen labels from all classes of interest for all the investigated problems by using natural random images. Nonetheless, we provide evidences that it is important that the images are not completely random (fully random pixels), for which it was not possible to obtain samples with stolen labels from all classes.

Additionally, another limitation of the Copycat method is against watermark defenses, such as EWE. This limitation is even important for companies that work with CNN models, as it is a way of protecting their models against the presented method. As shown in the experiments, the Copycat model retained the watermarks after the attack, allowing for the identification and claiming of company's IP. However, further tests must be conducted to identify the boundaries of this protection and if there are ways for the adversary to bypass it. Furthermore, other watermark defense methods should also be tested against the Copycat method.

# 7 Conclusion

This work presented a method to extract the knowledge of a black-box model using random natural images labeled by its maximum probabilities (hard-labels). Nowadays, many companies are providing cloud-based services powered by Convolutional Neural Networks or even smart robots and tools with this embedded technology. The growing ubiquity of these neural-based products highlights their increasing importance in society lives and underscores the need for reliable, secure, and robust systems that can be trusted to operate in a variety of problem domains. Our research was motivated by the relevance of this topic and also by the need of the literature for explore and better understand the vulnerabilities of these systems.

The academic community has already shown interest in investigating and better understanding the behavior of these models. As demonstrated in Section 3.1, some studies have revealed that state-of-the-art networks are vulnerable to attacks with adversarial examples and can even generate high-confidence predictions for images that are perceptually meaningless to humans. In addition, other research has presented techniques to extract knowledge from target models (also known as Oracles) through techniques such as model extraction, model compression and knowledge distillation. Therefore, our primary objective was to further investigate the intersection of these vulnerabilities. Specifically, we hypothesize that a model extraction could be performed using public random natural images (NPDD), which are available for everyone, and the Oracle's hard-labels, which any model must provide to its users. We also assumed that the surrogate model could be fine-tuned if the adversary owned a small problem domain dataset (PDD) and labeled it with the Oracle's hard-labels. We delimit our scope on image classification models with CNNs.

We confirmed our main hypothesis performing several experiments on model extraction attack using our Copycat method (Section 6.1) and also evaluating various defenses proposed in the literature against it (Section 6.2). In the first set of experiments presented in Subsection 6.1.2, the Copycat models achieved a performance greater than 96.7% on the proposed problems[1] and 96.4% performance on a real-world API (Subsection 6.1.7) using NPDDs. Moreover, when these models were fine-tuned by a PDD labeled by the Oracle, they needed fewer images from NPDD to achieve high copy performance (Subsection 6.1.3). In addition, the experiments presented in the Subsection 6.2.1 and Subsection 6.2.2 showed that some defense methods are not effective in detecting the attack performed by our method. On the other hand, experiments on Subsection 6.2.3 revealed that although

[1] Classification of facial expression, handwritten digits, house numbers in street view, human actions, objects, traffic signs, and pedestrian detection.

watermark defenses do not prevent model extraction attacks, they can protect the Oracle's Intellectual Property when attacked by the Copycat method.

In addition, the Copycat method achieved high performance in the experiments between different architectures (Subsection 6.1.4), robustness in the reproduction of the experiments (Subsection 6.1.5) and similarity between the Oracle and Copycat models in the quantitative and qualitative analyses (Subsections 6.1.1 and 6.1.6). And Section 6.3 shows that the main limitation of our method is related to the lack of guarantee of getting all the necessary labels in new problems that have not yet been tested.

These results confirmed that our hypothesis is valid and effective in several problems, *i. e.*, the Copycat method can achieve a high performance on attacks without requiring data from the problem domain, or the model probabilities (soft-labels), or logits. Thus, our results alert companies about the existing vulnerabilities in CNN models, which can generate significant financial losses. At the same time, these results allow companies and academia to understand the security risks associated with model extraction attacks. Research carried out in this area can help improve the security of machine learning models, which is crucial as these models are increasingly being used in many domains of society.

And the future directions of research are:

- Explore additional defense methods against model extraction and evaluate the robustness of the Copycat method against these defenses.
- Extend the method to other types of problems beyond image classification.
- Investigate the behavior of more complex datasets to further understand the applicability of the Copycat method. Given that Copycat method uses model hard-labels and NPDD, it can be challenging to extract models trained with more complex datasets, especially when it is difficult to obtain enough images to extract labels for all desired classes.
- Investigate the use of adversarial examples to extract a model without watermarked information and assess the effectiveness of this approach. The methods used in adversarial examples exploit image pixels modifications to cause Oracle misclassification. But in model extraction, this exploration could be done to generate images with minimal traces of the watermark. Another possibility is to generate modifications on the already labeled images and subsequently extract the surrogate model to a new one, thereby eliminating some watermarked traces.
- Explore the possibility of copying the style of a PDD model to an NPDD model using style-transfer techniques and evaluate the performance of the Copycat method with this enhanced attack. Problem domain images proved to be beneficial for the model extraction in our experiments. This suggests that using images that are similar to the training set can contribute to obtaining more meaningful labels during the extraction process. As a result, employing style transfer techniques from

a small problem domain dataset to thousands of NPDD images could allow the extraction of models with fewer images.

# Bibliography

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Available from Internet: <https://www.tensorflow.org/>.

ADDEPALLI, S.; NAYAK, G. K.; CHAKRABORTY, A.; RADHAKRISHNAN, V. B. Degan: Data-enriching GAN for retrieving representative samples from a trained classifier. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. New York, NY, USA: AAAI Press, 2020. p. 3130–3137. Available from Internet: <https://ojs.aaai.org/index.php/AAAI/article/view/5709>.

AMRIT, P.; SINGH, A. K. Survey on watermarking methods in the artificial intelligence domain and beyond. *Computer Communications*, v. 188, p. 52–65, apr 2022. ISSN 01403664. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S0140366422000664>.

ANDERS, C. J.; NEUMANN, D.; SAMEK, W.; MüLLER, K.-R.; LAPUSCHKIN, S. Software for dataset-wide xai: From local explanations to global insights with Zennit, CoRelAy, and ViRelAy. *CoRR*, abs/2106.13200, 2021.

ANOWAR, F.; SADAOUI, S.; SELIM, B. Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE). *Computer Science Review*, v. 40, p. 100378, may 2021. ISSN 15740137. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S1574013721000186>.

ARRAS, L.; OSMAN, A.; SAMEK, W. CLEVR-XAI: A benchmark dataset for the ground truth evaluation of neural network explanations. *Information Fusion*, v. 81, p. 14–40, may 2022. ISSN 15662535. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S1566253521002335>.

BA, J.; CARUANA, R. Do deep nets really need to be deep? In: GHAHRAMANI, Z.; WELLING, M.; CORTES, C.; LAWRENCE, N.; WEINBERGER, K. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014. v. 27, p. 2654–2662. Available from Internet: <https://proceedings.neurips.cc/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf>.

BACH, S.; BINDER, A.; MONTAVON, G.; KLAUSCHEN, F.; MüLLER, K.-R.; SAMEK, W. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE*, v. 10, n. 7, p. e0130140, jul 2015. ISSN 1932-6203. Available from Internet: <https://dx.plos.org/10.1371/journal.pone.0130140>.

BEETHAM, J.; KARDAN, N.; MIAN, A. S.; SHAH, M. Dual student networks for data-free model stealing. In: *The Eleventh International Conference on Learning Representations (ICLR)*. [s.n.], 2023. Available from Internet: <https://openreview.net/forum?id=VE1s3e5xriA>.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: a review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, v. 35, n. 8, p. 1798—1828, August 2013. ISSN 0162-8828. Available from Internet: <https://doi.org/10.1109/TPAMI.2013.50>.

BUCILǍ, C.; CARUANA, R.; NICULESCU-MIZIL, A. Model Compression. In: *Proceedings of the 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*. [S.l.]: Association for Computing Machinery, 2006. p. 535—541.

CHAN, W.; KE, N. R.; LANE, I. Transferring knowledge from a RNN to a DNN. In: *Interspeech 2015*. ISCA, 2015. p. 3264–3268. Available from Internet: <https://www.isca-speech.org/archive/interspeech_2015/chan15_interspeech.html>.

CLANUWAT, T.; BOBER-IRIZAR, M.; KITAMOTO, A.; LAMB, A.; YAMAMOTO, K.; HA, D. Deep learning for classical japanese literature. *CoRR*, abs/1812.01718, 2018. Available from Internet: <http://arxiv.org/abs/1812.01718>.

COATES, A.; NG, A.; LEE, H. An analysis of single-layer networks in unsupervised feature learning. In: *Proceedings of the Fourteenth Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*. [S.l.: s.n.], 2011. p. 215–223.

Correia-Silva, J. R.; Berriel, R. F.; Badue, C.; De Souza, A. F.; Oliveira-Santos, T. Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data. In: *International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–8.

Correia-Silva, J. R.; Berriel, R. F.; Badue, C.; De Souza, A. F.; Oliveira-Santos, T. Copycat CNN: Are random non-Labeled data enough to steal knowledge from black-box models? *Pattern Recognition*, v. 113, p. 107830, 2021. ISSN 0031-3203.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 248–255.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Available from Internet: <http://archive.ics.uci.edu/ml>.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2011. ISBN 9788521618805.

FELLBAUM, C. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998. ISBN 9780262272551. Available from Internet: <https://doi.org/10.7551/mitpress/7287.001.0001>.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, p. 193–202, 1980.

GAROFOLO, J. S.; LAMEL, L. F.; FISHER, W. M.; PALLETT, D. S.; DAHLGREN, N. L.; ZUE, V.; FISCUS, J. G. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Linguistic Data Consortium, 1993. Artwork Size: 715776 KB Pages: 715776 KB Type: dataset. Available from Internet: <https://catalog.ldc.upenn.edu/LDC93S1>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial nets. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 2672–2680.

GOODFELLOW, I.; SHLENS, J.; SZEGEDY, C. Explaining and harnessing adversarial examples. In: *International Conference on Learning Representations (ICLR)*. [S.l.: s.n.], 2015.

GROTHER, P. J.; FLANAGAN, P. A. *NIST Handprinted Forms and Characters, NIST Special Database 19*. National Institute of Standards and Technology, 1995. Available from Internet: <http://www.nist.gov/srd/nistsd19.cfm>.

GU, J.; WANG, Z.; KUEN, J.; MA, L.; SHAHROUDY, A.; SHUAI, B.; LIU, T.; WANG, X.; WANG, G.; CAI, J.; CHEN, T. Recent advances in convolutional neural networks. *Pattern Recognition*, Elsevier BV, v. 77, p. 354–377, may 2018. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0031320317304120>.

GUNNING, D.; AHA, D. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine*, v. 40, n. 2, p. 44–58, jun 2019. ISSN 2371-9621, 0738-4602. Available from Internet: <https://ojs.aaai.org/index.php/aimagazine/article/view/2850>.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. New York, NY: Springer New York, 2009. (Springer Series in Statistics). ISBN 978-0-387-84857-0 978-0-387-84858-7. Available from Internet: <http://link.springer.com/10.1007/978-0-387-84858-7>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.]: IEEE Computer Society, 2016. p. 770–778.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Identity mappings in deep residual networks. In: LEIBE, B.; MATAS, J.; SEBE, N.; WELLING, M. (Ed.). *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016. p. 630–645. ISBN 978-3-319-46493-0.

HENDRYCKS, D.; MAZEIKA, M.; DIETTERICH, T. G. Deep anomaly detection with outlier exposure. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. Available from Internet: <https://openreview.net/forum?id=HyxCxhRcY7>.

HINTON, G.; VINYALS, O.; DEAN, J. Distilling the Knowledge in a Neural Network. In: *NeurIPS Deep Learn. Represent. Learn. Workshop*. [S.l.: s.n.], 2015.

HOLZINGER, A.; SARANTI, A.; MOLNAR, C.; BIECEK, P.; SAMEK, W. Explainable AI Methods - A Brief Overview. In: HOLZINGER, A.; GOEBEL, R.; FONG, R.; MOON, T.; MüLLER, K.-R.; SAMEK, W. (Ed.). *xxAI - Beyond Explainable AI*. Cham:

Springer International Publishing, 2022. v. 13200, p. 13–38. ISBN 978-3-031-04082-5 978-3-031-04083-2. Series Title: Lecture Notes in Computer Science. Available from Internet: <https://link.springer.com/10.1007/978-3-031-04083-2_2>.

HUANG, L.-L.; YIN, F. Traffic Sign Recognition Using Perturbation Method. In: *Chin. Conf. Pattern Recognit. (CCPR)*. [S.l.]: Springer, 2014. p. 518–527.

HUBEL, D. H.; WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, v. 195, p. 215–243, 1968.

HULL, J. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 16, n. 5, p. 550–554, 1994.

JAGIELSKI, M.; CARLINI, N.; BERTHELOT, D.; KURAKIN, A.; PAPERNOT, N. High accuracy and high fidelity extraction of neural networks. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. USA: USENIX Association, 2020. (SEC'20). ISBN 978-1-939133-17-5.

JIA, H.; CHOQUETTE-CHOO, C. A.; PAPERNOT, N. Entangled watermarks as a defense against model extraction. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021. ISBN 978-1-939133-17-5. ArXiv: 2002.12200. Available from Internet: <http://arxiv.org/abs/2002.12200>.

JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUADARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In: *Proceedings of the 22nd ACM International Conference on Multimedia*. New York, NY, USA: Association for Computing Machinery, 2014. (MM '14), p. 675–678. ISBN 9781450330633. Available from Internet: <https://doi.org/10.1145/2647868.2654889>.

JUUTI, M.; SZYLLER, S.; MARCHAL, S.; ASOKAN, N. PRADA: Protecting Against DNN Model Stealing Attacks. In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. [S.l.: s.n.], 2019. p. 512–527.

KARIYAPPA, S.; QURESHI, M. K. Defending Against Model Stealing Attacks With Adaptive Misinformation. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, 2020. p. 767–775. ISBN 978-1-72817-168-5. Available from Internet: <https://ieeexplore.ieee.org/document/9157021/>.

KHAN, A.; SOHAIL, A.; ZAHOORA, U.; QURESHI, A. S. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, v. 53, n. 8, p. 5455–5516, dec 2020. ISSN 0269-2821, 1573-7462. Available from Internet: <https://link.springer.com/10.1007/s10462-020-09825-6>.

KOHLBRENNER, M.; BAUER, A.; NAKAJIMA, S.; BINDER, A.; SAMEK, W.; LAPUSCHKIN, S. Towards Best Practice in Explaining Neural Network Decisions with LRP. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, UK: IEEE, 2020. p. 1–7. ISBN 978-1-72816-926-2. Available from Internet: <https://ieeexplore.ieee.org/document/9206975/>.

KORNBLITH, S.; NOROUZI, M.; LEE, H.; HINTON, G. E. Similarity of neural network representations revisited. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. PMLR, 2019. (Proceedings

of Machine Learning Research, v. 97), p. 3519–3529. Available from Internet: <http://proceedings.mlr.press/v97/kornblith19a.html>.

KRIZHEVSKY, A.; HINTON, G. *Learning multiple layers of features from Tiny Images*. Dissertação (Master's thesis) — University of Toronto, Department of Computer Science, Toronto, Canada, 2009.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012. v. 25, p. 1097–1105. Available from Internet: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

KUZNETSOVA, A.; ROM, H.; ALLDRIN, N.; UIJLINGS, J.; KRASIN, I.; PONT-TUSET, J.; KAMALI, S.; POPOV, S.; MALLOCI, M.; KOLESNIKOV, A.; DUERIG, T.; FERRARI, V. The Open Images Dataset V4. *International Journal of Computer Vision (IJCV)*, v. 128, p. 1956–1981, 2020.

LANGNER, O.; DOTSCH, R.; BIJLSTRA, G.; WIGBOLDUS, D. H.; HAWK, S. T.; KNIPPENBERG, A. V. Presentation and validation of the Radboud Faces Database. *Cognition and Emotion*, v. 24, n. 8, p. 1377–1388, 2010.

LAPUSCHKIN, S.; WäLDCHEN, S.; BINDER, A.; MONTAVON, G.; SAMEK, W.; MüLLER, K.-R. Unmasking Clever Hans predictors and assessing what machines really learn. *Nature Communications*, v. 10, n. 1, p. 1096, mar 2019. ISSN 2041-1723. Available from Internet: <https://www.nature.com/articles/s41467-019-08987-4>.

LECUN, Y. Generalization and network design strategies. In: PFEIFER, R.; SCHRETER, Z.; FOGELMAN, F.; STEELS, L. (Ed.). *Connectionism in Perspective*. Zurich, Switzerland: Elsevier, 1989.

LECUN, Y.; BENGIO, Y. Convolutional networks for images, speech, and time-series. In: ARBIB, M. A. (Ed.). *The Handbook of Brain Theory and Neural Networks*. [S.l.]: MIT Press, 1995.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, MIT Press - Journals, v. 1, n. 4, p. 541–551, dec 1989. Available from Internet: <https://doi.org/10.1162%2Fneco.1989.1.4.541>.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Handwritten digit recognition with a back-propagation network. In: *Proceedings of the 2nd International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1989. (NIPS'89), p. 396–404.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, Nov 1998. ISSN 0018-9219.

LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; JACKEL, L. D.; BOTTOU, L.; BRUNOT, A.; CORTES, C.; DENKER,

J. S.; DRUCKER, H.; GUYON, I.; MULLER, U. A.; SACKINGER, E.; SIMARD, P.; VAPNIK, V. Comparison of learning algorithms for handwritten digit recognition. In: FOGELMAN, F.; GALLINARI, P. (Ed.). *International Conference on Artificial Neural Networks*. Paris: EC2 & Cie, 1995. p. 53–60.

LEE, T.; EDWARDS, B.; MOLLOY, I.; SU, D. Defending against neural network model stealing attacks using deceptive perturbations. In: *2019 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA, USA: IEEE, 2019. p. 43–49. ISBN 978-1-72813-508-3. Available from Internet: <https://ieeexplore.ieee.org/document/8844598/>.

LI, Y.; WANG, H.; BARNI, M. A survey of Deep Neural Network watermarking techniques. *Neurocomputing*, v. 461, p. 171–193, oct 2021. ISSN 09252312. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S092523122101095X>.

LIN, M.; CHEN, Q.; YAN, S. Network in network. In: BENGIO, Y.; LECUN, Y. (Ed.). *2nd International Conference on Learning Representations, ICLR*. [s.n.], 2014. Available from Internet: <http://arxiv.org/abs/1312.4400>.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: FLEET, D.; PAJDLA, T.; SCHIELE, B.; TUYTELAARS, T. (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 740–755. ISBN 978-3-319-10602-1.

LU, Y.; LU, J.; ZHANG, S.; HALL, P. Traffic signal detection and classification in street views using an attention model. *Computational Visual Media*, v. 4, n. 3, p. 253–266, 2018.

LUCEY, P.; COHN, J. F.; KANADE, T.; SARAGIH, J.; AMBADAR, Z.; MATTHEWS, I. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. [S.l.: s.n.], 2010. p. 94–101.

LUNDQVIST, D.; FLYKT, A.; ÖHMAN, A. The Karolinska Directed Emotional Faces (KDEF). *CD ROM from Department of Clinical Neuroscience, Psychology section, Karolinska Institutet*, 1998.

LYONS, M.; AKAMATSU, S.; KAMACHI, M.; GYOBA, J. Coding facial expressions with Gabor wavelets. In: *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition (FG)*. [S.l.: s.n.], 1998. p. 200–205.

MAATEN, L. van der; HINTON, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, v. 9, n. 86, p. 2579–2605, 2008. Available from Internet: <http://jmlr.org/papers/v9/vandermaaten08a.html>.

MARTINEZ, A. M.; BENAVENTE, R. The AR face database. *CVC Technical Report #24*, 06 1998.

MONTAVON, G.; BINDER, A.; LAPUSCHKIN, S.; SAMEK, W.; MüLLER, K.-R. Layer-Wise Relevance Propagation: An Overview. In: SAMEK, W.; MONTAVON, G.; VEDALDI, A.; HANSEN, L. K.; MüLLER, K.-R. (Ed.). *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Cham: Springer International Publishing, 2019. v. 11700, p. 193–209. ISBN 978-3-030-28953-9 978-3-030-28954-6. LRP Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-030-28954-6_10>.

MONTAVON, G.; LAPUSCHKIN, S.; BINDER, A.; SAMEK, W.; MüLLER, K.-R. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, v. 65, p. 211–222, may 2017. ISSN 00313203. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S0031320316303582>.

MOSAFI, I.; DAVID, E. O.; NETANYAHU, N. S. Deepmimic: Mentor-student unlabeled data based training. In: TETKO, I. V.; KŮRKOVÁ, V.; KARPOV, P.; THEIS, F. (Ed.). *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*. Cham: Springer International Publishing, 2019. p. 440–455. ISBN 978-3-030-30493-5.

MOSAFI, I.; DAVID, E. O.; NETANYAHU, N. S. Stealing knowledge from protected deep neural networks using composite unlabeled data. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2019. p. 1–8. ISSN 2161-4393.

MUNDER, S.; GAVRILA, D. M. An Experimental Study on Pedestrian Classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 28, n. 11, p. 1863–1868, 2006.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Madison, WI, USA: Omnipress, 2010. (ICML'10), p. 807–814. ISBN 9781605589077.

NETZER, Y.; WANG, T.; COATES, A.; BISSACCO, A.; WU, B.; NG, A. Y. Reading Digits in Natural Images with Unsupervised Feature Learning. In: *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*. [S.l.: s.n.], 2011.

NGUYEN, A.; YOSINSKI, J.; CLUNE, J. Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 427–436.

NILSBACK, M.-E.; ZISSERMAN, A. A visual vocabulary for flower classification. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.: s.n.], 2006. v. 2, p. 1447–1454.

OREKONDY, T.; SCHIELE, B.; FRITZ, M. Knockoff Nets: Stealing Functionality of Black-Box Models. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2019. p. 4949–4958.

OREKONDY, T.; SCHIELE, B.; FRITZ, M. Prediction poisoning: Towards defenses against dnn model stealing attacks. In: *International Conference on Learning Representations*. [s.n.], 2020. Available from Internet: <https://openreview.net/forum?id=SyevYxHtDB>.

PANTIC, M.; VALSTAR, M.; RADEMAKER, R.; MAAT, L. Web-based database for facial expression analysis. In: *International Conference on Multimedia and Expo (ICME)*. [S.l.: s.n.], 2005.

PAPERNOT, N.; MCDANIEL, P.; GOODFELLOW, I.; JHA, S.; CELIK, Z. B.; SWAMI, A. Practical Black-Box Attacks Against Machine Learning. In: *Asia Conf. Comput. Commun. Secur. (ASIACCS)*. [S.l.]: ACM, 2017. p. 506—519.

PAPERNOT, N.; MCDANIEL, P.; JHA, S.; FREDRIKSON, M.; CELIK, Z. B.; SWAMI, A. The limitations of deep learning in adversarial settings. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. [S.l.: s.n.], 2016. p. 372–387.

PAPERNOT, N.; MCDANIEL, P. D.; GOODFELLOW, I. J. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv:1605.07277*, 2016.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. p. 8024–8035. Available from Internet: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

PENG, S.; CHEN, Y.; XU, J.; CHEN, Z.; WANG, C.; JIA, X. Intellectual property protection of DNN models. *World Wide Web*, nov 2022. ISSN 1386-145X, 1573-1413. Available from Internet: <https://link.springer.com/10.1007/s11280-022-01113-3>.

QUATTONI, A.; TORRALBA, A. Recognizing indoor scenes. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 413–420.

RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; SUTSKEVER, I. Language models are unsupervised multitask learners. *OpenAI blog*, v. 1, n. 8, p. 9, 2019. Available from Internet: <https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf>.

REBUFFI, S.-A.; BILEN, H.; VEDALDI, A. Learning multiple visual domains with residual adapters. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [S.l.]: Curran Associates, Inc., 2017. p. 506–516.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015.

SANYAL, S.; ADDEPALLI, S.; BABU, R. V. Towards data-free model stealing in a hard label setting. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, 2022. p. 15263–15272. ISBN 978-1-66546-946-3. Available from Internet: <https://ieeexplore.ieee.org/document/9880326/>.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, [Oxford University Press, Biometrika Trust], v. 52, n. 3/4, p. 591–611, 1965. ISSN 00063444. Available from Internet: <http://www.jstor.org/stable/2333709>.

SHI, Y.; SAGDUYU, Y.; GRUSHIN, A. How to steal a machine learning classifier with deep learning. In: *IEEE International Symposium on Technologies for Homeland Security (HST)*. [S.l.: s.n.], 2017. p. 1–5.

SIMONYAN, K.; VEDALDI, A.; ZISSERMAN, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. In: BENGIO, Y.; LECUN, Y. (Ed.). *2nd International Conference on Learning Representations (ICLR)*. [s.n.], 2014. Available from Internet: <http://arxiv.org/abs/1312.6034>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. In: BENGIO, Y.; LECUN, Y. (Ed.). *International Conference on Learning Representations ICLR*. San Diego, CA, USA: [s.n.], 2015. Available from Internet: <http://arxiv.org/abs/1409.1556>.

SOOMRO, K.; ZAMIR, A. R.; SHAH, M. *UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild*. [S.l.], 2012.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, JMLR.org, v. 15, n. 1, p. 1929–1958, jan 2014. ISSN 1532-4435.

STALLKAMP, J.; SCHLIPSING, M.; SALMEN, J.; IGEL, C. The german traffic sign recognition benchmark: A multi-class classification competition. In: *The 2011 International Joint Conference on Neural Networks*. [S.l.: s.n.], 2011. p. 1453–1460.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015. Available from Internet: <http://arxiv.org/abs/1409.4842>.

SZEGEDY, C.; ZAREMBA, W.; SUTSKEVER, I.; BRUNA, J.; ERHAN, D.; GOODFELLOW, I.; FERGUS, R. Intriguing properties of neural networks. In: *International Conference on Learning Representations (ICLR)*. [S.l.: s.n.], 2014.

TANG, Z.; WANG, D.; ZHANG, Z. Recurrent Neural Network Training with Dark Knowledge Transfer. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2016. p. 5900–5904.

TIMOFTE, R.; ZIMMERMANN, K.; GOOL, L. V. Multi-view traffic sign detection, recognition, and 3d localisation. In: *2009 Workshop on Applications of Computer Vision (WACV)*. [S.l.: s.n.], 2009. p. 1–8.

TRAMÈR, F.; ZHANG, F.; JUELS, A.; REITER, M. K.; RISTENPART, T. Stealing machine learning models via prediction apis. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. [S.l.: s.n.], 2016. (SEC'16), p. 601—618.

WANG, B.; YAO, Y.; SHAN, S.; LI, H.; VISWANATH, B.; ZHENG, H.; ZHAO, B. Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: *2019 IEEE Symposium on Security and Privacy (SP)*. [S.l.: s.n.], 2019. p. 707–723.

WARDEN, P. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, abs/1804.03209, 2018. Available from Internet: <http://arxiv.org/abs/1804.03209>.

XIAO, H.; RASUL, K.; VOLLGRAF, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

XUE, M.; ZHANG, Y.; WANG, J.; LIU, W. Intellectual property protection for deep

learning models: Taxonomy, methods, attacks, and evaluations. *IEEE Transactions on Artificial Intelligence*, v. 3, n. 6, p. 908–923, dec 2022. ISSN 2691-4581. Available from Internet: <https://ieeexplore.ieee.org/document/9645219/>.

YIN, L.; WEI, X.; SUN, Y.; WANG, J.; ROSATO, M. J. A 3D facial expression database for facial behavior research. In: *International Conference on Automatic Face and Resture Recognition (FGR)*. [S.l.: s.n.], 2006. p. 211–216.

ZAVAREZ, M. V.; BERRIEL, R. F.; OLIVEIRA-SANTOS, T. Cross-Database Facial Expression Recognition Based on Fine-Tuned Deep Convolutional Network. In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. [S.l.: s.n.], 2017. p. 405–412.

ZHANG, A.; LIPTON, Z. C.; LI, M.; SMOLA, A. J. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. Available from Internet: <https://d2l.ai>.

ZHANG, G.; LIU, B.; ZHU, T.; ZHOU, A.; ZHOU, W. Visual privacy attacks and defenses in deep learning: a survey. *Artificial Intelligence Review*, jan 2022. ISSN 0269-2821, 1573-7462. Available from Internet: <https://link.springer.com/10.1007/s10462-021-10123-y>.

# Appendix

# APPENDIX A – Code for running the simple example of the Copycat Method

The code is bellow, but you can download it on: <https://github.com/jeiks/Stealing_DL_Models/tree/master/Seeing_Copycat_Example>. Note that before running this script, you must verify that the following Python dependencies are installed on the system or virtual environment: torch, numpy, matplotlib, and Pillow.

And we also provide a simple and practical example with a shallow CNN model that can be trained on CIFAR10 dataset. It is available at: <https://github.com/jeiks/Stealing_DL_Models/tree/master/Example_of_use>.

```
import torch
from torch import nn

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.conv = nn.Conv2d(1, 1,
                              kernel_size=(2, 1),
                              stride=(1, 1),
                              bias=False)
        self.linear = nn.Linear(2, 1, bias=False)

    def forward(self, x):
        x = self.conv(x)
        x = self.linear(x)
        return x

    def get_feature_map(self, x):
        return self.conv(x)

    def _print_list(self, l, d=8):
        for x in l.squeeze().tolist():
            print(f'{x:{d+3}.{d}f}', end=' ')

    def _print(self, msg, w, decimals=8):
        if w is not None:
            print(f'  {msg}: [', end=' ')
            self._print_list(w, d=decimals)
            print(']')

    def print_parameters(self):
        self._print('Conv Kernel', self.conv.weight)
```

```
37          self._print(' {Conv Grad', self.conv.weight.grad)
38          self._print('Lin Weights', self.linear.weight)
39          self._print(' {Lin  Grad', self.linear.weight.grad)
40          print('-'*40)
41
42 def tensor_to_img(img):
43     return (img*255).squeeze().detach().numpy().astype(np.uint8)
44
45 def plt_images(X, save=False, rows=2, title=None):
46     N = X.shape[0]
47     fig, ax = plt.subplots(rows, N//rows)
48     if title is not None: fig.suptitle(title)
49     for idx,img in enumerate(X):
50         row = idx // (N // rows)
51         col = idx % (N // rows)
52         img = tensor_to_img(img)
53         ax[row,col].imshow(img, cmap='gray', vmin=0, vmax=255)
54         #ax[row,col].set_axis_off()
55         ax[row,col].set_xticks([])
56         ax[row,col].set_yticks([])
57         #print(img)
58     plt.tight_layout()
59     if save:
60         plt.savefig('figures-simple-cnn.svg')
61     else:
62         plt.show()
63
64 def save_images(X, size=(100,100)):
65     for idx,img in enumerate(X):
66         img = tensor_to_img(img)
67         Image.fromarray(img).resize(size, 0).save(f'image-{idx:03d}.png')
68
69 def train_model(net, X, y, max_epochs, lr):
70     loss_fn = torch.nn.MSELoss()
71     optimizer = torch.optim.SGD(net.parameters(), lr=lr)
72     net.train(True)
73     for epoch in range(max_epochs):
74         optimizer.zero_grad()
75         outputs = net(X)
76         loss = loss_fn(outputs, y)
77         loss.backward(retain_graph=True)
78         optimizer.step()
79         print(f'Epoch: {epoch+1:03d}')
80         outputs = torch.sign(outputs.detach())
81         print(f'       Outputs: ', end='')
82         print([x.item() for x in outputs])
83         print(f'          Loss: {loss.item():.8f}')
84         net.print_parameters()
85         if loss.item() < 0.01: break
86     net.eval()
87
88     return net
89
90 def plt_feature_map(net, X, y, marker='o', colors=['g','b']):
91     for p, t in zip(X, y):
92         group = int(t.squeeze().item())
93         c = 'r' if group==-1 else 'b'
```

```
94          point = net.get_feature_map(p.unsqueeze(0))
95          point = point.squeeze().detach().numpy()
96          plt.plot(point[0], point[1], marker=marker, c=c)
97
98  def plt_classification_space(net):
99      w = net.linear.weight.squeeze().detach().numpy()
100     w = w / np.linalg.norm(w)
101     # ortogonal:
102     v = np.array([-w[1],w[0]])
103
104     origin = np.array([[0, 0, 0],[0, 0, 0]])
105     #print(f'weights: {w}')
106     plt.quiver(*origin, w[0], w[1], color=['g'],
107                angles='xy', scale_units='xy',
108                scale=1, label='Weights', width=0.004)
109     # plot dimensions
110     aux = np.linspace(min(plt.xlim()[0], plt.ylim()[0],w[0],w[1])-0.2,
111                       max(plt.xlim()[1], plt.ylim()[1],w[0],w[1])+0.2, 2)
112     plt.xlim(aux)
113     plt.ylim(aux)
114     plt.plot(aux, aux*(v[1]/v[0]))
115
116 def get_inputs_2points():
117     X = [  # 1st:
118            [[ [1.0 , 0.0 ],
119               [0.0 , 1.0 ] ]],
120            # 2nd:
121            [[ [0.0 , 1.0 ],
122               [1.0 , 0.0 ] ]]  ]
123     y = [  [[[-1.]]],   #1st
124            [[[ 1.]]]  ] #2nd
125     return (X,y)
126
127 def get_inpus_8points():
128     X = [  # 1st group:
129            [[ [1.0 , 0.0 ],
130               [0.0 , 1.0 ] ]],
131            [[ [0.75, 0.0 ],
132               [0.0 , 0.75] ]],
133            [[ [0.5 , 0.0 ],
134               [0.0 , 0.5 ] ]],
135            [[ [0.25, 0.0 ],
136               [0.0 , 0.25] ]],
137            # 2nd group:
138            [[ [0.0 , 1.0 ],
139               [1.0 , 0.0 ] ]],
140            [[ [0.0 , 0.75],
141               [0.75, 0.0 ] ]],
142            [[ [0.0 , 0.5 ],
143               [0.5 , 0.0 ] ]],
144            [[ [0.0 , 0.25],
145               [0.25, 0.0 ] ]]  ]
146     y = [  [[[-1.]]], [[[-1.]]], [[[-1.]]], [[[-1.]]],   #1st
147            [[[ 1.]]], [[[ 1.]]], [[[ 1.]]], [[[ 1.]]]  ] #2nd
148     return (X,y)
149
150 if __name__ == '__main__':
```

```
151     max_epochs = 100
152     lr = 0.1
153     uniform_distrib = True
154     # uniform or normal distribution:
155     rand = torch.rand if uniform_distrib else torch.randn
156     plt.ion()
157 
158     # Getting points to train the Oracle:
159     X,y = get_inpus_8points()
160     X = torch.tensor(X, requires_grad=True)
161     y = torch.tensor(y, requires_grad=True)
162     # Generating image attack
163     attack_x = rand(3*X.shape[0], 1, 2, 2)
164 
165     # Plotting images:
166     plt_images(X, rows=2, title='Training dataset')
167     plt_images(attack_x, rows=4, title='Attack dataset')
168 
169     # Training the Oracle:
170     oracle = SimpleNet()
171     oracle.print_parameters()
172     oracle = train_model(oracle, X, y, max_epochs, lr)
173 
174     # ATTACK: Getting labels from the oracle
175     attack_y = torch.sign(oracle(attack_x))
176 
177     # Plotting results:
178     plt.figure(figsize=(8,8))
179     plt.title('Oracle')
180     plt_feature_map(oracle, X, y, marker='o')
181     plt_classification_space(oracle)
182     plt_feature_map(oracle, attack_x, attack_y, marker='x')
183 
184     # Training the Copycat
185     copycat = SimpleNet()
186     copycat.print_parameters()
187     copycat = train_model(copycat, attack_x, attack_y, max_epochs, lr)
188 
189     plt.figure(figsize=(8,8))
190     plt.title('Copycat')
191     plt_feature_map(copycat, X, y, marker='o')
192     plt_classification_space(copycat)
193     plt_feature_map(copycat, attack_x, attack_y, marker='x')
194 
195     input('Press ENTER to finish...')
```

# APPENDIX B – LRP Heatmaps of TDD

This appendix presents Examples of PDD heatmaps generated with LRP on Copycat Framework. Twenty random heatmaps of TDD are presented for each problem.

More heatmaps can be found at:

<https://jeiks.github.io/Stealing_DL_Models/framework-heatmaps/pdd>.


Figure 65 – Examples of PDD heatmaps generated with LRP on Copycat Framework for ACT101. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.

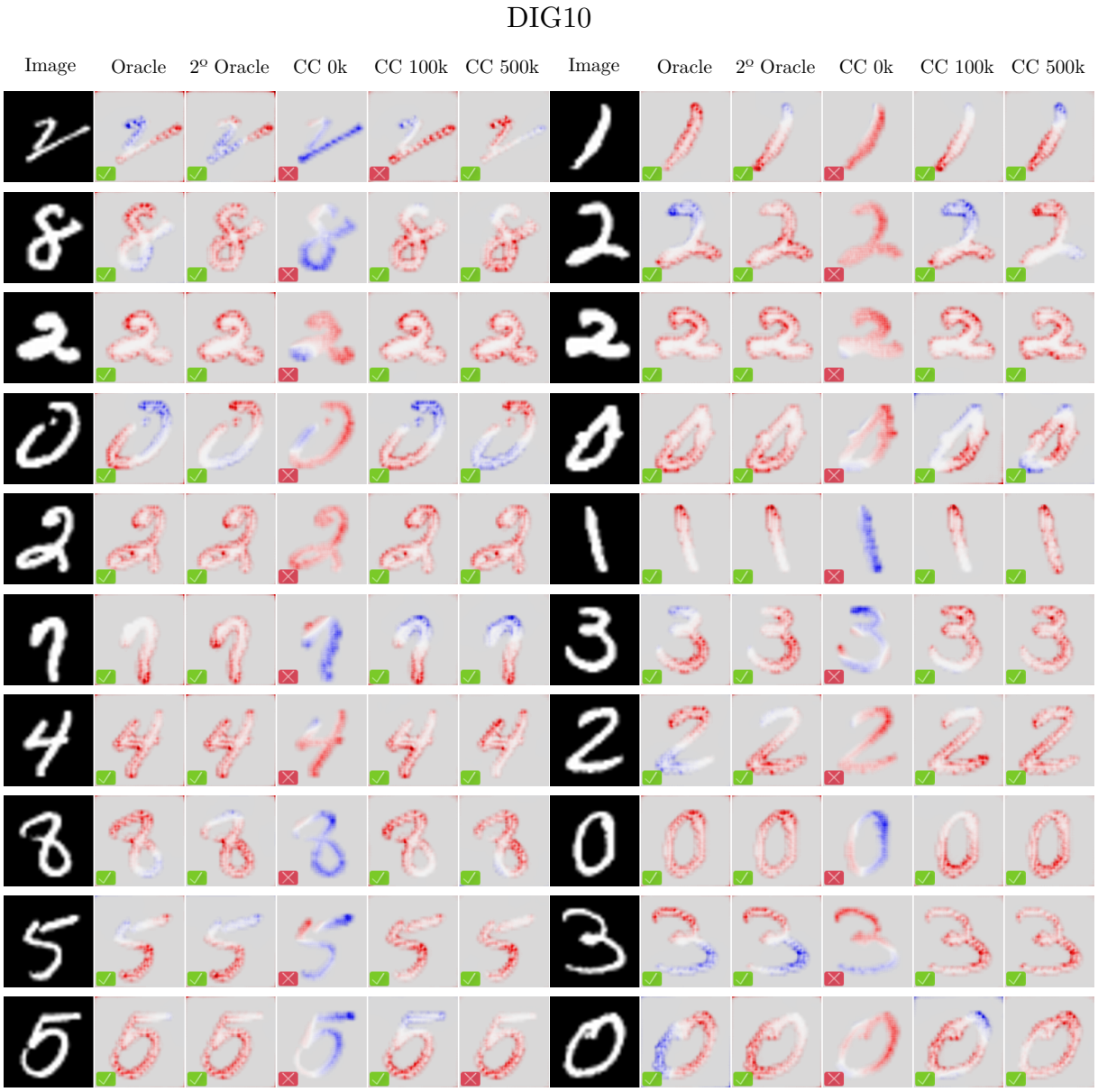



Figure 66 – Examples of PDD heatmaps generated with LRP on Copycat Framework for DIG10. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.


Figure 67 – Examples of PDD heatmaps generated with LRP on Copycat Framework for FER7. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.


Figure 68 – Examples of PDD heatmaps generated with LRP on Copycat Framework for GOC9. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.

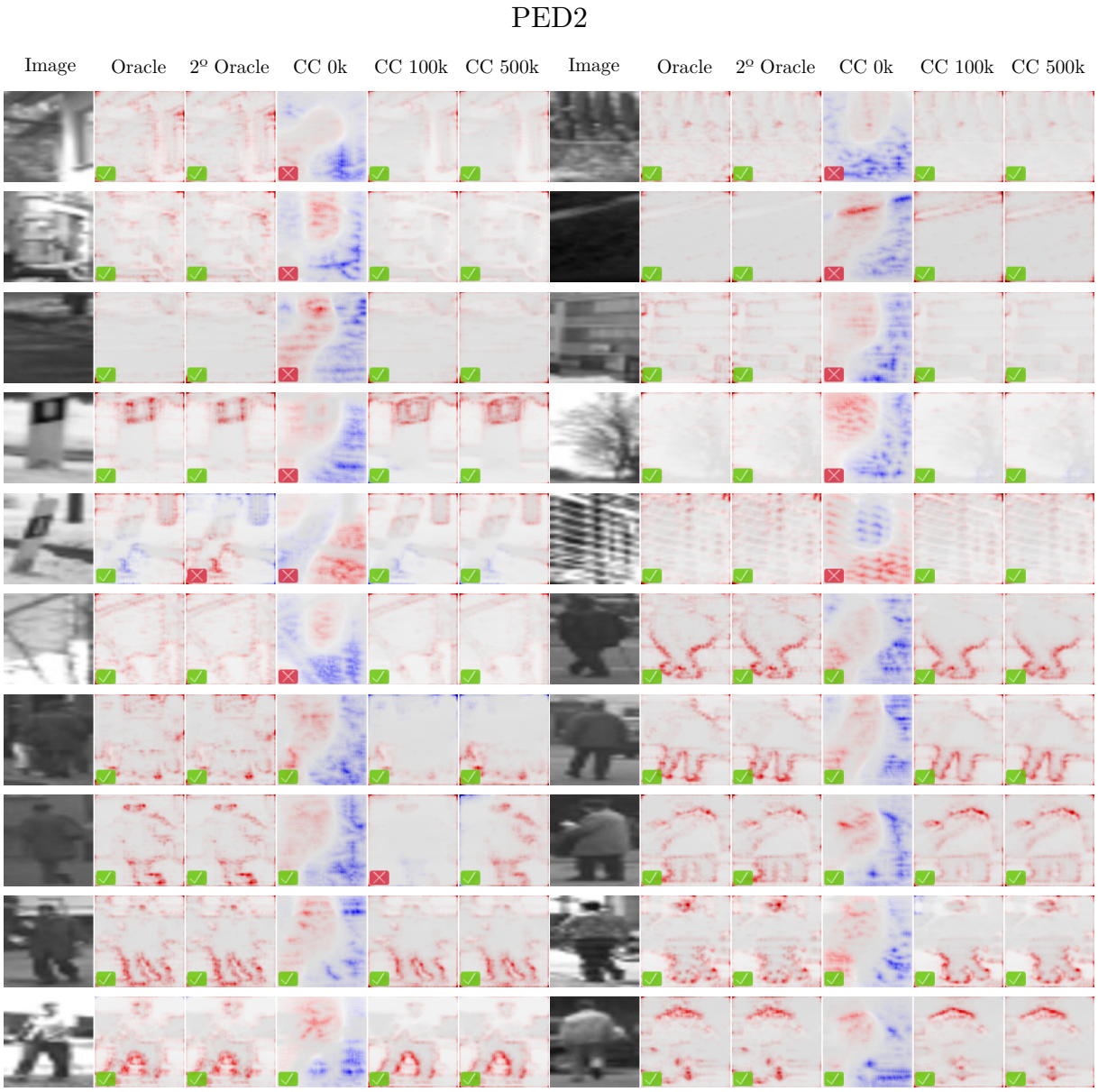



Figure 69 – Examples of PDD heatmaps generated with LRP on Copycat Framework for PED2. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.


Figure 70 – Examples of PDD heatmaps generated with LRP on Copycat Framework for SHN10. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.


Figure 71 – Examples of PDD heatmaps generated with LRP on Copycat Framework for SIG30. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The *green* $V$ and *red* $X$ symbols indicates correct and wrong predictions, respectively.

# APPENDIX C – LRP Heatmaps of NPDD

This appendix presents Examples of NPDD heatmaps generated with LRP on Copycat Framework. Twenty random images of NPDD were selected and their respective heatmaps are presented for each problem.

More heatmaps can be found at:

<https://jeiks.github.io/Stealing_DL_Models/framework-heatmaps/npdd>.

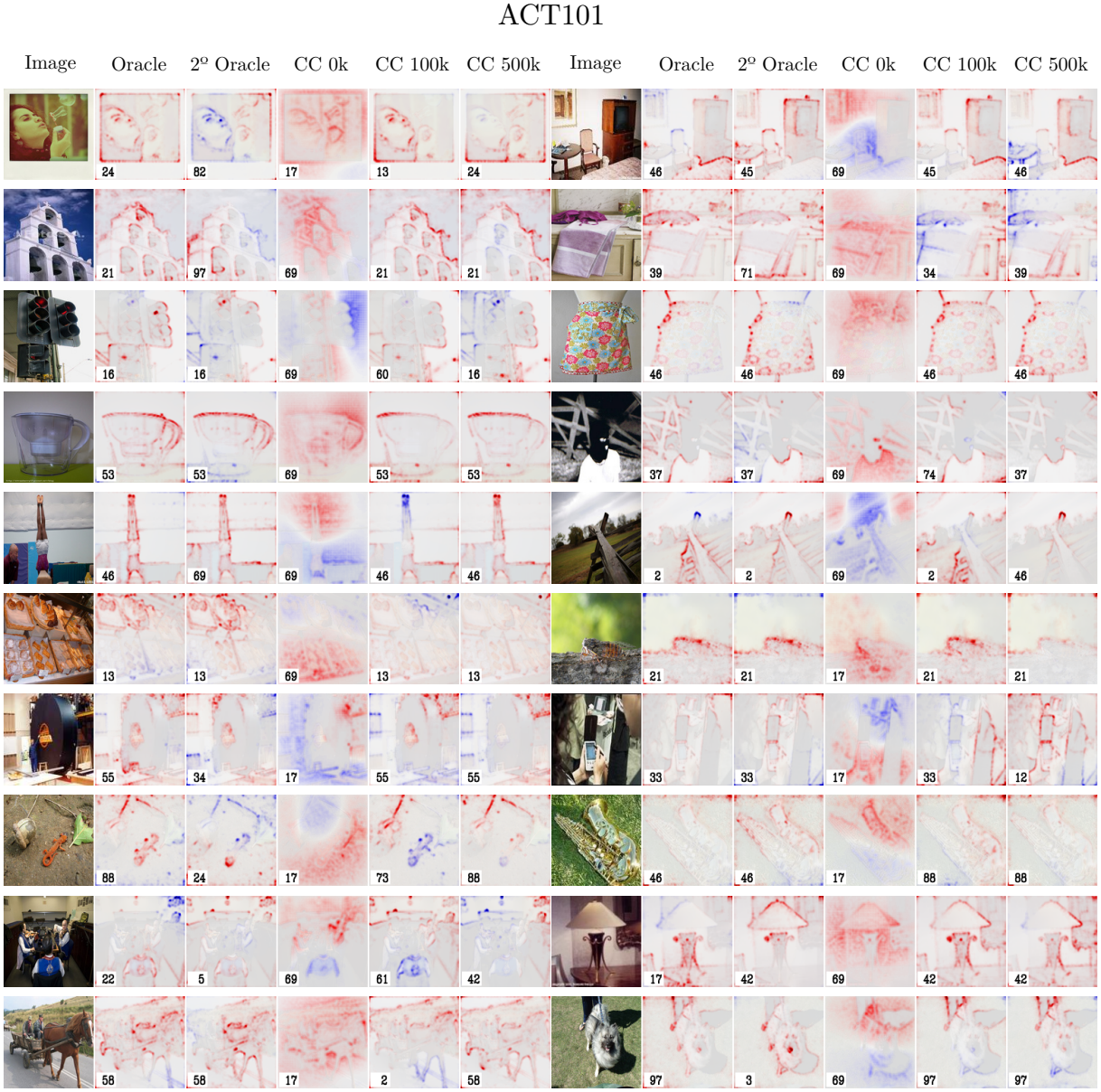



Figure 72 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for ACT101. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

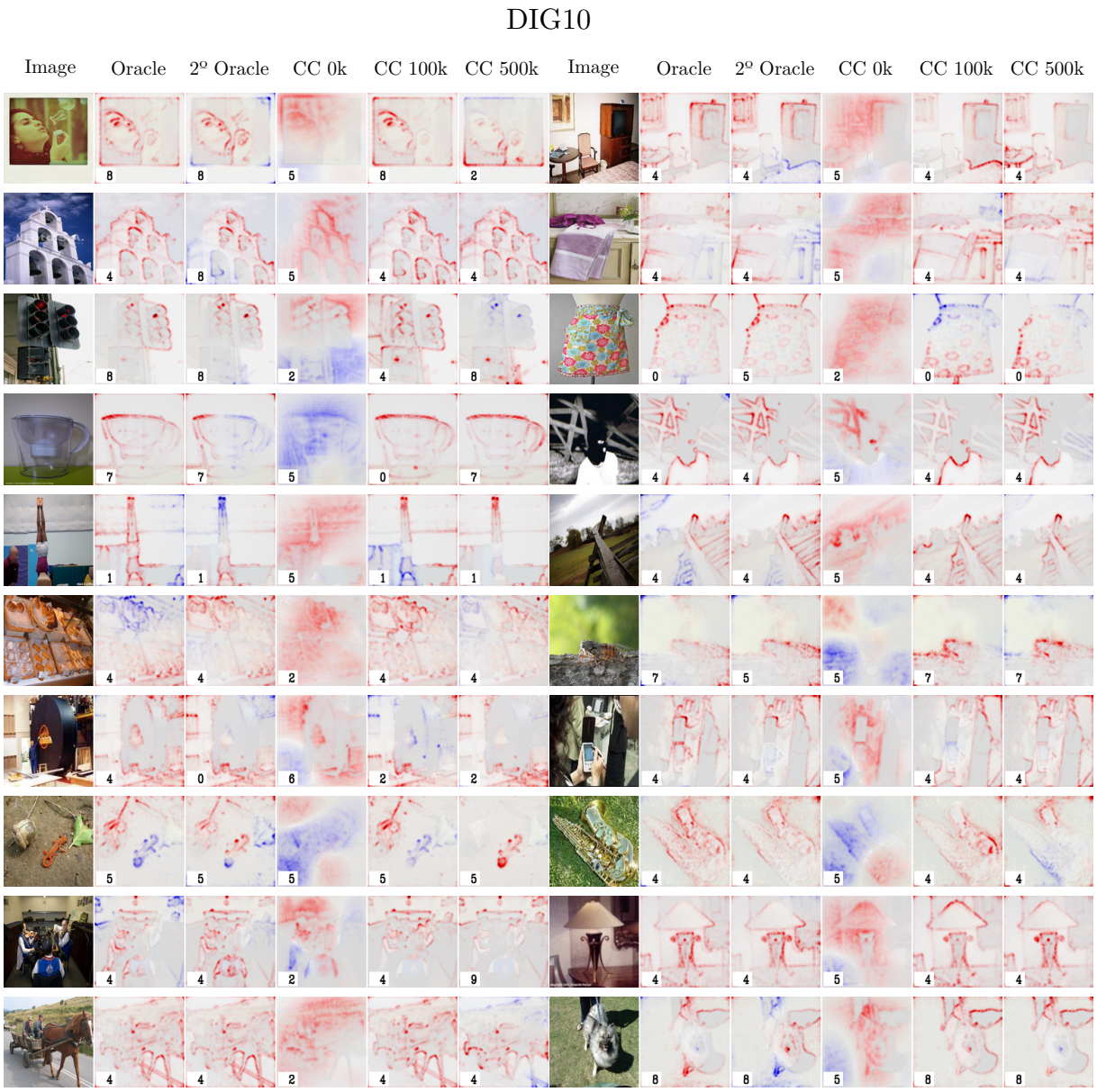



Figure 73 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for DIG10. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

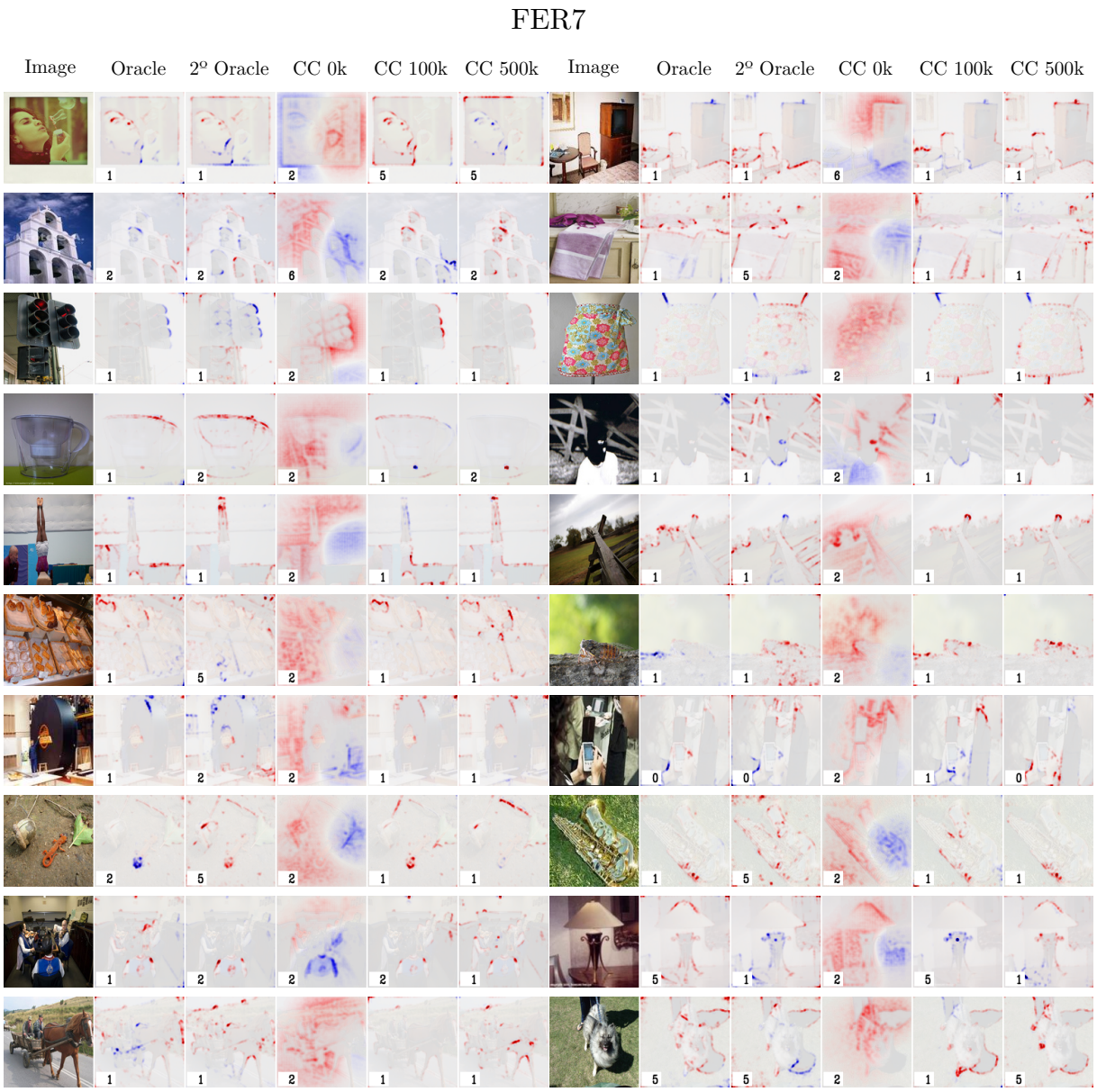



Figure 74 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for FER7. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

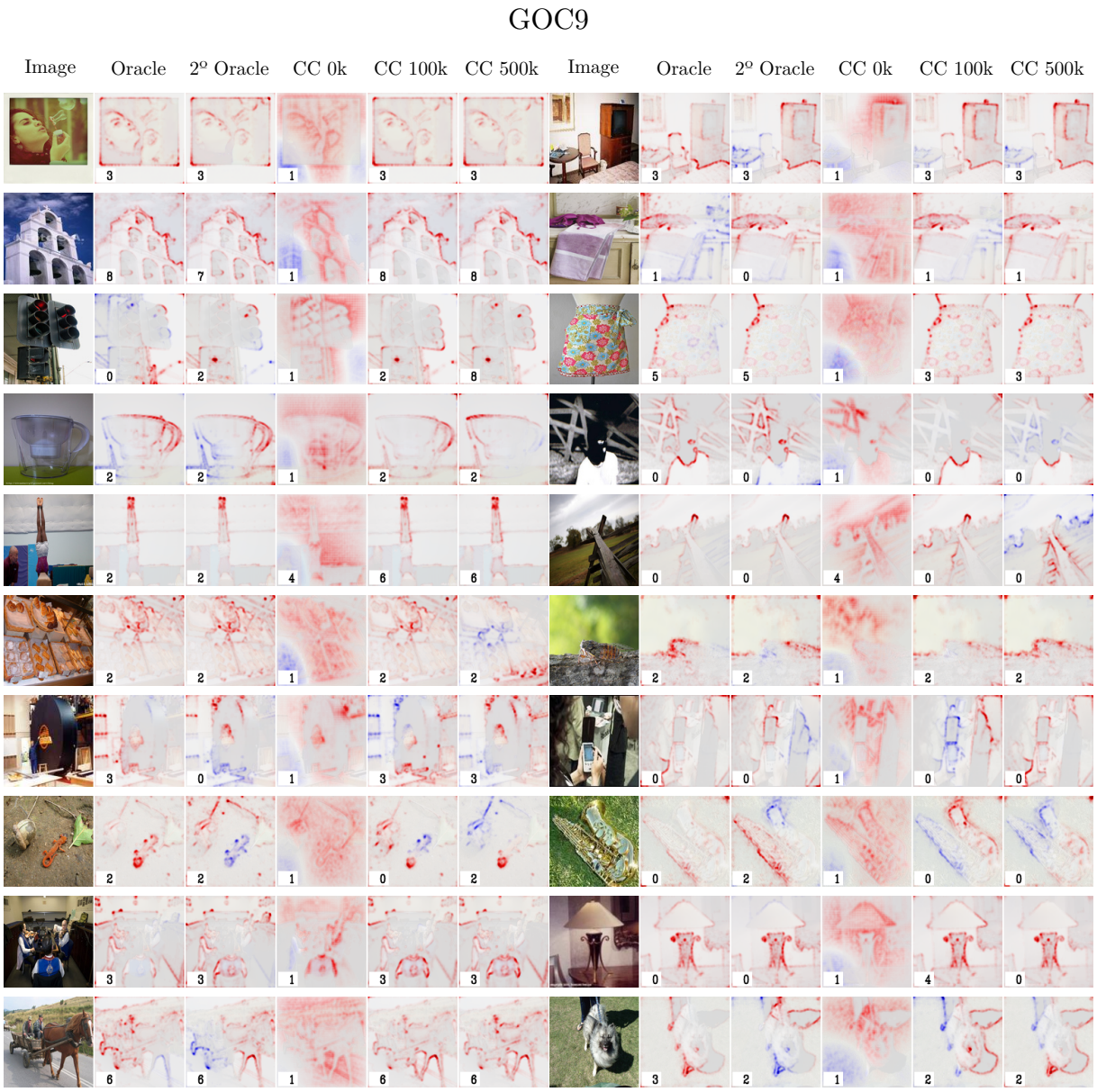



Figure 75 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for GOC9. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.


Figure 76 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for PED2. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.


Figure 77 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for SHN10. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

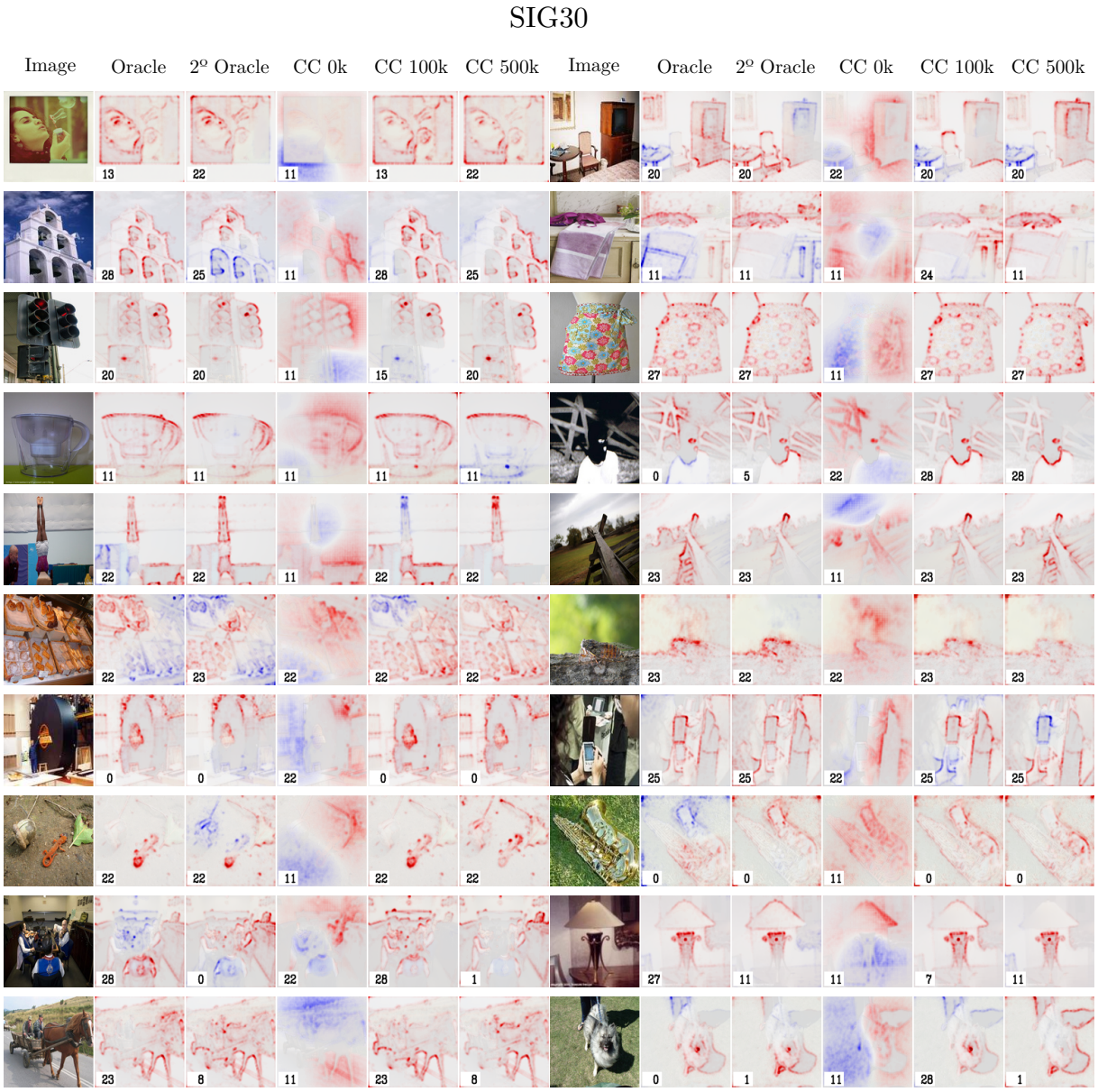



Figure 78 – Examples of NPDD heatmaps generated with LRP on Copycat Framework for SIG30. The red and blue pixels refers to positive and negative relevance, respectively. Each block displays two random images followed by their heatmaps: Oracle, 2º Oracle trained on same ODD, Copycat without training (CC $0k$), Copycat trained with $100k$ of NPDD (CC $100k$), and Copycat trained with $500k$ of NPDD (CC $500k$). The stolen labels are presented on the bottom left corner of each heatmap.

# APPENDIX D – Confusion Matrices and Classification Reports

This Appendix presents the confusion matrices and the classification reports for the proposed problems trained on the Copycat Framework in Pytorch. To facilitate comparison of the reports between the models, we also provide a table with the precision, recall, and F1-Score for each class. For each class, the Precision column represents the proportion of correct predictions made by the model for that class, the Recall column represents the proportion of actual samples from that class that were identified correctly by the model, and the F1-Score column is the harmonic mean of the precision and recall values. The higher F1-Scores are in bold to make them easy to find. Additionally, we present the number of stolen labels per class for both the NPDD and PDD datasets.

As can be seen in the tables, the classes with more stolen labels do not correspond to the classes with higher F1-Scores. However, the classes that have better F1-Scores in the Oracle usually correspond to high F1-Scores in the Copycat and its fine-tuned model.

## D.1 ACT101

This section presents the confusion matrices and the classification reports for the problem ACT101. Given that this problem has 101 classes that do not fit correctly on the page, these confusion matrices were generated as images, and this page is in landscape orientation. In addition, their table caption is smaller than that of other tables, but the table contents present the same metrics.

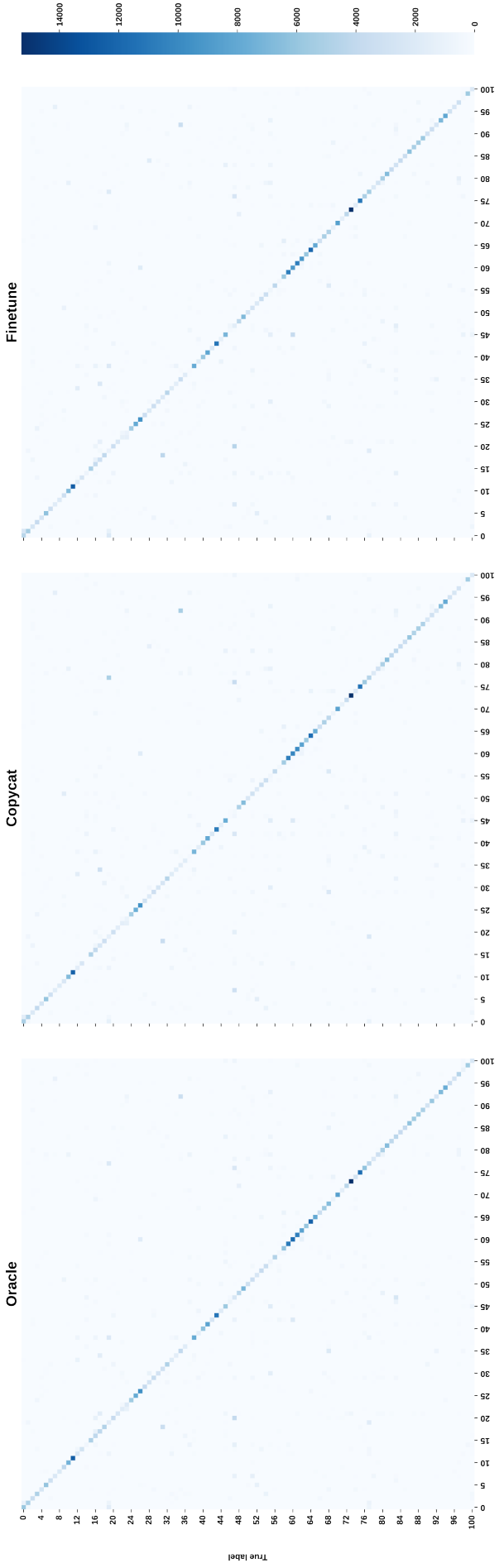



Figure 79 – Confusion Matrices for ACT101 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 18 – Classification reports (precision, recall and F1-Score for each class) for ACT101 models and the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 0.68 | 0.81 | *0.74* | 0.65 | 0.72 | *0.68* | 0.57 | 0.61 | *0.58* | 2,187 | 1,197 |
| 1 | 0.64 | 0.82 | *0.72* | 0.68 | 0.81 | *0.74* | 0.57 | 0.87 | *0.69* | 751 | 938 |
| 2 | 0.67 | 0.57 | *0.62* | 0.59 | 0.37 | *0.45* | 0.55 | 0.41 | *0.47* | 19,558 | 969 |
| 3 | 0.66 | 0.80 | *0.73* | 0.65 | 0.65 | *0.65* | 0.69 | 0.60 | *0.64* | 21,362 | 1,125 |
| 4 | 0.55 | 0.71 | *0.62* | 0.51 | 0.77 | *0.61* | 0.34 | 0.72 | *0.46* | 4,236 | 1,032 |
| 5 | 0.76 | 0.84 | *0.80* | 0.70 | 0.86 | *0.77* | 0.73 | 0.86 | *0.79* | 5,490 | 1,015 |
| 6 | 0.82 | 0.81 | *0.82* | 0.76 | 0.83 | *0.79* | 0.78 | 0.83 | *0.80* | 652 | 1,061 |
| 7 | 0.35 | 0.57 | *0.43* | 0.21 | 0.44 | *0.29* | 0.26 | 0.45 | *0.33* | 4,397 | 1,066 |
| 8 | 0.69 | 0.98 | *0.81* | 0.76 | 0.94 | *0.84* | 0.74 | 0.93 | *0.83* | 2,277 | 1,034 |
| 9 | 0.89 | 0.78 | *0.83* | 0.88 | 0.45 | *0.60* | 0.78 | 0.63 | *0.70* | 343 | 929 |
| 10 | 0.91 | 0.83 | *0.87* | 0.87 | 0.78 | *0.82* | 0.91 | 0.76 | *0.83* | 1,702 | 1,042 |
| 11 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 2,191 | 970 |
| 12 | 0.45 | 0.40 | *0.42* | 0.32 | 0.29 | *0.31* | 0.45 | 0.32 | *0.37* | 5,086 | 236 |
| 13 | 0.49 | 0.94 | *0.64* | 0.45 | 0.86 | *0.59* | 0.48 | 0.87 | *0.62* | 26,087 | 1,331 |
| 14 | 0.12 | 0.16 | *0.14* | 0.02 | 0.01 | *0.01* | 0.11 | 0.21 | *0.14* | 669 | 1,100 |
| 15 | 0.89 | 0.93 | *0.91* | 0.76 | 0.93 | *0.83* | 0.84 | 0.93 | *0.88* | 3,856 | 1,070 |
| 16 | 0.78 | 0.42 | *0.55* | 0.72 | 0.41 | *0.52* | 0.71 | 0.38 | *0.50* | 1,137 | 859 |
| 17 | 0.68 | 0.50 | *0.57* | 0.54 | 0.30 | *0.38* | 0.48 | 0.38 | *0.43* | 544 | 1,136 |
| 18 | 0.55 | 0.94 | *0.70* | 0.47 | 0.66 | *0.55* | 0.47 | 0.80 | *0.60* | 90 | 870 |
| 19 | 0.36 | 0.17 | *0.23* | 0.25 | 0.13 | *0.17* | 0.23 | 0.10 | *0.14* | 1,213 | 250 |
| 20 | 0.39 | 0.60 | *0.47* | 0.49 | 0.51 | *0.50* | 0.33 | 0.53 | *0.41* | 1,585 | 1,006 |
| 21 | 0.31 | 0.75 | *0.44* | 0.56 | 0.52 | *0.54* | 0.31 | 0.71 | *0.43* | **80,143** | 1,274 |
| 22 | 0.51 | 0.45 | *0.48* | 0.33 | 0.39 | *0.36* | 0.36 | 0.33 | *0.34* | 1,890 | 1,204 |
| 23 | 0.75 | 0.41 | *0.53* | 0.63 | 0.22 | *0.32* | 0.44 | 0.22 | *0.29* | 1,208 | 991 |
| 24 | 0.84 | 0.97 | *0.90* | 0.77 | 0.96 | *0.86* | 0.77 | 0.94 | *0.85* | 5,088 | 997 |
| 25 | 0.94 | 1.00 | *0.97* | 0.87 | 1.00 | *0.93* | 0.84 | 1.00 | *0.91* | 1,520 | 948 |
| 26 | 0.91 | 0.80 | *0.85* | 0.84 | 0.81 | *0.83* | 0.92 | 0.78 | *0.84* | 8,053 | 1,127 |
| 27 | 0.86 | 0.86 | *0.86* | 0.88 | 0.68 | *0.77* | 0.78 | 0.75 | *0.77* | 312 | 1,051 |
| 28 | 0.63 | 0.62 | *0.63* | 0.49 | 0.48 | *0.48* | 0.52 | 0.47 | *0.50* | 1,116 | 1,008 |
| 29 | 0.48 | 0.68 | *0.56* | 0.38 | 0.62 | *0.47* | 0.49 | 0.61 | *0.54* | 684 | 614 |
| 30 | 0.39 | 0.88 | *0.54* | 0.40 | 0.81 | *0.53* | 0.40 | 0.82 | *0.54* | 2,329 | 1,225 |
| 31 | 0.85 | 0.43 | *0.57* | 0.63 | 0.38 | *0.47* | 0.73 | 0.30 | *0.42* | 3,306 | 1,089 |
| 32 | 0.64 | 0.77 | *0.70* | 0.55 | 0.74 | *0.63* | 0.62 | 0.70 | *0.66* | 2,567 | 1,101 |
| 33 | 0.53 | 0.50 | *0.52* | 0.42 | 0.52 | *0.46* | 0.43 | 0.51 | *0.47* | 4,287 | 1,188 |
| 34 | 0.39 | 0.47 | *0.43* | 0.21 | 0.45 | *0.29* | 0.28 | 0.44 | *0.34* | 5,869 | 964 |
| 35 | 0.53 | 0.49 | *0.51* | 0.47 | 0.20 | *0.28* | 0.51 | 0.43 | *0.47* | 3,630 | 857 |
| 36 | 0.65 | 0.56 | *0.60* | 0.55 | 0.56 | *0.56* | 0.55 | 0.56 | *0.56* | 508 | 1,289 |
| 37 | 0.13 | 0.04 | *0.06* | 0.08 | 0.06 | *0.07* | 0.05 | 0.04 | *0.05* | 5,382 | 1,219 |
| 38 | 0.58 | 0.84 | *0.68* | 0.64 | 0.78 | *0.71* | 0.54 | 0.84 | *0.65* | 4,750 | **1,943** |
| 39 | 0.47 | 0.32 | *0.38* | 0.29 | 0.35 | *0.32* | 0.36 | 0.39 | *0.38* | 6,012 | 1,124 |
| 40 | 0.95 | 0.95 | *0.95* | 0.93 | 0.90 | *0.92* | 0.94 | 0.90 | *0.92* | 3,560 | 989 |
| 41 | 0.85 | 0.97 | *0.90* | 0.75 | 0.95 | *0.84* | 0.75 | 0.95 | *0.84* | 7,505 | 981 |
| 42 | 0.33 | 0.75 | *0.45* | 0.24 | 0.68 | *0.36* | 0.43 | 0.61 | *0.50* | 7,232 | 1,071 |
| 43 | 0.86 | 0.98 | *0.92* | 0.84 | 0.98 | *0.90* | 0.80 | 0.98 | *0.88* | 588 | 1,043 |
| 44 | 0.50 | 0.56 | *0.53* | 0.44 | 0.43 | *0.44* | 0.49 | 0.40 | *0.44* | 1,917 | 1,267 |
| 45 | 0.61 | 0.58 | *0.59* | 0.54 | 0.79 | *0.64* | 0.49 | 0.71 | *0.58* | 3,473 | 1,058 |
| 46 | 0.19 | 0.31 | *0.24* | 0.04 | 0.07 | *0.05* | 0.07 | 0.11 | *0.08* | 10,179 | 1,259 |
| 47 | 0.35 | 0.16 | *0.22* | 0.17 | 0.02 | *0.04* | 0.29 | 0.10 | *0.15* | 239 | 28 |
| 48 | 0.68 | 0.67 | *0.68* | 0.63 | 0.78 | *0.70* | 0.72 | 0.67 | *0.69* | 6,550 | 1,074 |
| 49 | 0.95 | 0.93 | *0.94* | 0.90 | 0.89 | *0.89* | 0.91 | 0.89 | *0.90* | 4,634 | 1,011 |
| 50 | 0.58 | 0.70 | *0.63* | 0.75 | 0.53 | *0.62* | 0.72 | 0.57 | *0.64* | 546 | 985 |
| 51 | 0.48 | 0.42 | *0.44* | 0.35 | 0.37 | *0.36* | 0.43 | 0.32 | *0.37* | 1,258 | 15 |
| 52 | 0.94 | 0.64 | *0.76* | 0.90 | 0.53 | *0.66* | 0.91 | 0.55 | *0.69* | 6,453 | 957 |
| 53 | 0.86 | 0.69 | *0.76* | 0.93 | 0.63 | *0.75* | 0.88 | 0.65 | *0.75* | **38,496** | 1,016 |
| 54 | 0.54 | 0.54 | *0.54* | 0.44 | 0.50 | *0.47* | 0.46 | 0.52 | *0.49* | 2,775 | 1,074 |
| 55 | 0.37 | 0.10 | *0.15* | 0.16 | 0.07 | *0.10* | 0.16 | 0.07 | *0.10* | 992 | 799 |
| 56 | 0.71 | 0.94 | *0.81* | 0.52 | 0.82 | *0.64* | 0.54 | 0.83 | *0.65* | 1,789 | 992 |
| 57 | 0.17 | 0.12 | *0.14* | 0.22 | 0.14 | *0.17* | 0.18 | 0.11 | *0.14* | 1,198 | 738 |
| 58 | 0.91 | 0.71 | *0.80* | 0.90 | 0.65 | *0.76* | 0.93 | 0.66 | *0.77* | 787 | 998 |
| 59 | 0.93 | 0.95 | *0.94* | 0.94 | 0.94 | *0.94* | 0.90 | 0.92 | *0.91* | 10,891 | 988 |
| 60 | 0.78 | 0.78 | *0.78* | 0.75 | 0.70 | *0.72* | 0.78 | 0.59 | *0.68* | 5,028 | 927 |
| 61 | 0.84 | 0.88 | *0.86* | 0.84 | 0.84 | *0.84* | 0.82 | 0.89 | *0.85* | 3,271 | 1,005 |
| 62 | 0.98 | 0.82 | *0.89* | 0.99 | 0.83 | *0.91* | 0.99 | 0.89 | *0.94* | 663 | 1,020 |
| 63 | 0.86 | 0.98 | *0.91* | 0.80 | 0.96 | *0.87* | 0.76 | 0.97 | *0.85* | 1,613 | 1,375 |
| 64 | **1.00** | **0.92** | ***0.96*** | 1.00 | 0.88 | *0.93* | 1.00 | 0.89 | *0.94* | 813 | 980 |
| 65 | 0.85 | 0.96 | *0.90* | 0.88 | 0.95 | *0.91* | 0.83 | 0.96 | *0.89* | 7,250 | 1,017 |
| 66 | 0.67 | 0.69 | *0.68* | 0.62 | 0.63 | *0.63* | 0.53 | 0.60 | *0.57* | 573 | 264 |
| 67 | 0.76 | 0.74 | *0.75* | 0.78 | 0.67 | *0.72* | 0.73 | 0.67 | *0.70* | 4,864 | 1,236 |
| 68 | 0.98 | 0.57 | *0.72* | 0.98 | 0.37 | *0.53* | 0.97 | 0.41 | *0.58* | 481 | 971 |
| 69 | 0.33 | 0.25 | *0.28* | 0.30 | 0.23 | *0.26* | 0.33 | 0.34 | *0.34* | 5,960 | 1,172 |
| 70 | 0.92 | 0.95 | *0.94* | 0.89 | 0.95 | *0.92* | 0.90 | 0.96 | *0.93* | 563 | 990 |
| 71 | 0.63 | 0.61 | *0.62* | 0.58 | 0.48 | *0.52* | 0.66 | 0.54 | *0.59* | 19,784 | 1,109 |
| 72 | 0.74 | 0.82 | *0.78* | 0.76 | 0.71 | *0.73* | 0.73 | 0.80 | *0.76* | 5,739 | 876 |
| 73 | 0.96 | 0.91 | *0.93* | 0.96 | 0.93 | *0.94* | 0.97 | 0.90 | *0.94* | 5,330 | 1,021 |
| 74 | 0.46 | 0.49 | *0.47* | 0.32 | 0.38 | *0.35* | 0.44 | 0.31 | *0.36* | 3,240 | 997 |
| 75 | 0.93 | 0.96 | *0.94* | 0.93 | 0.96 | *0.94* | 0.95 | 0.90 | *0.93* | 2,546 | 1,104 |
| 76 | 0.59 | 0.66 | *0.62* | 0.48 | 0.61 | *0.54* | 0.52 | 0.53 | *0.53* | 475 | |
| 77 | 0.61 | 0.41 | *0.49* | 0.45 | 0.46 | *0.46* | 0.68 | 0.52 | *0.59* | 1,661 | 930 |
| 78 | 0.50 | 0.55 | *0.52* | 0.33 | 0.54 | *0.41* | 0.50 | 0.54 | *0.52* | 7,212 | 1,131 |
| 79 | 0.36 | 0.68 | *0.47* | 0.34 | 0.66 | *0.44* | 0.31 | 0.58 | *0.40* | 2,685 | 1,263 |
| 80 | 0.72 | 0.57 | *0.63* | 0.66 | 0.58 | *0.62* | 0.70 | 0.57 | *0.63* | 3,463 | 1,057 |
| 81 | 0.90 | 0.97 | *0.93* | 0.79 | 0.98 | *0.88* | 0.75 | 0.98 | *0.85* | 3,425 | 1,010 |
| 82 | 0.96 | 0.90 | *0.93* | 0.89 | 0.91 | *0.90* | 0.93 | 0.81 | *0.87* | 3,143 | 1,032 |
| 83 | 0.57 | 0.35 | *0.44* | 0.48 | 0.35 | *0.41* | 0.41 | 0.24 | *0.30* | 2,057 | 1,451 |
| 84 | 0.93 | 1.00 | *0.96* | 0.70 | 0.99 | *0.82* | 0.63 | 1.00 | *0.77* | 121 | 808 |
| 85 | 0.66 | 0.81 | *0.73* | 0.80 | 0.73 | *0.77* | 0.75 | 0.74 | *0.74* | 801 | 833 |
| 86 | 0.86 | 0.93 | *0.89* | 0.81 | 0.92 | *0.86* | 0.76 | 0.94 | *0.84* | 1,102 | 1,000 |
| 87 | 0.95 | 0.91 | *0.93* | 0.88 | 0.81 | *0.84* | 0.87 | 0.84 | *0.85* | 960 | 648 |
| 88 | 0.77 | 0.81 | *0.79* | 0.67 | 0.77 | *0.72* | 0.70 | 0.76 | *0.73* | 7,395 | 992 |
| 89 | 0.83 | 0.87 | *0.85* | 0.80 | 0.88 | *0.84* | 0.80 | 0.99 | *0.89* | 2,080 | 1,008 |
| 90 | 0.78 | 0.59 | *0.67* | 0.83 | 0.53 | *0.65* | 0.63 | 0.62 | *0.63* | 889 | 530 |
| 91 | 0.83 | 0.95 | *0.88* | 0.60 | 0.51 | *0.55* | 0.61 | 0.53 | *0.57* | 72 | 728 |
| 92 | 0.25 | 0.67 | *0.37* | 0.20 | 0.64 | *0.31* | 0.28 | 0.65 | *0.39* | 5,945 | 1,381 |
| 93 | 0.78 | 0.95 | *0.85* | 0.70 | 0.92 | *0.79* | 0.70 | 0.94 | *0.81* | 1,471 | 993 |
| 94 | 0.95 | 0.96 | *0.95* | **0.97** | **1.00** | ***0.98*** | **0.96** | **1.00** | ***0.98*** | 4,373 | 1,019 |
| 95 | 0.87 | 0.92 | *0.89* | 0.84 | 0.79 | *0.81* | 0.78 | 0.80 | *0.79* | 1,512 | 1,356 |
| 96 | 0.52 | 0.88 | *0.66* | 0.41 | 0.80 | *0.54* | 0.45 | 0.77 | *0.57* | 1,269 | 1,019 |
| 97 | 0.75 | 0.61 | *0.67* | 0.76 | 0.36 | *0.49* | 0.73 | 0.42 | *0.53* | 17,791 | 121 |
| 98 | 0.86 | 0.30 | *0.44* | 0.42 | 0.09 | *0.15* | 0.62 | 0.09 | *0.16* | 806 | 862 |
| 99 | 0.77 | 0.91 | *0.83* | 0.75 | 0.91 | *0.83* | 0.79 | 0.92 | *0.85* | 3,004 | 972 |
| 100 | 0.51 | 0.38 | *0.43* | 0.50 | 0.34 | *0.41* | 0.62 | 0.33 | *0.43* | 3,441 | 1,109 |

## D.2 DIG10

This section presents the confusion matrices and the classification reports for the problem DIG10.

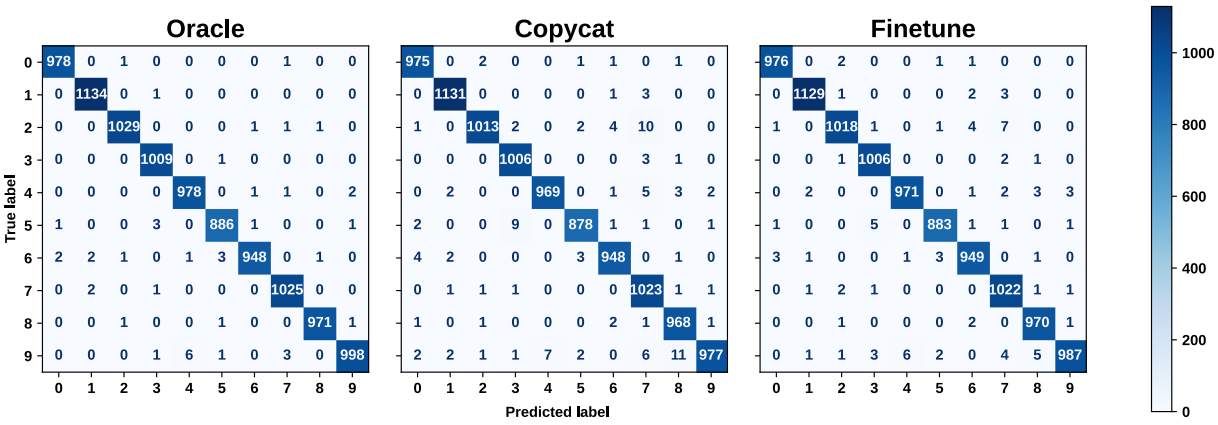



Figure 80 – Confusion Matrices for DIG10 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 19 – Classification reports for DIG10 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | **1.00** | **1.00** | ***1.00*** | 0.99 | 0.99 | *0.99* | **0.99** | **1.00** | ***1.00*** | 39,503 | 3,746 |
| 1 | **1.00** | **1.00** | ***1.00*** | **0.99** | **1.00** | ***1.00*** | **1.00** | **0.99** | ***1.00*** | 31,737 | 6,007 |
| 2 | **1.00** | **1.00** | ***1.00*** | 1.00 | 0.98 | *0.99* | 0.99 | 0.99 | *0.99* | 79,059 | **8,735** |
| 3 | **0.99** | **1.00** | ***1.00*** | 0.99 | 1.00 | *0.99* | 0.99 | 1.00 | *0.99* | 20,467 | 6,322 |
| 4 | 0.99 | 1.00 | *0.99* | 0.99 | 0.99 | *0.99* | 0.99 | 0.99 | *0.99* | **110,732** | 4,161 |
| 5 | 0.99 | 0.99 | *0.99* | 0.99 | 0.98 | *0.99* | 0.99 | 0.99 | *0.99* | 48,643 | 4,928 |
| 6 | 1.00 | 0.99 | *0.99* | 0.99 | 0.99 | *0.99* | 0.99 | 0.99 | *0.99* | 5,577 | 5,733 |
| 7 | **0.99** | **1.00** | ***1.00*** | 0.97 | 1.00 | *0.98* | 0.98 | 0.99 | *0.99* | 33,865 | 3,930 |
| 8 | **1.00** | **1.00** | ***1.00*** | 0.98 | 0.99 | *0.99* | 0.99 | 1.00 | *0.99* | **92,817** | **7,584** |
| 9 | 1.00 | 0.99 | *0.99* | 0.99 | 0.97 | *0.98* | 0.99 | 0.98 | *0.99* | 37,600 | 3,854 |

## D.3 FER7

This section presents the confusion matrices and the classification reports for the problem FER7.

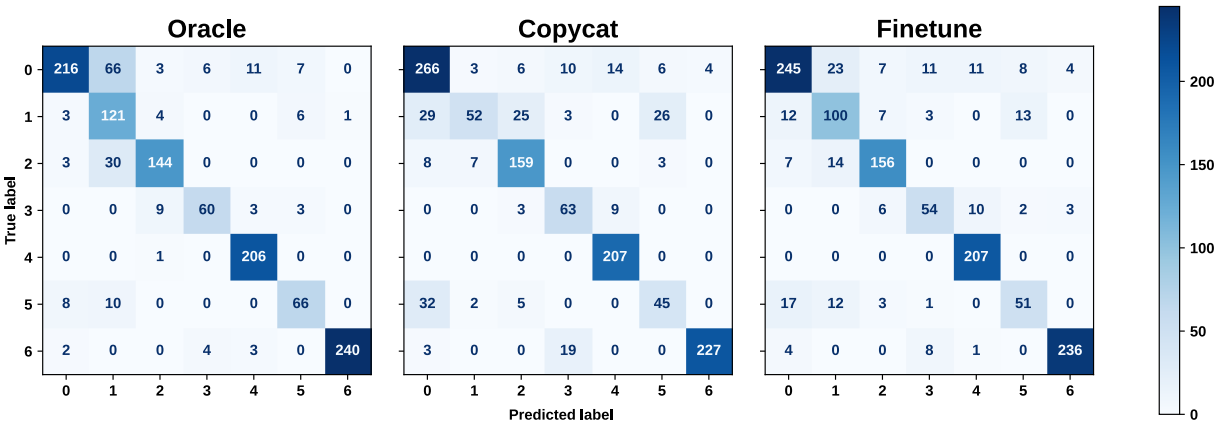



Figure 81 – Confusion Matrices for FER7 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 20 – Classification reports for FER7 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 0.93 | 0.70 | *0.80* | 0.79 | 0.86 | *0.82* | 0.86 | 0.79 | *0.82* | 5,791 | 7,719 |
| 1 | 0.53 | 0.90 | *0.67* | 0.81 | 0.39 | *0.52* | 0.67 | 0.74 | *0.70* | **404,910** | **12,478** |
| 2 | 0.89 | 0.81 | *0.85* | 0.80 | 0.90 | *0.85* | 0.87 | 0.88 | *0.88* | 27,578 | **12,320** |
| 3 | 0.86 | 0.80 | *0.83* | 0.66 | 0.84 | *0.74* | 0.70 | 0.72 | *0.71* | 1,800 | 6,519 |
| 4 | **0.92** | **1.00** | ***0.96*** | **0.90** | **1.00** | ***0.95*** | **0.90** | **1.00** | ***0.95*** | 1,406 | 9,356 |
| 5 | 0.80 | 0.79 | *0.80* | 0.56 | 0.54 | *0.55* | 0.69 | 0.61 | *0.65* | **53,604** | 8,611 |
| 6 | **1.00** | **0.96** | ***0.98*** | **0.98** | **0.91** | ***0.95*** | **0.97** | **0.95** | ***0.96*** | 4,911 | 8,657 |

## D.4 GOC9

This section presents the confusion matrices and the classification reports for the problem GOC9.

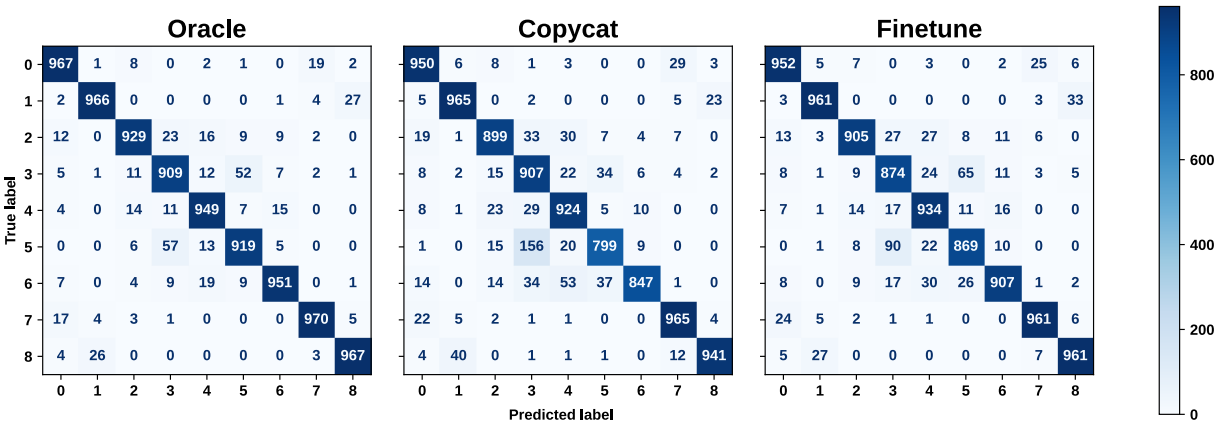



Figure 82 – Confusion Matrices for GOC9 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 21 – Classification reports for GOC9 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 0.95 | 0.97 | *0.96* | 0.92 | 0.95 | *0.94* | 0.93 | 0.95 | *0.94* | 72,351 | **39,258** |
| 1 | **0.97** | **0.97** | ***0.97*** | **0.95** | **0.97** | ***0.96*** | **0.96** | **0.96** | ***0.96*** | 22,407 | 31,001 |
| 2 | 0.95 | 0.93 | *0.94* | 0.92 | 0.90 | *0.91* | 0.95 | 0.91 | *0.93* | **136,676** | **41,849** |
| 3 | 0.90 | 0.91 | *0.90* | 0.78 | 0.91 | *0.84* | 0.85 | 0.87 | *0.86* | **101,781** | 31,802 |
| 4 | 0.94 | 0.95 | *0.94* | 0.88 | 0.92 | *0.90* | 0.90 | 0.93 | *0.92* | 22,166 | 30,947 |
| 5 | 0.92 | 0.92 | *0.92* | 0.90 | 0.80 | *0.85* | 0.89 | 0.87 | *0.88* | 49,087 | 16,881 |
| 6 | 0.96 | 0.95 | *0.96* | 0.97 | 0.85 | *0.90* | 0.95 | 0.91 | *0.93* | 34,148 | 27,443 |
| 7 | **0.97** | **0.97** | ***0.97*** | **0.94** | **0.97** | ***0.95*** | **0.96** | **0.96** | ***0.96*** | 32,999 | 28,113 |
| 8 | **0.96** | **0.97** | ***0.97*** | **0.97** | **0.94** | ***0.95*** | 0.95 | 0.96 | *0.95* | 28,385 | 21,806 |

## D.5 PED2

This section presents the confusion matrices and the classification reports for the problem PED2.

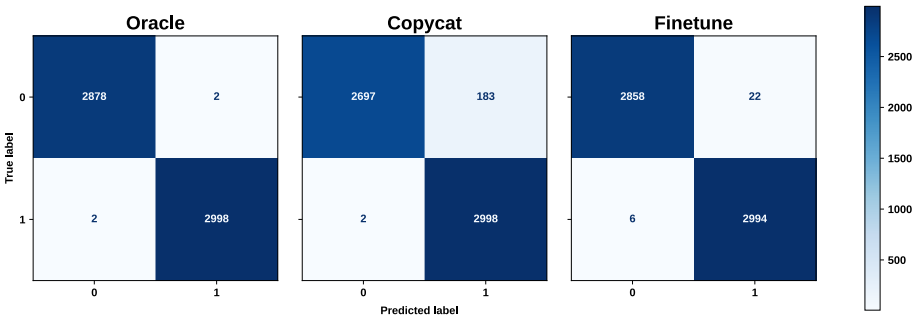



Figure 83 – Confusion Matrices for PED2 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 22 – Classification reports for PED2 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 1.00 | 1.00 | *1.00* | 1.00 | 0.94 | *0.97* | 1.00 | 0.99 | *1.00* | 75,243 | 9,400 |
| 1 | 1.00 | 1.00 | *1.00* | 0.94 | 1.00 | *0.97* | 0.99 | 1.00 | *1.00* | 424,757 | 10,200 |

## D.6 SHN10

This section presents the confusion matrices and the classification reports for the problem SHN10.

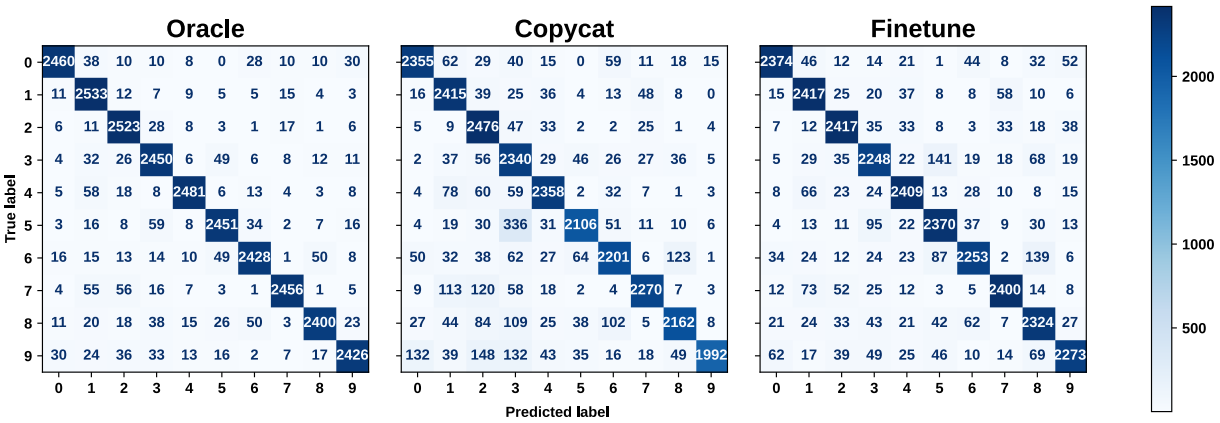



Figure 84 – Confusion Matrices for SHN10 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 23 – Classification reports for SHN10 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 0.96 | 0.94 | *0.95* | **0.90** | **0.90** | ***0.90*** | **0.93** | **0.91** | ***0.92*** | 15,489 | 1,752 |
| 1 | 0.90 | 0.97 | *0.94* | 0.85 | 0.93 | *0.89* | 0.89 | 0.93 | *0.91* | **214,467** | **5,197** |
| 2 | 0.93 | 0.97 | *0.95* | 0.80 | 0.95 | *0.87* | **0.91** | **0.93** | ***0.92*** | 54,150 | **4,181** |
| 3 | 0.92 | 0.94 | *0.93* | 0.73 | 0.90 | *0.81* | 0.87 | 0.86 | *0.87* | 40,090 | 2,853 |
| 4 | **0.97** | **0.95** | ***0.96*** | **0.90** | **0.91** | ***0.90*** | **0.92** | **0.93** | ***0.92*** | **72,134** | 2,485 |
| 5 | 0.94 | 0.94 | *0.94* | 0.92 | 0.81 | *0.86* | 0.87 | 0.91 | *0.89* | 8,844 | 2,386 |
| 6 | 0.95 | 0.93 | *0.94* | 0.88 | 0.85 | *0.86* | 0.91 | 0.87 | *0.89* | 30,344 | 1,985 |
| 7 | **0.97** | **0.94** | ***0.96*** | **0.93** | **0.87** | ***0.90*** | **0.94** | **0.92** | ***0.93*** | 39,269 | 1,981 |
| 8 | 0.96 | 0.92 | *0.94* | 0.90 | 0.83 | *0.86* | 0.86 | 0.89 | *0.87* | 9,826 | 1,609 |
| 9 | 0.96 | 0.93 | *0.94* | 0.98 | 0.76 | *0.86* | 0.93 | 0.87 | *0.90* | 15,387 | 1,603 |

## D.7 SIG30

This section presents the confusion matrices and the classification reports for the problem SIG30.

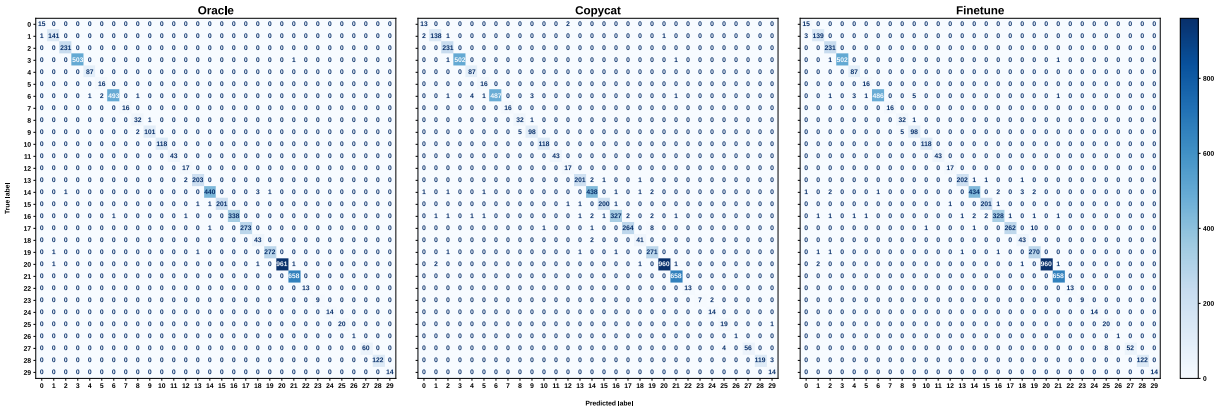



Figure 85 – Confusion Matrices for SIG30 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL.

Table 24 – Classification reports for SIG30 models: Oracle, Copycat trained with $500k$ of NPDD, and the Copycat fine-tuned with PDD-SL. This table presents the precision, recall and F1-Score for each class. In addition, it is also presented the number of stolen labels obtained per class.

| Classes | Oracle | | | Copycat NPDD | | | Fine-tuned Copycat | | | #NPDD | #PDD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | Precision | Recall | *F1-Score* | | |
| 0 | 0.94 | 1.00 | *0.97* | 0.81 | 0.87 | *0.84* | 0.79 | 1.00 | *0.88* | **55,006** | 1,051 |
| 1 | 0.99 | 0.99 | *0.99* | 0.98 | 0.97 | *0.98* | 0.97 | 0.98 | *0.98* | 35,660 | 1,009 |
| 2 | **1.00** | **1.00** | ***1.00*** | 0.97 | 1.00 | *0.99* | 0.97 | 1.00 | *0.99* | 11,364 | 1,055 |
| 3 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 1,816 | 872 |
| 4 | 0.99 | 1.00 | *0.99* | 0.95 | 1.00 | *0.97* | 0.96 | 1.00 | *0.98* | 7,151 | 1,019 |
| 5 | 0.89 | 1.00 | *0.94* | 0.84 | 1.00 | *0.91* | 0.89 | 1.00 | *0.94* | 6,033 | 1,002 |
| 6 | 1.00 | 0.99 | *0.99* | 1.00 | 0.98 | *0.99* | 1.00 | 0.98 | *0.99* | 4,623 | 924 |
| 7 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 5,219 | 944 |
| 8 | 0.91 | 0.97 | *0.94* | 0.86 | 0.97 | *0.91* | 0.86 | 0.97 | *0.91* | 25,388 | 903 |
| 9 | 0.99 | 0.98 | *0.99* | 0.96 | 0.95 | *0.96* | 0.94 | 0.95 | *0.95* | 7,395 | 931 |
| 10 | **1.00** | **1.00** | ***1.00*** | **0.99** | **1.00** | ***1.00*** | **0.99** | **1.00** | ***1.00*** | 6,224 | 1,069 |
| 11 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 7,938 | 994 |
| 12 | 0.85 | 1.00 | *0.92* | 0.85 | 1.00 | *0.92* | 0.94 | 1.00 | *0.97* | 539 | 882 |
| 13 | 0.99 | 0.99 | *0.99* | 0.99 | 0.98 | *0.98* | 0.99 | 0.99 | *0.99* | 5,365 | 1,128 |
| 14 | 1.00 | 0.99 | *0.99* | 0.98 | 0.98 | *0.98* | 0.99 | 0.98 | *0.98* | 2,685 | 947 |
| 15 | **1.00** | **0.99** | ***1.00*** | 0.99 | 0.99 | *0.99* | 0.99 | 0.99 | *0.99* | 3,356 | 961 |
| 16 | **1.00** | **0.99** | ***1.00*** | 0.99 | 0.96 | *0.98* | 0.99 | 0.96 | *0.98* | 4,804 | 836 |
| 17 | **1.00** | **1.00** | ***1.00*** | 0.99 | 0.96 | *0.98* | 1.00 | 0.96 | *0.98* | 11,253 | 1,141 |
| 18 | 0.91 | 1.00 | *0.96* | 0.93 | 0.95 | *0.94* | 0.90 | 1.00 | *0.95* | 938 | 884 |
| 19 | 1.00 | 0.99 | *0.99* | 0.96 | 0.99 | *0.97* | 0.95 | 0.99 | *0.97* | 25,951 | **1,351** |
| 20 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 23,990 | 1,078 |
| 21 | **1.00** | **1.00** | ***1.00*** | **0.99** | **1.00** | ***1.00*** | **0.99** | **1.00** | ***1.00*** | 13,951 | 1,000 |
| 22 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **62,977** | 1,011 |
| 23 | **1.00** | **1.00** | ***1.00*** | 1.00 | 0.78 | *0.88* | **1.00** | **1.00** | ***1.00*** | 40,676 | 968 |
| 24 | **1.00** | **1.00** | ***1.00*** | 0.88 | 1.00 | *0.93* | **1.00** | **1.00** | ***1.00*** | 24,106 | **1,268** |
| 25 | **1.00** | **1.00** | ***1.00*** | 0.83 | 0.95 | *0.88* | 0.71 | 1.00 | *0.83* | 19,344 | 829 |
| 26 | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | **1.00** | **1.00** | ***1.00*** | 8,175 | 888 |
| 27 | **1.00** | **1.00** | ***1.00*** | 1.00 | 0.93 | *0.97* | 1.00 | 0.87 | *0.93* | 36,182 | 1,094 |
| 28 | **1.00** | **1.00** | ***1.00*** | 1.00 | 0.98 | *0.99* | **1.00** | **1.00** | ***1.00*** | 31,068 | 995 |
| 29 | **1.00** | **1.00** | ***1.00*** | 0.78 | 1.00 | *0.88* | **1.00** | **1.00** | ***1.00*** | 10,823 | 966 |