eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor Thesis

Bachelor Degree in Artificial Intelligence

.

Investigating Adversarial Vulnerabilities and Defensive Strategies in Prototype-Based Self-Explaining Deep Neural Networks

Jon Irastorza Ancín

Advisors
Jon Vadillo Jueguen

June 23, 2024

Acknowledgements

I would like to express my deepest gratitude to my family and friends for their unwavering support and encouragement throughout the course of my studies. Your love, patience, and belief in me have been the cornerstone of my journey, providing me with the strength to overcome challenges and strive for excellence.

To my advisor, Jon Vadillo, I am profoundly grateful for your constant support and guidance throughout my research. I could not have asked for a better advisor. Thank you, Jon.

Summary

A crucial limitation of current Deep Learning models is their inability to explain or justify their decisions. To address this issue, self-explaining models have been developed, designed to provide explanations for their predictions. Prototype-based neural networks, for example, enhance interpretability by offering explanations based on learned prototypes. However, the reliability of these models has not been extensively studied, leaving their robustness in adversarial scenarios uncertain. This thesis investigates the adversarial robustness in prototype-based self-explaining neural networks, which are susceptible to adversarial attacks. These attacks involve subtly altering the inputs in ways that are nearly imperceptible to humans but cause the model to produce incorrect predictions or misleading explanations. This research presents a comprehensive framework for generating and defending against adversarial attacks on these networks. Various attack paradigms are explored, including class-changing and explanation-altering attacks, both targeted and untargeted. To counter these threats, we propose robust training methodologies and tailored modifications to improve the resilience of the models while maintaining interpretable explanations. Experimental results demonstrate the low robustness of prototype-based self-explainable neural networks as well as the effectiveness of our defensive strategies, highlighting the importance of integrating robustness into the interpretability of neural networks. These findings contribute to the development of more secure and reliable AI systems capable of providing transparent and trustworthy explanations while maintaining robustness against adversarial perturbations.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In recent years, Deep Learning (DL) has emerged as a powerful paradigm in the field of Artificial Intelligence (AI), revolutionizing various domains including computer vision, natural language processing, speech recognition, and robotics. The remarkable success of DL can be attributed to its ability to automatically learn hierarchical representations of data directly from raw inputs, thereby alleviating the need for handcrafted features and domain-specific knowledge.

The evolution of DL has undoubtedly revolutionized AI, but with its soaring complexity comes a critical concern: trustworthiness. Deep Neural Networks (DNNs), while incredibly powerful, often operate as "black-boxes", leaving stakeholders in sensitive domains like healthcare, finance, and autonomous driving uneasy about their decisions. The opacity of these networks raises significant questions about their reliability and interpretability, highlighting the urgent need for explainability methods. Understanding how DNNs arrive at their decisions is not just a matter of academic curiosity—it is essential for building trust. In healthcare, misinterpretations could lead to incorrect diagnoses or treatment plans. In finance, opaque models may make unfair decisions with far-reaching economic consequences. And in autonomous driving, the inability to comprehend the reasoning behind a system's actions could result in accidents with potentially catastrophic outcomes.

In addition to interpretability, another critical aspect of deploying DL models in real-world applications is ensuring their robustness and stability, meaning that these models are able to maintain performance and be consistent with their predictions despite variations in the input data. Adversarial attacks are a present threat for DL models because they are able to harm the performance and consistency of the models by introducing small and imperceptible perturbations to input data with the intention of misleading the model's predictions. These perturbations, often indistinguishable to human observers, can cause DNNs to produce incorrect outputs with potentially severe consequences. Robustness and interpretability not only enhance the reliability and safety of AI systems but also fosters trust among users and stakeholders, fundamental for the widespread adoption of AI.

1.1 Contents

This thesis is structured into 10 chapters, organized as follows:

- In chapter 2, the objectives and planification of the project are exposed.
- In chapter 3, an overview of deep learning, adversarial attacks, and explainable models is provided.
- In chapter 4, the experimental model architecture and functionality are described.
- In chapter 5, adversarial attacks against prototype-based neural networks are discussed.
- In chapter 6, the experimental framework and methodologies used in the study are detailed.
- In chapter 7, the robustness of prototype-based neural networks is evaluated.
- In chapter 8, strategies for improving the robustness of prototype-based neural networks are explored.
- In chapter 9, a deeper analysis of robust prototype-based neural networks is provided.
- In chapter 10, the conclusions of the research are summarized.

CHAPTER 2

Objectives and Planification

In this chapter, the objectives and planning for this bachelor's thesis project are presented.

2.1 Objectives

Currently, the robustness of self-explainable neural networks against adversarial attacks has not been extensively studied in the literature, making their reliability in adverse scenarios uncertain. Given this gap in knowledge, the primary goal of this bachelor's thesis is to analyze the vulnerability of these networks to adversarial attacks and to propose strategies to enhance their robustness. The specific objectives are as follows:

Enhance Interpretability in Prototype-Based Neural Networks Understand the functioning of prototype-based neural networks, how they make predictions and provide explanations. Identify potential limitations in terms of interpretability and address these limitations through modifications in the architecture and the training process.

Develop a Framework for Adversarial Attacks on Prototype-Based Neural Networks Conduct a review of the most commonly used adversarial attacks in the literature and adapt them to attack prototype-based neural networks. Develop a framework that allows for the generation of a wide variety of adversarial attacks capable of altering both the predictions and explanations provided by these architectures.

Improve the Robustness of Prototype-Based Neural Networks against Adversarial Attacks Develop and implement strategies to enhance the resilience of prototype-based neural networks against adversarial attacks while preserving their interpretability. This involves reviewing existing defense mechanisms and proposing tailored modifications to ensure enhanced robustness without compromising the clarity of the explanations.

Assess the Effectiveness and Impact of Defensive Strategies Assess the effectiveness of the proposed defensive strategies in improving the robustness of prototype-based neural networks by evaluating their performance in both adversarial and non-adversarial scenarios. This assessment will include examining any potential trade-offs between enhanced robustness and interpretability, ensuring that the models continue to provide clear and reliable explanations despite the applied defenses.

2.2 Planification

This section outlines the plan for executing the project, divided into several phases, each with specific tasks and estimated time allocations. Figure 2.1 shows a Gantt chart that visually represents the timeline for each phase, while Figure 2.2, shows the estimated hours of dedication for each phase.

By following this plan, the project aims to achieve its objectives efficiently, contributing valuable insights into the robustness of self-explainable deep learning models against adversarial attacks.

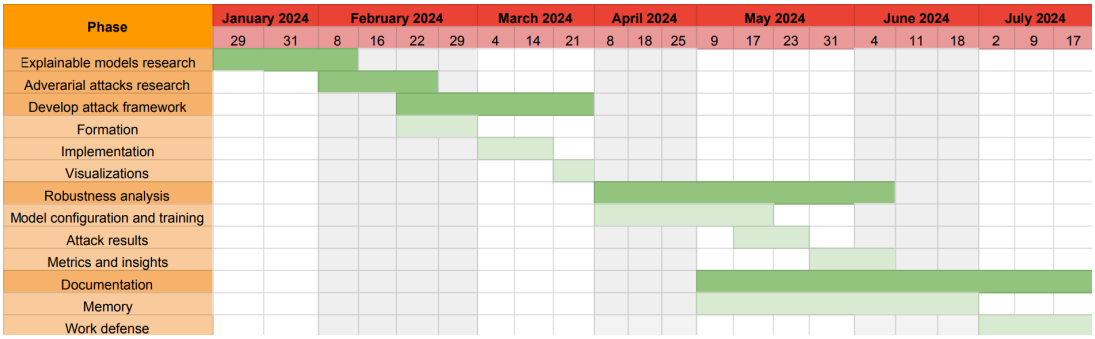


Figure 2.1: Gantt diagram displaying the estimated time allocation for the various phases of the project.

Phase	Estimated dedication (hours)
Explainable models research	30
Adverarial attacks research	30
Develop attack framework	50
Formation	20
Implementation	20
Model visualization	10
Robustness analysis	120
Model configuration and training	70
Attack results	30
Metrics and insights	30
Documentation	80
Memory	60
Work defense	20
Total	310

Figure 2.2: Estimated hours of dedication for each phase of the project.

CHAPTER 3

Background

3.1 Deep Learning

DL, a subset of Machine Learning (ML), has emerged as a powerful tool in solving complex problems across various domains. Its inception draws inspiration from the biological neural networks of the human brain, attempting to mimic their functionality to process vast amounts of data and make sense of it. Much like the brain's neurons communicate through synapses, DL models utilize interconnected layers of artificial neurons, or nodes, to interpret and learn from data.

DNNs lie at the heart of DL, representing intricate computational architectures composed of layers of interconnected nodes arranged in a hierarchical manner. These nodes, also termed neurons, operate by applying simple mathematical transformations to incoming data and transmitting the processed information forward to subsequent layers. The activation function, a crucial component within each neuron, determines whether and to what extent the neuron should fire based on the input it receives. Much like the brain's plasticity, which adapts when learning, DNNs also demonstrate computational adaptability. With exposure to large datasets, DNNs adjust their internal parameters to capture underlying patterns, refining representations and boosting predictive abilities. However, DNNs struggle with handling data that is out of the training distribution, where their performance can significantly degrade. This limitation poses challenges for applications requiring robust generalization to new, unseen environments.

One of the remarkable aspects of DL is its versatility, with various architectures tailored to different tasks. Just as different types of cells in the human body serve specific functions, different architectures like Multilayer Perceptrons and Convolutional Neural Networks excel in different domains. These architectures will be presented in detail in the following sections.

3.1.1 Multi-layer Perceptrons

A MLP is a feedforward artificial neural network (ANN) model that consists of multiple layers of neurons, organized in a sequence of interconnected layers. Each layer, except the input layer, is associated with a set of learnable parameters, including weights ($W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$) and biases ($b^{(l)} \in \mathbb{R}^{n^{(l)}}$), which are adjusted during the training process to optimize the network's performance.

3.1.1.1 Structure

An MLP typically comprises three types of layers:

1. **Input Layer ($l = 0$):** This layer consists of neurons representing the input features of the data. Each neuron corresponds to one feature, and the number of neurons in this layer depends on the dimensionality of the input data.

2. **Hidden Layers ($l = 1, 2, ..., L-1$):** These are one or more layers located between the input and output layers. Each neuron in a hidden layer receives input from the neurons in the previous layer and produces an output that serves as input to the subsequent layer. The number of hidden layers and neurons in each layer is configurable and depends on the complexity of the problem.

3. **Output Layer ($l = L$):**This layer produces the final output of the network. The number of neurons in this layer depends on the nature of the task. For instance, for binary classification, a single neuron with a sigmoid activation function may be used, while for multi-class classification, multiple neurons with softmax activation are employed.

3.1.1.2 Operation

Let $W^{(l)}$ be the weight matrix, $b^{(l)}$ be the bias vector, and $a^{(l)}$ be the activation vector of layer l. The input layer serves to propagate the raw input x features through the network so $a^{(0)} = x$. For subsequent layers ($l = 1, 2, ..., L$), the operation of an MLP can be summarized as follows:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)} \tag{3.1}$$

$$a^{(l)} = \sigma(z^{(l)}) \tag{3.2}$$

where $\sigma(\cdot)$ is the activation function applied element-wise to the linear combination $z^{(l)}$ of the inputs. Activation functions introduce non-linearity to the network, allowing it to learn complex patterns in the data. Without activation functions, the network would reduce to a linear transformation, severely limiting its expressive power. Additionally, a bias term $b^{(l)}$ is added to the weighted sum of inputs in each layer. This bias term effectively shifts the activation function, enabling the network to better fit the training data and improve its performance. For the output layer in multi-class classification tasks, the softmax activation

function is commonly used. It converts the raw output scores into probabilities by applying the following formula:

$$s_k^{(L)} = \text{softmax}(z^{(L)})_k = \frac{e^{z_k^{(L)}}}{\sum_{k'=1}^{K} e^{z_k^{(L)}}} \tag{3.3}$$

where j indexes the neurons in the output layer, and K is the total number of neurons in the output layer. Softmax is used in multi-class classification tasks because it ensures that the output probabilities sum up to 1, making them interpretable as class probabilities. This property is essential for determining the most likely class prediction.

After applying the softmax function, the class with the highest probability is chosen as the predicted class. In other words, the output neuron with the highest probability represents the predicted class label. Mathematically, the predicted class label $\hat{y}$ can be obtained as:

$$\hat{y} = \underset{k \in K}{\arg\max}\ s_k^{(L)} \tag{3.4}$$

3.1.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a specialized type of ANN designed to process and analyze data with a grid-like topology, such as images. CNNs are particularly effective for tasks involving visual data due to their ability to capture spatial hierarchies and patterns through the use of convolutional layers.

3.1.2.1 Structure

A CNN typically comprises several types of layers:

1. **Input Layer:** This layer holds the raw input data, such as an image. Each input neuron corresponds to one pixel of the image, and the number of neurons depends on the image's dimensions (height, width, and number of channels).

2. **Convolutional Layers:** These layers apply convolution operations to the input data using learnable filters (kernels) that slide over the input's spatial dimensions. Each filter produces a feature map that highlights specific patterns or features in the input data. The parameters of the convolutional layers include the size of the filters, the stride (step size of the filter movement), and the padding (zero-padding around the input's borders).

3. **Activation Layers:** After each convolutional layer, an activation function is applied to introduce non-linearity into the model. This helps the network learn more complex patterns.

4. **Pooling Layers:** These layers downsample the feature maps, reducing their spatial dimensions while retaining important features. Common pooling operations include

max pooling and average pooling. Pooling helps to make the network more invariant to small translations and reduces the computational load.

5. **Fully Connected Layers:** Towards the end of the network, one or more fully connected layers are used, which can be seen as a MLP (see 3.1.1). Fully connected layers are responsible for combining features learned by convolutional layers to make predictions.

6. **Output Layer:** This layer produces the final output, similar to the output layer in the MLP hidden and output layer. It typically uses a softmax activation function (see Equation 3.3) in classification tasks to generate class probabilities, with the number of neurons corresponding to the number of target classes K.

3.1.2.2 Advantages and Applications

CNNs offer several key advantages:

- **Spatial Invariance:** CNNs can detect features regardless of their spatial position in the input, making them robust to translations and distortions in the data.
- **Hierarchical Feature Learning:** By stacking multiple convolutional and pooling layers, CNNs learn hierarchical representations, from simple edges and textures in early layers to complex patterns and object parts in deeper layers.
- **Parameter Sharing:** Convolutional layers share parameters across spatial dimensions, significantly reducing the number of parameters compared to fully connected layers and improving computational efficiency.

3.2 Adversarial Attacks

The remarkable success of DL systems in various domains is attributed to their ability to learn complex patterns from data and make accurate predictions. However, this proliferation of DL models has also attracted the attention of adversaries seeking to exploit vulnerabilities for malicious purposes.

Adversarial attacks [1], a class of attacks specifically crafted to deceive ML models, have emerged as a formidable threat to the reliability and security of these systems. By introducing imperceptible perturbations to input data, adversaries can manipulate ML models into making erroneous predictions, thereby compromising their functionality and undermining their trustworthiness.

3.2.1 Taxonomy of Adversarial Attacks

This section delves into the nuances of adversarial attacks, exploring factors such as the level of adversary information, the objectives driving the attacks, the frequency of attack iterations, and the overview of perturbation strategies [2].

- **Adversary Information** The nature of attacks varies based on the level of insight the adversary possesses regarding the network. There are two main categories:
  - *White-box attacks*: The attacker has complete access to training data, architecture, weights, and all layer outputs of the model.
  - *Black-box attacks*: The attacker is restricted to accessing only the output layer of the model. However, the frontier of what constitutes a black-box attack can be diffuse, as attackers may leverage partial knowledge or other indirect information about the model.
- **Adversarial Objective** The objective of attacks hinges on the desired misclassification by the adversary. Two primary objectives are recognized:
  - *Untargeted attacks*: Aim to introduce a perturbation η that alters the model's classification from the original, such that $f(x + \eta) \neq f(x)$.
  - *Targeted attacks*: Seek to induce a specific misclassification, directing the model to predict a designated target class other than the correct one, thus leading to $y_t \neq f(x)$ and consequently $f(x + \eta) = y_t$.
- **Attack Frequency** Attacks differ based on the number of iterations required to transform the original input x into an adversarial example. Two principal types are discerned:
  - *One-time attacks*: Completed in a single iteration to achieve the desired perturbation.
  - *Iterative attacks*: Necessitate multiple iterations to generate the desired perturbation. While iterative approaches often yield superior adversarial examples, they come at the expense of heightened computational demands.
- **Perturbation Overview** The application of perturbations varies depending on whether the same perturbation is applied to multiple examples or tailored individually. Two primary types are identified:
  - *Individual perturbations*: Distinct perturbations are crafted for each example.
  - *Universal perturbations*: A single perturbation is crafted to extend across multiple examples within the same dataset. Universal perturbations offer utility in scenarios where generating individual perturbations for each example proves computationally prohibitive.

3.2.2 Popular Adversarial Attack Methods

In 2014, Szegedy et al. [1] introduced adversarial examples and proposed a optimization problem such that, given the input $x \in X$ and the target class $y_t \in Y$, the perturbation η is

defined as follows:

$$\begin{aligned} &\text{minimize } ||\eta||_2 \text{ subject to:} \\ &1.\ f(x+\eta) = y_t \qquad (3.5) \\ &2.\ x+\eta \in X \qquad (3.6) \end{aligned}$$

This problem was solved using L-BFGS, which was a expensive linear search method. In the following years, better and more efficient methods emerged. In what follows, we introduce the most influential attacks.

3.2.3 Fast Gradient Sign Method

This is a one-step white-box attack proposed by Goodfellow et al. [3], which efficiently computes adversarial perturbations using first order information. The attack can be seen as linearizing the loss function $L(\theta, x, y)$ and generating a perturbation η following the direction of the gradient of the loss function $\nabla_x L(\theta, x, y)$ with respect to the input x, denoted as $\text{sign}(\nabla_x L(\theta, x, y))$. The perturbation η and the corresponding adversarial example x' are obtained as follows:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x L(\theta, x, y)) \qquad (3.7)$$

$$x' = x + \eta \qquad (3.8)$$

Here ϵ bounds the l_∞ norm of the perturbation, y is the ground truth label for x and θ represents the parameters of the model. If a target class y_t wants to be targeted, the same idea can be followed, but in this case the input x is perturbed in the opposite direction of the gradients, $-\text{sign}(\nabla_x L(\theta, x, y_t))$. Please note that in order to obtain different adversarial examples from an example x, a initial perturbation can be applied to x before computing η. See Figure 3.1 for visual interpretation of the FGSM attack.

3.2.4 Projected Gradient Descent

This is a iterative white-box attack proposed by Madry et al. and it is considered to be the strongest adversarial attack using first order information about the network [4]. It can be seen as a multi-step version of the FGSM attack:

$$x'_{t+1} = \Pi_{x'_t+\eta}(x'_t + \alpha \cdot \text{sign}(\nabla_x L(\theta, x'_t, y))), \qquad x'_0 = x \qquad (3.9)$$

here x'_t is the adversarial example at the t-th iteration and α sets the step size of the perturbation. The projection operation Π over the perturbed example $x'_t + \eta$ ensures that $||x' - x||_\infty \leq \epsilon$ in case of using l_∞ norm or $||x' - x||_2 \leq \epsilon$ in case of using l_2 norm. Please note that in order to obtain different adversarial examples from an example x, a initial perturbation can be applied to x'_0. See Figure 3.2 for a visual interpretation of the PDG method.

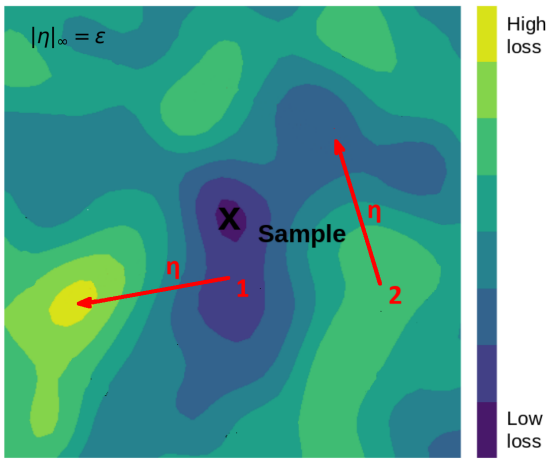


Figure 3.1: Visualization of the FGSM attack. The contour plot represents the loss landscape, with the black 'X' marking the original sample position. The red arrows indicate the direction of perturbation η in two different directions, where $\|\eta\|_\infty = \epsilon$. Points 1 and 2 represent different starting points due to random starts in the FGSM attack. The color bar on the right shows the loss values, with yellow indicating high loss and purple indicating low loss.

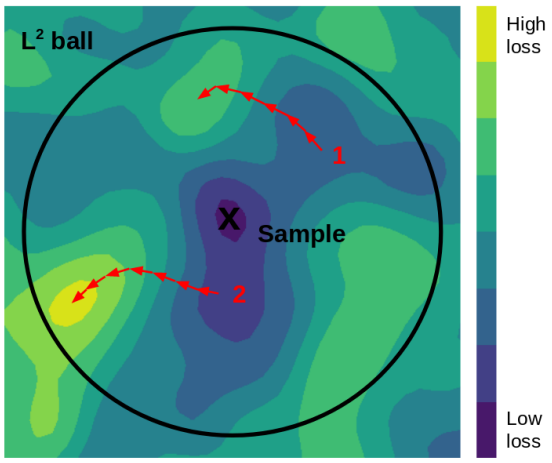


Figure 3.2: Visualization of the PGD L^2 attack. The contour plot represents the loss landscape, with the black 'X' marking the original sample position. The black circle indicates the L^2 ball. The red arrows indicate the direction of perturbation η in two different directions, where $\|\eta\|_\infty = \epsilon$. Points 1 and 2 represent different starting points due to random starts in the attack. The color bar on the right shows the loss values, with yellow indicating high loss and purple indicating low loss.

3.3 Explainable Models

Neural network interpretability is a critical area of research aimed at making the decision-making processes of DL models transparent and understandable. Various methods have been developed to achieve interpretability, each with distinct approaches and benefits. Below, we describe four primary explanation methods: explanations by example, attribution, hidden semantics and logic rules. For a comprehensive overview of these and other methods, refer to the survey by Zhang et al. [5].

Explanations by Example Example-based explanations provide insight by comparing new inputs to previously seen examples, explaining predictions based on similarities. In recent years, prototype-based methods have become popular for their clear and interpretable outputs, using representative examples (prototypes) to clarify model decisions. Several innovative models have implemented prototype-based explanations, each enhancing interpretability in unique ways.

Li et al. [6] combine deep learning with case-based reasoning, learning prototypes for classification. They explain predictions by comparing new inputs to these prototypes, typically requiring a decoder to visualize them. This model will be used in our experimentation and detailed in Chapter 4.

Alvarez Melis and Jaakkola [7] extend this idea with self-explaining neural networks (SENNs), which use abstract and interpretable basis concepts instead of direct prototypes. SENN integrates interpretability into the model architecture, using relevance scores and representing each concept through data elements that most activate them or synthetic inputs that isolate concept activation. This often avoids the need for a separate decoder (see Figure 3.3).

Figure 3.3: The input images (left) are explained using self-explaining neural networks (SENN) by identifying and scoring relevant concepts (middle). The explanation for SENN includes a characterization of concepts in terms of defining prototypes (right) [7].

Chen et al. [8] introduce ProtoPNet, which uses part-based visual reasoning. Unlike other models that use whole examples, ProtoPNet compares parts of an image to learned prototypes,

providing a more detailed explanation of the decisions. This approach shows which image parts contribute to classifications without needing a decoder (see Figure 3.4).

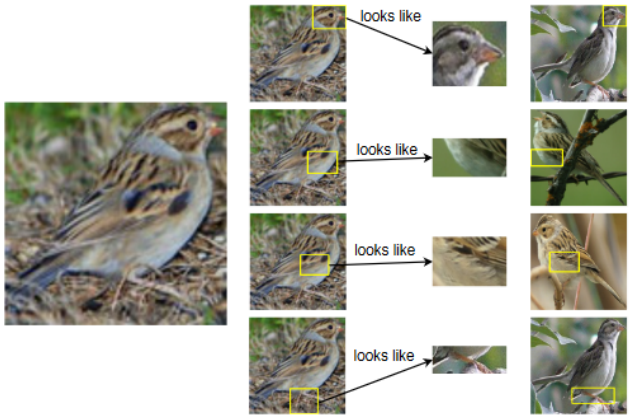


Figure 3.4: Image of a clay-colored sparrow and how parts of it look like some learned prototypical parts of a clay-colored sparrow used to classify the bird's species. This illustrates ProtoPNet's part-based visual reasoning approach [8].

Explanations with Attribution Attribution methods focus on determining the importance of each input feature in the model's predictions. These methods [9] [10] [11] [12], such as gradient-based approaches, generate saliency maps highlighting which parts of an input (e.g., regions in an image) are most influential in the decision-making process. Attribution techniques can be local, explaining individual predictions, or global, aggregating feature importances across many inputs to provide an overall understanding of the model's behavior (see Figure 3.5).

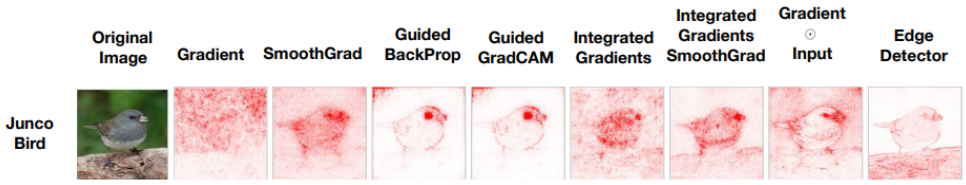


Figure 3.5: This figure shows the original image of a Junco Bird and the corresponding saliency maps generated by various methods. These saliency maps illustrate how different techniques highlight important features in the image, helping to understand which parts of the image contribute most to the model's prediction [10].

Explanations with Hidden Semantics Hidden semantics explanations seek to interpret the meanings of internal neurons or layers in neural networks. This methods [13] [14] associate abstract concepts with specific activations in the network, akin to the "grandmother cell" hypothesis in neuroscience. Visualization techniques, like activation maximization, are used to identify patterns that neurons respond to, providing insights into the network's internal representations and decision-making process (see Figure 3.6).

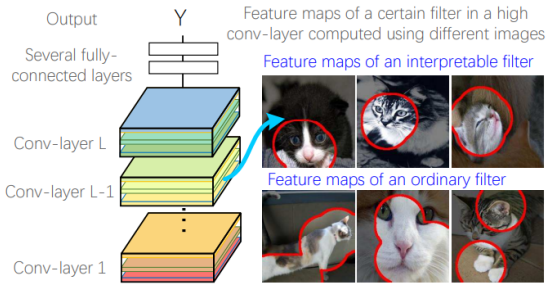


Figure 3.6: This figure illustrates the feature maps of a certain filter in a high convolutional layer computed using different images. The highlighted regions in red circles show how specific filters in interpretable convolutional neural networks (CNNs) can focus on distinct parts of objects, such as the faces of animals. This targeted focus on object parts demonstrates the improved interpretability of these CNNs compared to traditional ones, where high-layer filters may describe a mixture of patterns [14].

Explanations with Logic Rules Logic rule-based explanations aim to translate the decision-making process into understandable "if-then" rules. These methods [15] [16] focus on converting complex network predictions into simpler, human-readable rules. This approach helps users understand the conditions under which certain predictions are made, thus enhancing transparency and trust in the model (see Table 3.1).

If	**Predict**
FICO score ≤ 649	Bad Loan
$649 \leq$ FICO score ≤ 699 and \$5,400 $\leq$ loan amount $\leq$ \$10,000	Good Loan

Table 3.1: The table shows conditions for predicting whether a loan on the Lending Club website will turn out bad or good based on FICO score and loan amount [15].

CHAPTER 4

Self-explainable Architectures and Functionalities

This chapter provides a comprehensive explanation of the model utilized in our experimentation. It begins with a detailed description of the model's architecture and operation, including how predictions are generated and interpreted. Following this, it elaborates on various loss functions employed to train the model effectively. The specific implementation can be found in the repository [1] accompanying this thesis.

4.1 Introduction

The model presented in this chapter is a prototype-based neural network that integrates case-based reasoning for enhanced interpretability, as detailed in *Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions* by Li et al. [6]. This model employs an autoencoder to reduce input dimensionality and extract essential features, which are then compared to learned prototype vectors in the latent space. The classification process is informed by these comparisons, based on squared L^2 distances, which provide clear and visualizable explanations for the model's predictions. By training prototypes to be similar to actual training examples, the model ensures a intuitive interpretation for its predictions.

4.2 Model Architecture

Let us assume a classification problem with a training set $D = \{(x_i, y_i)\}_{i=1}^{n}$, where given the features $x_i \in \mathbb{R}^p$ the label $y_i \in \{1, ..., K\}$ must be predicted.

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

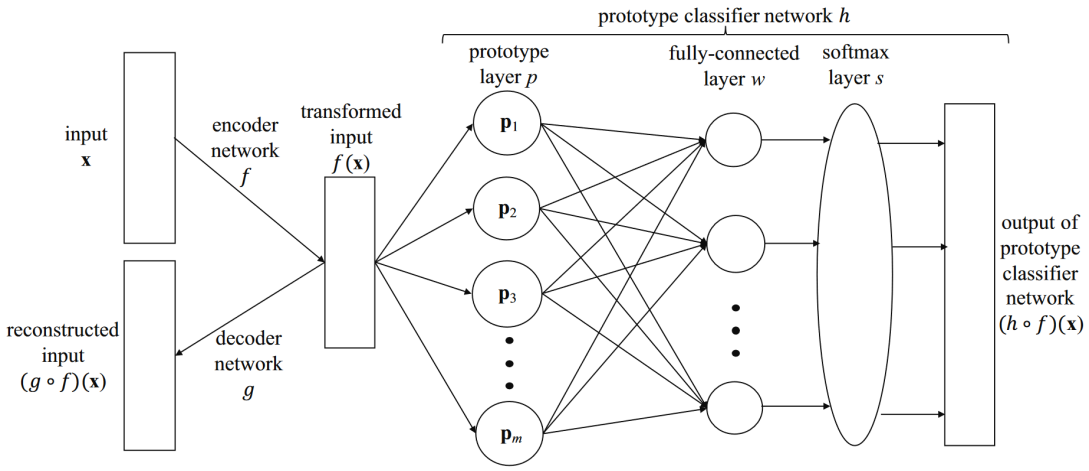


Figure 4.1: Model architecture [6]

The first layer of the model consists of an autoencoder formed by a encoder $f : \mathbb{R}^p \to \mathbb{R}^q$ and a decoder $g : \mathbb{R}^q \to \mathbb{R}^p$. The input x_i is passed through the encoder, which reduces the dimensionality of the input while preserving the useful features needed for the prediction.

Once the input is encoded $f(x_i) \in \mathbb{R}^q$ it is forwarded through the prototype classification network h. This network consists of a prototype layer $p : \mathbb{R}^q \to \mathbb{R}^m$, a fully connected layer $w : \mathbb{R}^m \to \mathbb{R}^K$ and a softmax layer, $s : \mathbb{R}^K \to \mathbb{R}^K$. During the training phase, the network learns m prototype vectors $p_1, ..., p_m \in \mathbb{R}^q$. Since the prototype vectors are in the same latent space as the encoded input $z = f(x_i) \in \mathbb{R}^q$, they can be fed through the decoder $p_{image} = g(p_j)$ and be visualized in the input space $\mathbb{R}^q$. The prototype layer p computes the squared L^2 distance between each prototype p_j and z:

$$p(z) = \left[\|z - p_1\|_2^2, \|z - p_2\|_2^2, \ldots, \|z - p_m\|_2^2\right]^\top \tag{4.1}$$

In Figure 4.1, each unit $p_j \in \{p_1, \ldots, p_m\}$ in the prototype layer p corresponds to the distance $\|z - p_j\|_2^2$. One could think of eliminating the autoencoder in order to have directly interpretable prototypes without the need of the decoder and therefore obtain the distances with respect to the raw input x_i. However, in the case of image classification, where the number of features x_i is equal to the number of pixels, traditional distance metrics do not work correctly, leading to poor results.

After the column of distances $p(z)$ has been computed, it is passed through the fully-connected layer w, which consists of a weight matrix W of dimension $K \times m$, such that $v = Wp(z)$ is computed. Here v is a K-dimensional vector, where v_i is the weighted sum of distances for class $i \in \{1, ..., K\}$. The strengths of the learned weight connections reveal which prototypes are more representative of each class, thereby helping us understand the model's decision-making process. A detailed interpretation of the weight matrix W is provided in Chapter 6.

Finally, the weighted sums in v are normalized by a softmax layer s in order to get a probability distribution over the K classes:

$$s(v)_k = \frac{e^{v_k}}{\sum_{k'=1}^{K} e^{v_{k'}}}, k = 1, \ldots, K \tag{4.2}$$

where $s(v)_k$ is the probability that the input x_i belongs to the class $k = 1, \ldots, K$ and v_k is the k-th component of the vector v. In general, the class with the highest probability is selected as the output class, which is $\hat{y} = \arg\max_{k \in K} s(v)_k$.

The visualization in Figure 4.2 illustrates the prediction process of the model, trained over the MNIST dataset [17]. The model in this case has learned $m = 15$ prototypes. Notably, the closest prototypes resemble the digit '1', just like the input image, which indicates that the model correctly identifies and classifies the digit by associating it with similar learned prototypes. This visualization demonstrates how the model utilizes these distances to make its classification decision, highlighting the interpretability of the prediction process through comparison with learned prototypes.

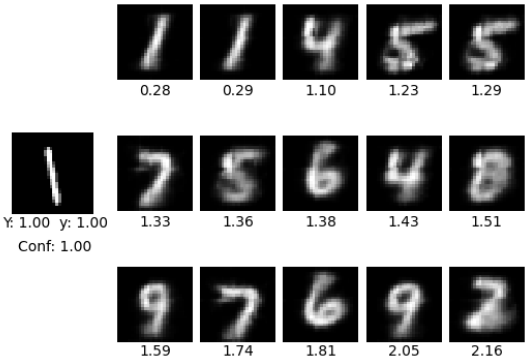


Figure 4.2: Visualization of the prediction result. The leftmost image is the input digit with its true label (Y: 1.00), predicted label (y: 1.00), and confidence score (Conf: 1.00). The subsequent images represent the decoded prototype vectors in the pixel space along with their computed (rounded) squared L^2 distances to the encoded input digit.

4.3 Cost Function

The cost function used to train the model focuses on obtaining good accuracy while maintaining interpretable prototype-based explanations. The cost function for training the model is composed of four terms, which are described below.

Cross-Entropy Classification Loss (E) The standard cross-entropy loss is used for penalizing the misclassifications of the model, which is given by the following formula:

$$E(h \circ f, x, y) = \sum_{k=1}^{K} -\mathbb{1}[y = k] \log((h \circ f)_k(x)), \tag{4.3}$$

where $(h \circ f)_k$ is the k-th component of $(h \circ f)(x)$.

Autoencoder Loss (A) The autoencoder loss measures the difference between the original and reconstructed input from the latent space. In this case, the l_2 norm is used to measure the difference, so the autoencoder loss is given by:

$$A(g \circ f, x) = \|(g \circ f)(x) - x\|_2^2, \tag{4.4}$$

where $(g \circ f)(x)$ is the reconstruction of the input x.

First Interpretability Loss (R1) The loss acts as a regularizer, forcing the prototype vectors to decode to images that are similar to the input ones. This can be interpreted as requiring each prototype vector to be as close as possible to at least one training example, which is mathematically defined as:

$$R_1(p_1, \ldots, p_m, D = \{x_i, y_i\}_{i=1}^n) = \frac{1}{m} \sum_{j=1}^{m} \min_{i \in [1,n]} \|p_j - f(x_i)\|_2^2, \tag{4.5}$$

where $\min_{i \in [1,n]} \|p_j - f(x_i)\|_2^2$ represents the distance for the prototype vector j to his closest encoded example $f(x_i)$, $x_i \in D$. Minimizing this loss forces the prototype vectors to have meaningful decodings in the input space. Note that $R1$ ideally computes the distances from every prototype to each of the examples in the dataset D, which grow linearly with the number of examples n. However, for large datasets this is not feasible, so the loss is computed only over the random minibatch used by the Stochastic Gradient Descent (SGD) algorithm.

Second Interpretability Loss (R2) This loss also acts as a regularizer, forcing the training examples to be as close as possible to one of the prototype vectors. This can be seen as clustering the examples around prototypes in the latent space, which is mathematically defined as:

$$R_2(p_1, \ldots, p_m, x) = \min_{j \in [1,m]} \|f(x) - p_j\|_2^2, \tag{4.6}$$

where $\min_j \in [1, m] \|f(x) - p_j\|_2^2$ is the distance from the encoded example x to it's closest prototype $p_j \in \{p_1, \ldots, p_m\}$.

Combined Loss (L) Once each loss term is explained, all the terms are added into a more general loss function over which the model can be trained given a training dataset $D = \{x_i, y_i\}_{i=1}^n$:

$$\begin{aligned} L((f, g, h), x, y, D) \quad = \quad & \lambda_e E(h \circ f, x, y) + \lambda_a A(g \circ f, x) \\ & + \lambda_1 R_1(p_1, \ldots, p_m, D) \\ & + \lambda_2 R_2(p_1, \ldots, p_m, x), \end{aligned} \tag{4.7}$$

note that λ_e, λ_a, λ_1 and λ_2 are hyperparameters in $\mathbb{R}$, which are used to adjust the contribution of each loss term. See Figure 4.3 for a ilustration of the effect of the R1 and R2 losses over the prototype vectors.

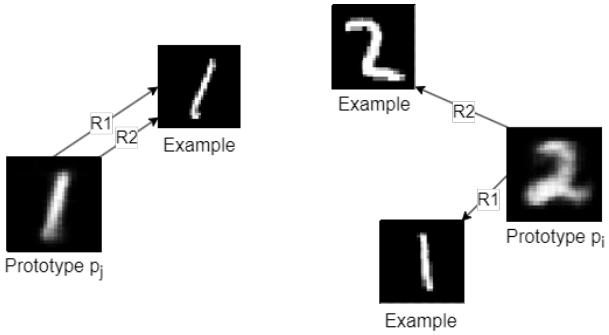


Figure 4.3: Illustration of R1 (see Equation 4.5) and R2 (see Equation 4.6) interpretability losses effect. The arrows indicate the sense(s) where the prototype vectors are moving due to the influence of the losses. The numbers indicate the corresponding label for each prototype and example. While the prototype p_j is moving towards a example of his corresponding class, the prototype p_i is also moving towards his closest example, which does not belong to the class of the prototype. This is the behavior we try to void using the AL 4.10 loss function.

4.4 Alternative Cost Function

Using the previous cost function, users have no control over the number of prototypes learned by class. In the work of Chen et al. [8], a predetermined number of prototypes t is allocated for each class $k \in \{1 \dots K\}$. By doing this, they ensure that every class is represented by t prototypes, such that the total number of prototypes is $m = K \cdot t$.

Furthermore, they propose fixing the weight matrix W in the fully-connected layer w. Let P_k be the subset of prototypes that represent the class k, so $p_j \in P_k$ means that the prototype vector p_j represents class k and similarly, $p_j \notin P_{k'}$ means that it does not represent the class k'. The weight matrix is filled such that $W_{kj} = -1 \quad \forall p_j \in P_k$ and $W_{kj} = 0.5 \quad \forall p_j \notin P_k$, where $k = 1 \dots K$ and $j = 1 \dots m$. By fixing this values, they enforce that the prototypes represent characteristics that are relevant for class k but not for other classes. The negative connection between prototypes $p_j \in P_k$ and class k means that small distances to this prototypes should increase the probability that the input belongs to class k and diminish the probability that belongs to the other classes.

However, after learning relevant prototypes, they perform a convex optimization on the weight matrix W so that for k and j with $p_j \notin P_k$ the final model fulfills that $W_{kj} \approx 0$. This particular fixed values for the fully-connected layer prevents relying on a negative reasoning process where an example belongs to class k because it does not to class $k' \neq k$. This fixing leads to a distance based explanation just by the example similarity to each class prototypes, removing the weighted distances sum, which can be less intuitive.

In our particular case, we decided to directly fix the values to 0 to prevent having a second optimization process. This approach simplifies the training procedure and ensures that prototypes only contribute positively to the class they represent, without the need for additional optimization to adjust the weight matrix. Fixing the fully-connected layer may not be sufficient to learn t interpretable prototypes for each class k, so the *clustering* loss is

introduced:

Clustering Loss (CLS) The clustering loss encourages each training example to be close to at least one prototype of its own class, which can be mathematically defined as follows:

$$CLS(p_1, \ldots, p_m, x, y) = \min_{p_j \in P_y} \|f(x) - p_j\|_2^2, \tag{4.8}$$

where $p_j \in P_y$ refers to the set of prototypes that represent the class y of the example x.

We also propose a modification to the original First Interpretability Loss (R1) (see Equation 4.5). The prototypes representing the class k should be as close as possible to one of the examples from that class, by doing this we ensure that $p_j \in P_k$ have meaningful decodings of class k in the input space. This can be mathematically defined as:

$$S_1(p_1, \ldots, p_m, D = \{x_i, y_i\}_{i=1}^n\}) = \frac{1}{m} \sum_{j=1}^{m} \min_{p_j \in P_k \wedge y_i = k} \|p_j - f(x_i)\|_2^2, \tag{4.9}$$

Alternative Loss (AL) Once the new loss terms are explained, they are added into a more general loss function over which the model can be trained given a training dataset $D = \{x_i, y_i\}_{i=1}^n$:

$$\begin{aligned} AL((f, g, h), x, y, D) \quad = \quad & \lambda_e E(h \circ f, x, y) + \lambda_a A(g \circ f, x) \\ & + \lambda_1 S_1(p_1, \ldots, p_m, D) + \lambda_{cls} CLS(p_1, \ldots, p_m, x, y), \end{aligned}$$

note that E is the Classification Loss (see Equation 4.3) and A is the Autoencoder Loss (see Equation 4.4). The S_1 and CLS losses are the ones defined above. Note that λ_e, λ_a, λ_1 and λ_{cls} are hyperparameters in $\mathbb{R}$, which are used to adjust the contribution of each loss term. See Figure 4.4 for a ilustration of the effect of the CLS and S1 losses over the prototype vectors.

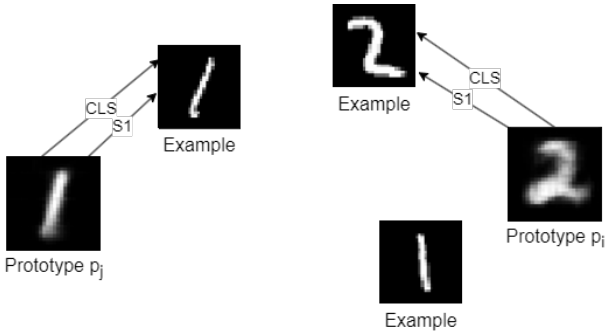


Figure 4.4: Illustration of the CLS (see Equation 4.8) and S1 (see Equation 4.9) losses effect. The arrows indicate the sense where the prototype vectors are moving due to the losses influence. The numbers indicate the corresponding label for each prototype and examples. In this case, unlike with the R1 (see Equation 4.5) and R2 (see Equation 4.6) Losses (see Figure 4.3), both prototypes p_j and p_i are getting closer to the examples of their class because of CLS and S1. This behavior is desirable and more logical, enabling users to specify the desired class k for a prototype p_j and ensure that it is going to be close to examples for its corresponding class k.

CHAPTER 5

Adversarial Attacks Against Prototype-based Neural Networks

In this chapter, a custom-designed framework for generating adversarial attacks for the model described in Chapter 4 is introduced. It is important to recall that these models make predictions considering the distances in the latent space from the inputs to the prototypes, which enables users to interpret the models' classification decisions. In other words, the models provide both the prediction made and the distances to the prototypes as an explanation. This characteristic offers ample opportunities for proposing various adversarial attacks, where both the output and the explanation provided by the model can be manipulated in multiple ways. Below, we enumerate the main attack paradigms we have identified:

- **Change class untargeted (CU):** The adversarial attack attempts to change the class of the examples without targeting any specific class.
- **Change class targeted (CT):** The class of each example is tried to be changed to a selected target class.
- **Change explanation untargeted (EU):** The adversarial attack aims to change the explanation provided by the model, without targeting any specific class explanation.
- **Change explanation targeted (ET):** The attack attempts to alter the explanation provided by the model to a target class explanation.
- **Change explanation and class both untargeted (ECU):** Combines the objectives of changing both the class and the explanation of the examples, without targeting specific class or explanation.
- **Change explanation and class same target (ECST):** The attack targets both a class and the corresponding explanation for that class.
- **Change explanation and class different target (ECDT):** The adversarial attack targets a class and the explanation corresponding to a different class.

Each attack paradigm has its own unique characteristics and challenges, making it essential to develop a unified approach that can accommodate all these different types of attacks. Crafting adversarial attacks is not a trivial task, especially when considering the need to manipulate both the model's output and the explanations it provides. The complexity lies in ensuring that the adversarial examples effectively achieve their intended goals, whether it be changing the predicted class, altering the explanation, or both. See Figure 5.1 for a visual representation of some successful attacks on a model trained on the MNIST dataset [17].

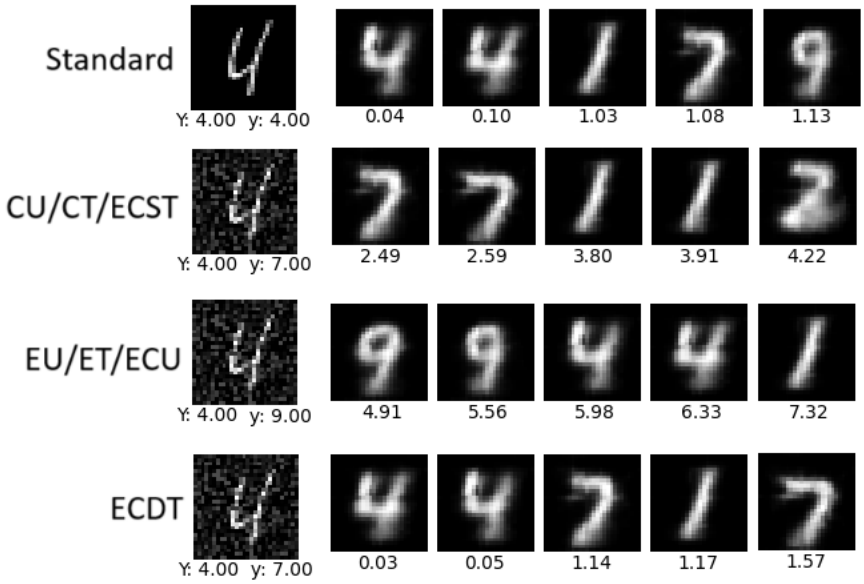


Figure 5.1: Visual examples of successful adversarial attacks on a model trained on the MNIST dataset. Each row corresponds to a different attack paradigm: non-adversarial classification (Standard), CU/CT/ECST, EU/ET/ECU, and ECDT. The results show that attacks can achieve multiple objectives, but for simplicity, they are condensed into representative examples. The leftmost image in each row is the input image, with Y representing the real class of the example and y representing the predicted class. The five images on the right in each row are the closest prototypes in each case, with the values below indicating the distances to them in the latent space.

Given these intricacies, it is crucial to design a versatile and robust loss function that can handle various attack objectives seamlessly. The solution we propose addresses this need by offering a single, flexible framework that can be adapted to generate any of the identified adversarial attack types through specific configuration settings. The following section delves into the specifics of the proposed loss function used to generate these adversarial attacks.

5.1 Proposed Loss Function

As explained in Chapter 3, adversarial attacks aim to deceive the model. Designing a versatile loss function to facilitate multiple adversarial attacks is a complex task due to the need to manipulate both the model's output and the explanations it provides. In this section, we

introduce a robust and flexible loss function that achieves this goal, accommodating various attack paradigms through specific configurations.

Our optimization scheme is based on maximizing the adversarial loss, which comprises two terms: the classification loss and the explanation loss. By carefully configuring these terms, we can generate adversarial examples tailored to various attack objectives.

Suppose an adversarial example is to be generated from an input x with label $y \in \{1, \ldots, K\}$. To alter the explanation and/or the prediction, the adversarial loss will comprise two terms.

5.1.1 Classification Loss (C)

The cross-entropy loss E (see Equation 4.3) is used, with a coefficient α_1 which controls wheter the attack is targeted or untargeted, as will be explained below:

$$C(h \circ f, x, y_t, \alpha_1) = \alpha_1 E(h \circ f, x, y_t), \tag{5.1}$$

where $h \circ f$ is the output layer of the network. If a targeted attack toward a class y_t is wanted, then $\alpha_1 = -1$. In this way, the cross-entropy towards the target class y_t is minimized, forcing the model to predict the target class. Conversely, for an untargeted attack, $\alpha_1 = 1$ and $y_t = y$, so that the attack will maximize the cross-entropy loss for the correct label y.

5.1.2 Explanation Loss (EX)

To adversarially manipulate the explanations, a specially designed loss function that considers the architecture of the model is used. The purpose of this loss function is to modify the distances to specific prototypes to achieve a targeted explanation e_k. Essentially, the Explanation Loss quantifies the sum of the distances to the prototypes that we want to either minimize or maximize. The formal definition of the Explanation Loss is as follows:

$$EX(e_k, p, \alpha_2) = \alpha_2 \cdot \frac{1}{m} \sum_{j=1}^{m} l_j \cdot p_j, \tag{5.2}$$

where $p = \left[\|z - p_1\|_2^2, \|z - p_2\|_2^2, \ldots, \|z - p_m\|_2^2\right]$ which represent the L^2 distances from the encoded input $z = f(x)$ to the prototype vectors $p_1, ..., p_m$. The vector $l \in \mathbb{R}^m$ acts as a mask, aiming to consider the distances to be altered depending on the target explanation e_k.

To provide more clarity, we delve into the specifics of how the mask l is defined. If e_k is a specific prototype, indicating that the goal is to alter the distance between the example x and the prototype p_j, then:

$$l_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Another case may be e_k to be all the prototypes that represent class k ($p_i \in P_k$), meaning that the distance to them is wanted to be altered, in that case:

$$l_i = \begin{cases} 1 & \text{if } p_i \in P_k \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Once the definition of the mask l is understood, the EX loss can be applied flexibly. If the parameter $\alpha_2 = 1$, the distances to the corresponding prototypes are maximized by the adversarial attack, and if $\alpha_2 = -1$, the distances are minimized. This dual capability allows for precise control over the generated adversarial examples, enabling us to achieve the desired manipulation of the model's explanations.

5.1.3 General Adversarial Loss (ADVL)

Once each loss term is explained, all the terms are combined into a more general loss function, which can generate a variety of adversarial examples with different objectives, as described before:

$$ADVL(f, h, p, e_t, y_t, \alpha_1, \alpha_2) \quad = \quad C(h \circ f, x, y_t, \alpha_1) + EX(e_k, p, \alpha_2), \tag{5.5}$$

where α_1 or α_2 can also take value 0 in case a term is not taken into account when computing the adversarial loss. As it can be seen in Table 5.1, a detailed ADVL loss configuration is used to generate the desired adversarial examples in each case. The implementation of the ADVL can be found in the repository [1] accompanying this thesis.

Attack	α_1	α_2	y_t	e_k	**Description**
CU	1	0	y	-	Attempts to change the class of the examples without targeting any specific class.
CT	-1	0	$y_t \in \{1, \ldots, k\}$	-	Attempts to change the class of each example to a selected target class.
EU	0	1	-	y	Aims to change the explanation provided by the model without targeting any specific class explanation.
ET	0	-1	-	$e_k \in \{1, \ldots, k\}$	Attempts to alter the explanation provided by the model to a target class explanation.
ECU	1	1	y	y	Combines the objectives of changing both the class and the explanation of the examples without targeting specific class or explanation.
ECST	-1	-1	$y_t \in \{1, \ldots, k\}$	y_t	Targets both a class and the corresponding explanation for that class.
ECDT	-1	-1	$y_t \in \{1, \ldots, k\}$	$e_k \in \{1, \ldots, k\}, e_k \neq y_t$	Targets a class and the explanation corresponding to a different class.

Table 5.1: Specification of the configuration for the ADVL loss (see Equation 5.5) for each of the attacks. Here, y refers to the original class and y_t refers to the target class.

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

CHAPTER 6

Setting a Representative Experimental Framework

In this chapter, we introduce the foundational models and methodologies for our experimentation using the MNIST dataset [17]. We begin by detailing the dataset and preprocessing steps, followed by describing the architecture of a multilayer convolutional autoencoder suggested in the original model paper [6]. We then discuss the training procedures, including data augmentation techniques, and the used configurations. Recognizing limitations in the base model, we propose improvements aimed at enhancing class representation and interpretability. This chapter provides the baseline for further analysis and experimentation in subsequent chapters. All the trained models, visualization and evaluation scripts can be found in the repository [1] accompanying this thesis.

6.1 Dataset of Study

Our experimentation relies on the MNIST dataset [17], a foundational collection of handwritten digits from 0 to 9 that is widely used in machine learning research. The dataset comprises 60.000 training examples and 10.000 testing examples. For our specific analysis, we partitioned the data into 55.000 training examples, 5.000 validation examples, and 10.000 testing examples, each sized at 28×28 pixels. Additionally, we applied preprocessing to normalize every pixel value within the images to the $[0, 1]$ range.

6.2 Architecture Details

For our experimental setup, we adopted the architectural framework proposed by Li et al. [6], featuring a multilayer convolutional autoencoder. This architecture is known for its

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

symmetrical configuration of the encoder f and decoder g, which helps to reduce redundancy in spatial feature extraction and enables the learning of hierarchical features conducive to state-of-the-art classification performance.

The encoder f processes the original image through a series of four convolutional layers. Each convolutional layer includes a convolution operation followed by a pointwise nonlinearity, and downsampling is achieved using strided convolution. Specifically, all convolutional layers in the encoder utilize 3×3 kernels, zero padding, and a stride of 2. The encoder ultimately compresses the input image, reducing its dimensions from $28 \times 28 \times 1$ to a 40-dimensional vector with output shapes of $14 \times 14 \times 32$, $7 \times 7 \times 32$, $4 \times 4 \times 32$, and $2 \times 2 \times 10$.

The code vector from the encoder is forwarded to the prototype layer p. Concurrently, the unflattened feature maps are fed into the decoder g, which mirrors the encoder's structure with four deconvolutional layers using strided deconvolution for upsampling. By maintaining the same configuration in the decoder for deconvolution, the network effectively reconstructs the original image. This consistency in architectural elements across the encoder and decoder ensures both high classification accuracy and interpretability, as the learned features are visualized in the input space.

The activation function employed in each layer is the rectified linear unit (ReLU) function, defined as $\text{ReLU}(x) = \max(0, x)$, which is known for its effectiveness in promoting sparse activation and mitigating the vanishing gradient problem. However, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is specifically selected for the last decoder layer to ensure that the reconstructed output's pixel values fall within the $(0, 1)$ range. This choice aligns the output with the preprocessed image's pixel range, facilitating a more accurate reconstruction.

6.3 Training Details

In this section, we provide a comprehensive overview of the training procedures for the models discussed in this chapter. We detail the data augmentation techniques used to avoid overfitting, the configurations of the various models, the specific training settings and hyperparameters employed. This thorough examination aims to explain the rationale behind our methodological choices and their implications for model interpretability.

All models adhere to the architecture described in Section 6.2, and were trained for 750 epochs using a learning rate of 0.002 with the Adam optimizer. In order to enhance the robustness and generalization of the models, we incorporated elastic deformation as a data augmentation technique, as introduced by Simard et al. [18] and employed in the original architecture training [6]. Elastic deformation applies a broader range of transformations than affine transformations. For each mini-batch of 250 samples, a random elastic distortion is applied using a Gaussian filter with a standard deviation of 4 and a scaling factor of 20 for the displacement field. This integration of elastic deformation introduces randomness into the data augmentation process, ensuring that the neural network encounters a varied set of images during each epoch, significantly reducing overfitting.

6.3.1 Original Model Configuration and Limitations

To establish a foundation for our experiments, we initially focused on replicating the original model configuration proposed by Li et al. [6]. This approach allowed us to create a baseline model, referred to as the Standard Model (S15), which helps to illustrate the basic principles of prototype-based learning. The hypothesis guiding this model is that a limited number of prototypes ($m = 15$) can sufficiently capture the essential features necessary for effective classification and interpretability.

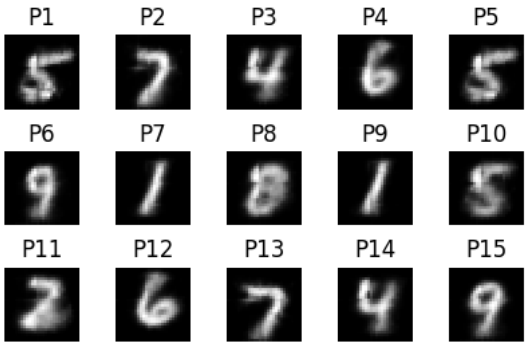
The Standard Model employs the original cost function L (see Equation 4.7) with the original hyperparameter configuration proposed by the authors [2]: $\lambda_a = 1$, $\lambda_1 = 1$, $\lambda_2 = 1$, and $\lambda_e = 20$.

6.3.1.1 Visualization

To assess the effectiveness and interpretability of the S15 model, we conducted a visualization of the learned prototypes and quality of the decoder. This step is crucial to determine whether the prototypes learned by the model capture relevant information about the digits and resemble real handwritten digits, which is fundamental for interpretability.

In Figure 6.1, we present the decoder's reconstruction capacity and the learned prototypes decoded in the pixel space. The decoder accurately reconstructs the inputs without losing relevant information, leading to correct decoding of the prototype vectors into the pixel space. Furthermore, the decoded prototypes resemble real handwritten digits with good quality, demonstrating the model's ability to capture and represent the essential features of the digits.

(a) Standard Model (S15) Reconstructions



(b) Standard Model (S15) Prototypes

Figure 6.1: Visualization of the **Standard Model (S15)**

However, while some digit classes (such as 5) are present in multiple prototypes, digit class 3 is not represented by any of the decoded prototype vectors. This imbalance and lack of class representation highlight a significant limitation in the interpretability of the S15 model. Imagine a situation where an image is classified as a digit 3, but the model's

[2] https://github.com/OscarcarLi/PrototypeDL/tree/master

provided explanation does not include any prototypical digit 3. This inconsistency undermines the model's interpretability, as the essence of this model is to provide clear, representative prototypes for each class.

6.3.1.2 Fully Connected Layer Interpretation

A deeper analysis of the S15 model reveals that the imbalance and lack of class representation can be attributed to the multiplication of the prototype layer's output by a learnable weight matrix W before it is fed into the softmax layer s. This process causes the distances from an encoded image to each prototype to have differing effects on the predicted class probabilities. As a result, the predictions are not directly based on the distances between the encoded input $z = f(x)$ and the prototype vectors $p_1, \ldots, p_{15}$, meaning that these distances influence the predicted probabilities for each class differently.

To clarify this, we analyzed the transposed weight matrix $W^\top$ from the fully-connected layer w, which connects the prototype layer p to the softmax layer s, as shown in Table 6.1. The table reveals the influence of the distance to each prototype on every class. In most cases, the decoded prototype vector p_j resembles an image of a class for which the corresponding entry in the weight matrix has a significantly negative value, conventionally known as the *visual class* of the prototype [6]. However, some interesting cases arise. For instance, prototype P1's visual class is a 5, but it has the most negative value corresponding to class 3. Another interesting case occurs with prototypes P12 and P13, where the most negative value corresponds to class 0, while their visual classes are 6 and 7 respectively. This behavior is not intuitive and therefore makes the model less interpretable.

The significantly negative weight values can be explained as follows: The prototype layer calculates the dissimilarity between an input image x and a prototype p_j by computing the squared L^2 distance between their latent space representations. If an image x does not belong to the visual class of prototype p_j, the distance $\|p_j - f(x)\|_2^2$ will be large. When this large distance is multiplied by the highly negative weight between prototype p_j and its visual class, the product becomes highly negative, significantly reducing the output probability of p_j's visual class. Therefore, image x is unlikely to be classified into p_j's visual class. On the other hand, if x belongs to p_j's visual class, the small distance $\|p_j - f(x)\|_2^2$ multiplied by the highly negative weight will not significantly reduce the activation. Consequently, the activations for all classes that x does not belong to are significantly reduced due to the influence of non-similar prototypes, leaving the activation for x's actual class comparatively high.

6.3.2 Proposed Solutions

To address the significant limitations identified in the S15 model—namely the lack of balanced class representation among prototypes and the influence of the weight matrix on class probabilities—we propose two improved models: the Enhanced Model (S30) and the Balanced Model (B30). The Enhanced Model aims to mitigate the issue of insufficient prototype diversity by increasing the number of prototypes, while the Balanced Model focuses on ensuring equal representation for all classes to resolve the imbalance and enhance interpretability.

	0	1	2	3	4	5	6	7	8	9
P1	1.50	0.31	3.52	-3.99	0.54	-2.56	0.02	0.55	-2.02	0.09
P2	-0.36	-0.54	-0.05	-1.06	0.86	0.05	1.31	-4.77	1.18	0.47
P3	1.14	0.93	-0.76	2.38	-3.90	-0.20	0.30	-0.28	-0.58	0.39
P4	-5.17	-0.21	-2.35	3.25	0.13	0.19	-5.48	0.91	2.13	0.27
P5	1.80	0.29	2.47	-2.87	0.08	-3.66	-0.16	0.67	-0.76	-0.51
P6	-0.09	0.19	0.34	0.63	-0.29	0.99	0.90	0.67	-0.25	-3.28
P7	3.80	-5.59	-0.93	1.47	-0.09	1.57	-0.41	-0.54	-1.56	-0.06
P8	-9.29	0.53	-0.46	2.20	1.45	1.96	-0.31	0.64	-10.05	1.27
P9	4.27	-4.40	-0.67	-0.25	0.36	1.60	-0.04	-0.41	-1.83	0.01
P10	2.01	0.44	2.23	-1.82	0.02	-5.20	0.29	0.64	-0.51	-0.40
P11	0.95	0.23	-9.33	-6.90	1.36	1.48	0.49	0.93	0.24	1.67
P12	-3.88	0.09	-2.34	1.75	-0.13	0.24	-3.60	0.36	3.28	0.45
P13	-5.19	0.58	1.65	-1.09	0.68	-1.04	0.98	-4.42	3.56	0.55
P14	1.61	0.96	-0.17	2.58	-5.15	0.03	0.82	-0.02	-0.42	-1.23
P15	0.21	0.28	0.14	-0.53	-0.07	0.47	0.90	0.26	0.90	-2.82

Table 6.1: The transposed weight matrix for the Standard Model (S15) between the prototype layer and the softmax layer is displayed. Within each row, the most negative weight is emphasized in red, indicating the strongest association between the prototype and a particular digit class. Typically, this negative weight aligns with the visual representation of the corresponding digit class.

6.3.2.1 Enhanced Model (S30)

This model aims to improve class representation by increasing the number of prototypes to $m = 30$. The hypothesis is that a larger set of prototypes would enhance the model's ability to learn at least one representative prototype for each class and enhance the variety and representation of prototypes. The training of this model follows the same training procedure and hyperparameters as the S15 model.

6.3.2.2 Balanced Model (B30)

The Balanced Model aims to address the issue of class representation imbalance that may be present in previous models. The hypothesis for this model is that ensuring an equal representation of all classes by allocating a fixed number of prototypes per class will resolve the imbalance, avoiding situations where some classes are underrepresented while others are overrepresented by the prototypes. Specifically, the model allocated $t = 3$ prototypes for each class, resulting in a total of $m = 30$ prototypes, the same number as the S30 model, which allows for fair comparisons.

To achieve this objective, the Balanced Model is trained using the alternative cost function AL (see Equation 4.10) with the following hyperparameters: $\lambda_{ce} = 20$, $\lambda_{ae} = 1$, $\lambda_1 = 1$, and $\lambda_{cls} = 1$. In this approach, the weight matrix W from the fully-connected layer is frozen and filled as explained in Section 4.4. This adjustment ensures that the weights connecting the prototype layer p to the softmax layer s are fixed, meaning predictions are made directly based

on the distances to the prototypes of each class, which provides clearer and more interpretable predictions. Users can directly understand the model's decisions based on the proximity of input samples to the prototypes without needing to consult the weight matrix W for deeper insights, as happens with the S15 and S30 models.

6.4 Visual Analysis

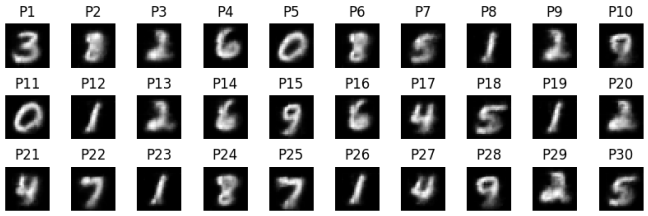
In this section, we evaluate the visualization and interpretability of the Enhanced Model (S30) and the Balanced Model (B30) to determine if they effectively address the interpretability challenges identified in the Standard Model (S15). By comparing these models, we can assess the improvements in class representation and overall interpretability achieved through the proposed modifications.

6.4.1 S30 Visual Analysis

Figure 6.2 shows the reconstructions and prototypes learned by the Enhanced Standard Model (S30). The reconstructions in Figure 6.2a demonstrate that the autoencoder performs well, accurately reconstructing the input images.

In Figure 6.2b, the prototypes learned by the S30 model are displayed. With 30 prototypes, each class is now represented by at least one prototype, addressing the limitation in the S15 model where some classes were not represented. However, while classes like digit 1 and 6 are well-represented by multiple prototypes, digit class 3 is still represented by only one prototype. This imbalance in representation means that some classes have multiple prototypes for explanations, while others have only one, which still poses a challenge for interpretability.

(a) Enhanced Standard Model (S30) Reconstructions



(b) Enhanced Standard Model (S30) Prototypes

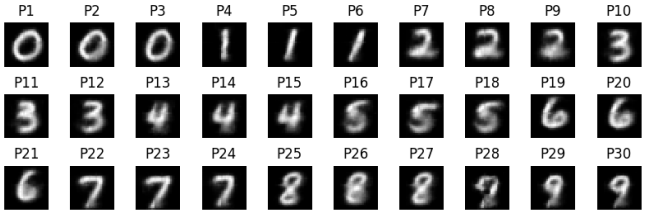
Figure 6.2: Visualization of the **Enhanced Standard Model (S30)**

6.4.2 B30 Visual Analysis

Figure 6.3 shows the reconstructions and prototypes learned by the Balanced Model (B30). The reconstructions in Figure 6.3a demonstrate that the autoencoder accurately reconstructs the input images, ensuring reliable prototype interpretations.

As shown in Figure 6.3b, the prototypes learned by the B30 model cover all classes evenly, solving the class representation imbalance found in the S15 and S30 models. This even distribution ensures better representation while maintaining good visual quality.

(a) Balanced Model (B30) Reconstructions



(b) Balanced Model (B30) Prototypes

Figure 6.3: Visualization of the **Balanced Model (B30)**

The B30 model stands out in terms of interpretability because its predictions are made directly based on the distances between the encoded images and the prototypes, as detailed in Subsection 6.3.2.2. This adjustment simplifies the explanations, as the model predictions can be understood without consulting the weight matrix W. Combined with the even representation of all classes by the prototype vectors, this ensures clearer and more consistent explanations of the model's decisions, making the B30 model the most interpretable among the three configurations.

6.5 Comparison of Accuracy and Reconstruction Error

In this section, we aim to determine if the enhancements in model interpretability contribute to relevant changes in classification accuracy and reconstruction error.

Table 6.2 shows the accuracy measured on the default test set and reconstruction error obtained by the trained models.

As it is shown in the table, the S30 and B30 models exhibit slightly higher accuracy than the S15 model, likely due to the increased number of prototypes. Moreover, these results are comparable to other non-interpretable models such as LeNet-5, which achieves an accuracy of 99.2% on the default test set [17] [6]. These findings indicate that enhancing interpretability does not necessitate sacrificing accuracy. In particular, the B30 model, which presents the highest level of interpretability (see Section 6.4.1), also achieves the highest accuracy on the default test set.

The reconstruction error (Equation 4.4) is measured on the undeformed training set, as proposed in the original paper [6]. Table 6.2 shows that the S15 model has the highest reconstruction error. However, this difference is not significant since the reconstructions provided by all three models are similar in visual quality (see Figures 6.1a, 6.2a, and 6.3a).

Metric	S15	S30	B30
Accuracy (%)	99.06	99.17	99.26
Reconstruction Error	0.0429	0.0359	0.0388

Table 6.2: Accuracy measured on the default test set and reconstruction error on the undeformed training set for the default trained models.

6.6 Conclusions

Interpretability Issues in the Original Model The original model (S15) displayed significant interpretability issues, primarily due to the imbalance in class representation among its prototypes. Some digit classes were represented by multiple prototypes, while others had none, undermining the model's ability to provide clear and consistent explanations for its classifications. Additionally, the influence of the weight matrix between the prototype layer and the softmax layer further complicated interpretability, as it caused the distances between encoded inputs and prototypes to influence predicted class probabilities in a non-transparent manner.

Successful Solutions for Enhanced Interpretability Our proposed Enhanced Model (S30) and Balanced Model (B30) effectively addressed the interpretability challenges of the original model. The S30 model increased the number of prototypes to improve class representation, ensuring that all digit classes were represented by at least one prototype. The B30 model further enhanced interpretability, by ensuring equal representation for all classes and employing a fixed weight matrix configuration that allowed predictions to be made directly based on distances to prototypes. This approach eliminated the need to consult the weight matrix for understanding model decisions, resulting in clearer and more consistent explanations.

Accuracy and Reconstruction Quality The improvements in interpretability did not come at the cost of classification accuracy. Both the S30 and B30 models exhibited slightly higher accuracy than the S15 model, indicating that the increased number of prototypes contributed positively to the models' performance. The B30 model, which offered the highest level of interpretability, also achieved the highest accuracy among the three configurations. Reconstruction error was similar across all models, with the S15 model showing the highest error, but the differences were not significant enough to impact the overall visual quality of the reconstructions. This consistency in reconstruction quality across models suggests that enhancing interpretability does not compromise the models' ability to accurately reconstruct input images.

CHAPTER 7

Evaluating the Robustness of Prototype-based Neural Networks

In this chapter, the performance and robustness of the trained models are evaluated comprehensively. The evaluation is conducted on various perturbed test sets created using different adversarial attacks, following the loss configurations explained in Chapter 5. This dual evaluation approach, comprising both standard test conditions and adversarial scenarios, aims to assess the effectiveness of the models under standard conditions and their potential resilience to adversarial perturbations. By comparing the results from the previous chapter, where performance was evaluated on the default test set, this chapter provides a thorough assessment of model robustness in diverse scenarios. The scripts used in the adversarial evaluation of the trained models can be found in the repository [1] accompanying this thesis.

7.1 Robustness Evaluation Setup

This section outlines the methodology for evaluating the robustness of the trained models against adversarial attacks. We detail the attack configurations and metrics used to assess the models' performance under both standard and adversarial conditions.

7.1.1 Specific Attack Configuration

To generate the adversarial examples in each case, the Projected Gradient Descent [4] (PGD) attack method with the l_∞ norm was used (see Section 3.2.4). The specific attack configuration parameters are: $\epsilon = 0.3$, $\alpha = 0.01$, 40 iterations, and random starts. These particular parameters were chosen based on the standard parametrization found in [4], where they generate adversarial examples over the MNIST dataset following this exact configuration. By

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

adhering to this widely accepted standard in the literature, the choice of parameters ensures that the results are more comparable and align with common evaluations conducted in the field.

7.1.2 Metrics for Evaluation

In this section, we introduce the metrics used to evaluate the performance and robustness of the trained models under adversarial conditions. These metrics provide a comprehensive understanding of how well the models withstand adversarial perturbations and maintain their accuracy. The evaluation focuses on two main aspects: the effectiveness of the adversarial attacks and the accuracy of the models under both clean and adversarial conditions.

7.1.2.1 Attack Effectiveness

The attack effectiveness is measured by evaluating the success of the attack in changing the classification and/or explanation of the model towards the intended outcomes.

For attacks where a class change is desired, the effectiveness is determined as follows: The attack is considered effective if the non-perturbed example was correctly classified by the model and is now misclassified. For targeted class attacks, the attack is only counted as effective if the misclassification is towards the target class y_t.

For attacks aiming to alter explanations, the effectiveness is determined as follows: The attack is deemed effective if the class of the closest prototype to the example has changed. In cases where a specific class k explanation e_k is desired, the attack is effective if the closest prototype now belongs to class k, whereas before it did not.

7.1.2.2 Accuracy

In addition to attack effectiveness, it is essential to measure the accuracy of the model to understand the overall impact of the attacks on model performance.

Default Accuracy: It is necessary to keep in mind the accuracy on the default (clean) test set obtained by the trained model, as presented in Section 6.5. This measurement serves as a baseline to understand the model's decay in accuracy due to adversarial attacks.

Adversarial Accuracy: This metric evaluates the accuracy of the model when tested on different adversarial test sets, generated following various configurations as explained in Section 7.1.1.

7.1.3 Measurement Methodology

To ensure robust and reliable measurements, each attack is executed 5 times. The mean and standard deviation of each metric are calculated to account for variability in the results. For targeted attacks, the target class and/or explanation is randomly selected for each example, ensuring that the chosen target is different from the original class of the example. Specifically,

the target class y_t and the desired explanation class e_k are selected such that $y_t \in \{0, \ldots, 9\} \setminus \{y\}$ and $e_k \in \{0, \ldots, 9\} \setminus \{y\}$, where y is the original class of the example.

7.2 Model Evaluation

In this section, we present a comprehensive evaluation of the trained models' performance and robustness. We analyze their effectiveness and accuracy under both standard and adversarial conditions. This evaluation helps in understanding how well the models perform when faced with adversarial attacks.

7.2.1 Effectiveness of the Attacks over the Models

Table 7.1 presents the mean and standard deviation of the effectiveness of each attack on the models. As observed, attacks tend to be similarly effective on the three models.

However, while the CU, CT, and ET attacks present high effectiveness, other attacks such as ECST and ECDT find it harder to meet their objectives. This may be due to the fact that these latter attacks need to meet both criteria to be considered effective.

The low effectiveness obtained with the ECDT attack may be attributed to the fact that the target class and explanation differ ($e_k \neq y_t$), which is contradictory. Furthermore, as explained in Chapter 4, these models make predictions based on the distances to the prototype vectors. Therefore, targeting the y_t class indirectly brings the data points closer to the prototypes corresponding to that class, which is not entirely compatible with also getting closer to the prototypes of another class e_k and still predict the target class y_t, leading to low effectiveness.

Attack / Model	**S15**	**S30**	**B30**
CU	98.85 ± 1.25%	99.17 ± 0.35%	99.26 ± 1.32%
CT	78.55 ± 2.13%	81.91 ± 1.47%	60.60 ± 2.21%
EU	54.11 ± 0.95%	42.57 ± 1.85%	38.15 ± 0.82%
ET	78.96 ± 1.53%	76.41 ± 0.65%	69.84 ± 1.81%
ECU	54.35 ± 1.38%	41.99 ± 1.22%	37.09 ± 0.57%
ECST	7.25 ± 1.74%	6.90 ± 1.49%	6.85 ± 0.95%
ECDT	7.56 ± 0.53%	8.20 ± 1.31%	7.09 ± 0.48%
Mean	**54.23 ± 1.36%**	**51.02 ± 1.19%**	**45.55 ± 1.17%**

Table 7.1: Mean and standard deviation of attack effectiveness on models S15, S30, and B30, including the overall mean values.

7.2.2 Accuracy of the Models under Adversarial Attacks

Table 7.2 presents the mean accuracies and standard deviations of each model over the standard and respective adversarial test sets. By examining the table, we can conclude that attack effectiveness is directly related to the drop in accuracy, as the attacks that produce the lowest accuracies exhibit the highest effectiveness.

Interestingly, even when an attack is not highly effective, such as the ECDT attack, the models still experience a significant drop in accuracy. This suggests that even when an attack does not fully achieve its objectives, it can still successfully deceive the models.

Furthermore, it appears that the balanced class representation provided by the prototypes, combined with the direct prediction based on distances to the prototypes (see Chapter 6), helps the B30 model achieve the highest mean accuracy. Additionally, both the S30 and B30 models attain higher mean accuracies than the S15 model, indicating that increasing the capacity of the models enhances their adversarial robustness.

	Accuracy		
Attack / Model	**S15**	**S30**	**B30**
Default	99.06%	99.17%	99.26%
CU	0.21 ± 0.87%	0.00 ± 1.34%	0.00 ± 0.45%
CT	11.97 ± 1.02%	11.25 ± 0.93%	16.50 ± 0.77%
EU	45.20 ± 0.92%	56.76 ± 1.32%	61.30 ± 1.14%
ET	20.18 ± 1.14%	22.93 ± 0.94%	29.58 ± 1.07%
ECU	44.88 ± 0.83%	57.30 ± 1.22%	62.37 ± 0.95%
ECST	19.57 ± 1.21%	23.32 ± 0.97%	29.36 ± 1.11%
ECDT	21.06 ± 1.24%	22.92 ± 1.01%	32.24 ± 0.93%
Mean	**32.77 ± 0.90%**	**36.71 ± 0.97%**	**41.33 ± 0.80%**

Table 7.2: Mean and standard deviation of test accuracy for each attack on models S15, S30, and B30, including the overall mean values.

7.3 Conclusions

Vulnerability of Models This study presents the first in-depth analysis of prototype-based neural networks' vulnerability to adversarial attacks. Our evaluation has revealed that these models exhibit a significant degree of vulnerability, which poses a serious problem as it compromises the reliability and security of the models in real-world applications. Furthermore, the interpretability provided by prototype-based neural networks becomes useless when adversarial examples fool both the explanations and the performance of the model. Addressing this issue is crucial, and solutions to improve model robustness will be explored in Chapter 9.

Effectiveness of Different Attacks We sought to determine the effectiveness of the proposed attacks and found considerable variation among them. Specifically, the CU (Change Class Untargeted), CT (Change Class Targeted), and ET (Change Explanation Targeted) attacks were the most effective, achieving high success rates in altering the model's classification and significantly impacting the accuracy obtained by the models. In contrast, the ECST (Change Explanation and Class Same Target) and ECDT (Change Explanation and Class Different Target) attacks were less effective, highlighting the complexity of achieving dual-objective adversarial success. The low effectiveness of the ECDT attack, in particular, indicates that

the reasoning process of prototype-based neural networks provides a robust defense against attacks targeting both the class and the explanation.

Discrepancy Between Accuracy and Effectiveness One key insight gained from this study is the discrepancy between an attack's ability to fool the model (impact on accuracy) and its success in achieving all adversarial objectives (changing both the class and/or the explanation). For example, the ECDT attack, despite its lower overall effectiveness, still significantly reduced model accuracy. This indicates that an attack can degrade performance even if it doesn't fully achieve its intended goals, emphasizing the complexity of adversarial robustness.

Capacity Alone Helps Increasing model capacity enhances robustness against adversarial attacks. The S30 and B30 models, which have higher capacities than the S15 model, demonstrated higher mean accuracies under adversarial conditions. This suggests that larger models could provide a greater level of robustness against adversarial perturbations, as observed in other related works [4].

Interpretability Enhances Robustness Our findings suggest that enhanced interpretability contributes to greater robustness against adversarial attacks. The B30 model, which utilizes balanced class representation and bases its predictions on distances to prototype vectors, demonstrated the highest mean accuracy and the lowest attack effectiveness. This insight suggests that prioritizing model interpretability could lead to more robust architectures, offering a new avenue for enhancing model robustness beyond traditional adversarial training methods.

CHAPTER 8

Improving the Robustness of Prototype-based Neural Networks

In this chapter, we explore adversarial training to enhance the robustness of prototype-based neural networks. We detail the training procedures and modifications to maintain model interpretability while improving robustness. Additionally, we examine different training strategies, such as training from scratch and fine-tuning already trained models from Chapter 6. A comprehensive visual analysis of the trained robust models is provided to assess their interpretability. Finally, we evaluate the performance and resilience of these models against diverse adversarial attacks, offering insights into their robustness and identifying potential areas for further research. All the trained robust models along with their corresponding visualization and evaluation scripts can be found in the repository [1] accompanying this thesis.

8.1 Adversarial Training

Adversarial training is a crucial technique for increasing a model's robustness against adversarial attacks [3]. This method involves training the model with a mix of clean and adversarial examples, thereby enhancing its ability to resist perturbations and improving its overall resilience.

8.1.1 Training Procedure

To train robust models, we generate an additional adversarial mini-batch for each mini-batch of 250 samples randomly selected from the training set, which results in a training process that alternates between non-adversarial and adversarial mini-batches. By exposing the models to a diverse adversarial and non-edversarial inputs, this approach enhances their robustness against adversarial attacks while preserving their performance on clean data.

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

8.1.2 Adversarial Example Generation

We decided to generate the adversarial examples using the Projected Gradient Descent (PGD) attack method (see Section 3.2.4) with the l_∞ norm, as it is the strongest attack using first-order information about the network [4]. The specific attack configuration parameters are: $\epsilon = 0.3$, $\alpha = 0.02$, 20 iterations, and random starts. The adversarial examples are generated using the CU type attack (see 5.1), where the classification loss is maximized, thereby reducing the probability of an example belonging to its original class y.

In addition, we decided to remove the elastic transformation technique described in Section 6.3. Instead, we introduced variability through the use of adversarial examples, which vary slightly due to random starts. By alternating between adversarial and non-adversarial mini-batches, we ensured that the model encountered diverse examples in each epoch. This approach allowed the models to train on a varied set of images during each epoch, effectively preventing overfitting.

8.1.3 Modified Autoencoder Loss for Adversarial Examples

It is important to note that adversarial examples tend to be noisy due to the perturbations added by adversarial attacks to the original images (see Figure 8.7). In the case of prototype-based networks, this noise can make the learned prototypes less clear, as they tend to resemble the training examples due to the R1 (see Equation 4.5), R2 (see Equation 4.6), S1 (see Equation 4.9), and CLS (see Equation 4.8) interpretability loss functions. To address this issue, this section proposes a solution by introducing a modification to the training procedure.

The original autoencoder loss term (see Equation 4.4) measures the difference between the original input and its reconstruction from the latent space. In order to prevent the undesired effects, such as the prototypes being similar o the adversarial examples, we decided to modify the autoencoder loss term for the adversarial examples. Specifically, when an adversarial example x' is crafted from the example x, the modified autoencoder loss $\hat{A}$ measures the reconstruction error of the adversarial example x' with respect to the original example x, which is mathematically defined as:

$$\hat{A}(g \circ f, x, x') = \left\| (g \circ f)(x') - x \right\|_2^2, \tag{8.1}$$

where $(g \circ f)(x')$ is the reconstruction of the adversarial input x' and x is the non-adversarial example. This modification is intended to ensure that the adversarial examples have similar reconstructions to the non-adversarial examples, and therefore obtain prototypes that are as clean and meaningful as possible. See Figure 8.1 for an example.

Furthermore, ensuring that adversarial examples have similar reconstructions to non-adversarial examples, contributes significantly to the robustness of the autoencoder. If two different inputs, x' and x, should be reconstructed to the same image, for the decoder to be able to produce a similar reconstruction $g(f(x)) \approx g(f(x'))$, they must have the same or a very similar encoding $f(x) \approx f(x')$. By achieving similar encodings, if the non-adversarial example x is correctly classified (i.e., it is closer to its corresponding class prototypes), the encoded

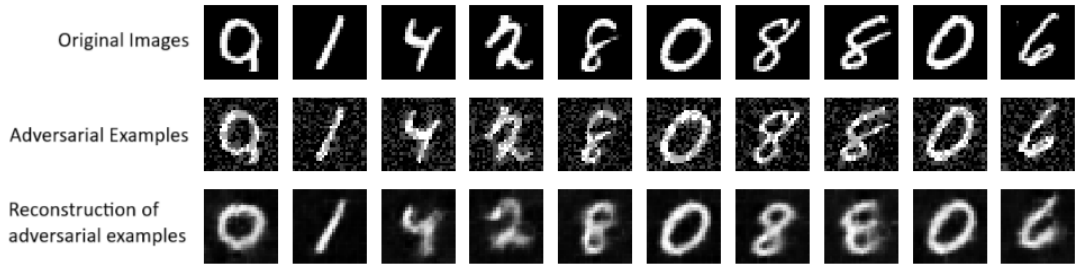


Figure 8.1: Example of the reconstruction of adversarial examples generated from original images. An adversarially trained model was used (RS30), and as can be seen, it is able to remove the adversarial noise from the adversarial examples, decoding them into more clear images, which are similar to the original ones.

adversarial example $f(x')$ will have similar distances to those prototypes. Consequently, this increases the likelihood of the adversarial example being correctly classified.

8.1.4 Training Approaches

This section discusses two main approaches to achieve robustness: training from scratch and fine-tuning pre-trained models. Each method has its own advantages and challenges, and understanding these can help in selecting the most appropriate strategy for different scenarios.

8.1.4.1 Training from Scratch

In order to train robust models, this approach focused on training the S30 and B30 models described in Chapter 6 by following the proposed adversarial training scheme. The S15 model was not adversarially trained since the original version did not provide sufficient interpretability, due to a lack of class representation (see Section 6.3.1.1).

The adversarially trained models follow the same architecture and training hyperparameters as their non-adversarial versions, and were also trained for 750 epochs (see Section 6.3). Using the same respective configurations as the parent models allows for a fair comparison, enabling us to appreciate the gains in robustness and any potential changes in prototype quality they may present.

8.1.4.2 Fine-tuning

Training robust models from scratch requires substantial computational effort and time due to the need for extensive training cycles and the generation of adversarial examples at each step. Fine-tuning addresses this issue by taking an already trained model and adapting it to a specific task with further training on task-specific data. This approach significantly reduces computational effort and training time while leveraging the pre-existing knowledge of the pre-trained model, requiring only a few additional epochs to achieve significant improvements.

Considering our context, the fine-tuning procedure follows the same adversarial training scheme and adversarial example generation method as the scratch training (see Section 8.1), alternating between non-adversarial and adversarial mini-batches. This alternation is crucial because training exclusively with adversarial examples can lead to excessive corruption of the model prototypes, reducing its overall performance on non-adversarial data and quality of the prototypes.

In our particular case, we decided to focus the experimentation in the already trained B30 model, since it presents the highest interpretability and performance (see Chapters 6 and 7). Initially, we fine-tuned the model without freezing any part of it. Subsequently, we conducted an additional experiments: one freezing the prototype vectors and another one freezing the autoencoder. We propose freezing different parts of the model to evaluate their importance in training robust models while maintaining interpretability.

Please note that all the fine-tuned models were trained for 20 epochs, as this was sufficient to significantly increase robustness. We verified that increasing the number of epochs to 50 did not bring significant improvements, as the accuracy plateaus.

8.2 Visual Analysis of Trained Robust Models

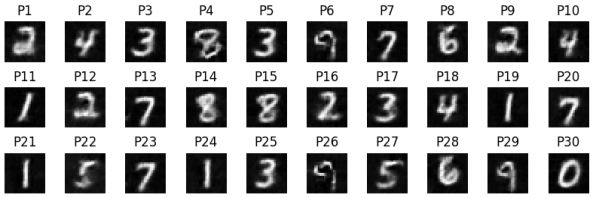
In this section, we present a visual analysis of the prototypes and reconstructions produced by various trained robust models to assess whether adversarial training can maintain interpretability and quality despite adversarial examples. This examination helps us understand the impact of different training approaches on model interpretability.

Adversarially trained models, both from scratch and fine-tuned, generally present interpretable prototypes and accurate reconstructions by the autoencoder. The prototypes may differ from the original models (B30 and S30) due to the randomness of the training process. The following list summarizes the obtained results:

- **Robust Enhanced Standard Model (RS30):** As shown in Figure 8.2, the RS30 model still resembles real hand-written digits as prototypes, while maintaining an accurate reconstruction of the inputs.
- **Robust Balanced Model (RB30):** As can be seen in Figure 8.3, the prototypes learned by the RB30 model still resemble real hand-written digits and present a balanced representation of the digit classes. Moreover, the decoder still shows accurate reconstructions. However, some prototypes contain some noise, as if they resembled adversarial examples, which may be a side effect of the adversarial training.
- **Fine-Tuned B30 with Nothing Frozen (FNB30n):** The visualizations present in Figure 8.4 remark that the fine-tuning does not impact the quality of the prototypes and the decodings.
- **Fine-Tuned B30 with Frozen Prototypes (FNB30p):** Note that the fact that the prototype vectors were frozen does not mean that their visualization in the pixel space remains the same, since the decoder g can change during the fine-tuning process.

However, as shown in Figure 8.5, the prototypes still resemble quality hand-written digits while the decoder correctly reconstructs the encoded inputs.

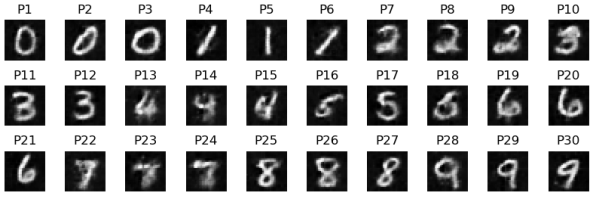
- **Fine-Tuned B30 with Frozen Autoencoder (FNB30a):** Interestingly, as shown in Figure 8.6, the obtained prototypes no longer represent handwritten digits. This result highlights that the prototypes may have adapted to the new adversarial/non-adversarial training set, while the autoencoder, being frozen, is no longer able to decode the prototypes into interpretable digits.



(a) Robust Standard Model (RS30) Prototypes

(b) Robust Standard Model (RS30) Reconstructions

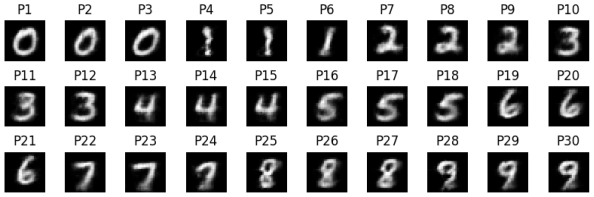
Figure 8.2: Visualization of the **Robust Standard Model (RS30)**



(a) Robust Balanced Model (RB30) Prototypes

(b) Robust Balanced Model (RB30) Reconstructions

Figure 8.3: Visualization of the **Robust Balanced Model (RB30)**



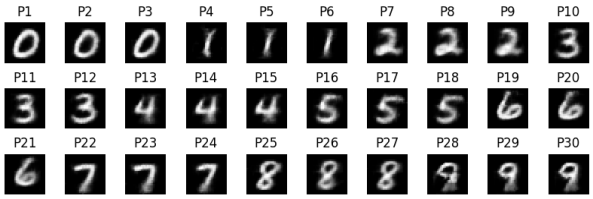
(a) Fine-Tuned B30 with Nothing Frozen (FNB30n) Prototypes

(b) Fine-Tuned B30 with Nothing Frozen (FNB30n) Reconstructions

Figure 8.4: Visualization of the **Fine-Tuned B30 with Nothing Frozen (FNB30n)**

8.3 Attack Effectiveness and Model Accuracy

This section is dedicated to the comprehensive evaluation of the performance and robustness of the trained robust models (RS30, RB30, FTB30n and FTB30p). These evaluations are conducted using the same criteria and methodologies as applied to the non-robust models (see Chapter 7) to ensure consistency and comparability. The evaluation process involves testing the models



(a) Fine-Tuned B30 with Prototypes Frozen (FNB30p) Prototypes

(b) Fine-Tuned B30 with Prototypes Frozen (FNB30p) Reconstructions

Figure 8.5: Visualization of the **Fine-Tuned B30 with Prototypes Frozen (FNB30p)**

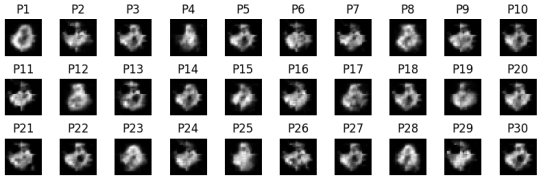


Figure 8.6: Prototypes learned by the **Fine-Tuned B30 with Frozen Autoencoder (FNB30a)** visualized in pixel space.

on both the standard test set and various perturbed test sets generated through different adversarial attacks. This dual testing approach aims to find out the models' effectiveness under normal conditions and their resilience against adversarial perturbations.

Table 8.1 illustrates the decrease in attack effectiveness for various attack configurations across all adversarially trained models. Each cell shows the mean attack effectiveness and standard deviation for the corresponding attack and model. The red percentage indicates the effectiveness loss compared to the respective non-robust model. The data reveals a substantial reduction in attack effectiveness, indicating that robust models exhibit increased resistance to adversarial attacks compared with their respective non-robust versions. Additionally, Table 8.2 presents the gains or losses in accuracy for both the default and adversarial test sets. Each cell shows the mean accuracy and standard deviation for the corresponding attack and model. The red or green percentage indicates the accuracy loss or gain compared to the respective non-robust model. It is noteworthy that all robust models experienced a decrease in accuracy on the default test set, probably due to the adaptation to adversarial examples. However, there is a significant increase in accuracy on the adversarial test set for all adversarially trained models. In some instances, the accuracy on the adversarial test set surpasses that on the default test set, suggesting that the robust models might have memorized the noise introduced by the PGD attack to some extent.

These results underscore the effectiveness of adversarial training in significantly enhancing the performance and robustness of prototype-based models. Notably, fine-tuned models (FTB30n and FTB30p) demonstrate similar robustness and performance improvements as models trained from scratch (RS30 and RB30). This comparable performance, along with the reduced training time required for fine-tuning (730 fewer epochs), suggests that fine-tuning non-adversarially trained models is a more efficient approach for achieving robustness.

Effectiveness with Percentage Loss				
Attack / Model	**RS30** (RS30 - S30)	**RB30** (RB30 - B30)	**FTB30n** (FTB30n - B30)	**FT30p** (FT30p - B30)
CU	6.92 ± 1.24% -92.25%	6.56 ± 1.73% -92.70%	11.07 ± 0.93% -88.19%	8.35 ± 1.89% -90.91%
CT	1.14 ± 0.54% -80.77%	0.94 ± 1.02% -59.66%	1.78 ± 0.67% -58.82%	1.22 ± 1.47% -59.38%
EU	0.51 ± 0.83% -42.06%	0.62 ± 1.48% -37.53%	1.26 ± 0.28% -36.89%	1.19 ± 1.15% -36.96%
ET	1.00 ± 1.33% -75.41%	0.77 ± 1.65% -69.07%	1.97 ± 0.74% -67.87%	1.39 ± 1.55% -68.45%
ECU	0.59 ± 1.12% -41.40%	0.57 ± 0.56% -36.52%	1.34 ± 0.98% -35.75%	1.34 ± 1.32% -35.75%
ECST	0.09 ± 1.23% -68.80%	0.08 ± 1.42% -68.06%	0.15 ± 0.36% -66.70%	0.16 ± 1.71% -66.69%
ECDT	0.11 ± 0.92% -67.60%	0.08 ± 1.35% -67.01%	0.23 ± 0.62% -66.86%	0.08 ± 1.24% -67.01%
Mean	**1.48 ± 1.03%** **-66.90%**	**1.37 ± 1.32%** **-61.51%**	**2.54 ± 0.65%** **-60.15%**	**1.96 ± 1.48%** **-60.74%**

Table 8.1: Mean and standard deviation of attack effectiveness with percentage loss for different models and attacks. Red text indicates the percentage loss in effectiveness compared to the respective non-robust model.

Accuracy with Percentage Gain				
Attack / Model	**RS30** (RS30 - S30)	**RB30** (RB30 - B30)	**FTB30n** (FTB30n - B30)	**FT30p** (FT30p - B30)
Default	98.25 ± 1.13% -0.92%	96.51 ± 1.13% -2.75%	98.38 ± 1.03% -0.88%	98.42 ± 0.95% -0.84%
CU	91.78 ± 1.16% +91.78%	91.93 ± 0.10% +91.93%	87.34 ± 1.94% +87.34%	90.11 ± 1.28% +90.11%
CT	97.31 ± 1.61% +86.06%	97.18 ± 1.61% +80.68%	95.73 ± 0.36% +79.23%	96.40 ± 1.24% +79.90%
EU	98.49 ± 1.41% +41.73%	98.09 ± 1.91% +36.79%	97.56 ± 0.09% +36.26%	97.86 ± 1.31% +36.56%
ET	98.07 ± 1.26% +75.14%	98.10 ± 1.66% +68.52%	96.87 ± 1.33% +67.29%	97.57 ± 0.14% +67.99%
ECU	98.41 ± 0.84% +41.11%	98.08 ± 0.29% +35.71%	97.42 ± 0.18% +35.05%	97.64 ± 0.60% +35.27%
ECST	98.00 ± 0.49% +74.68%	98.06 ± 0.60% +68.70%	97.04 ± 1.52% +67.68%	97.54 ± 1.63% +68.18%
ECDT	98.08 ± 1.13% +75.16%	98.05 ± 1.71% +65.81%	97.19 ± 1.81% +64.95%	97.36 ± 0.46% +65.12%
Mean	**97.30 ± 1.13%** **+60.59%**	**97.00 ± 1.13%** **+55.67%**	**95.94 ± 1.03%** **+54.62%**	**96.61 ± 0.95%** **+55.29%**

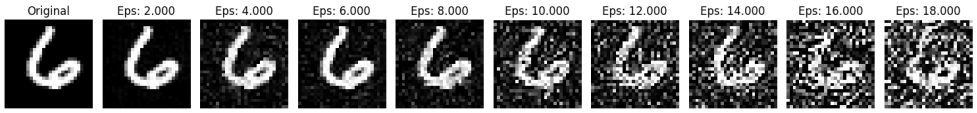
Table 8.2: Mean and standard deviation of test accuracy with percentage gain for different models and attacks. Red text indicates the percentage loss and green text indicates the percentage gain in accuracy compared to the respective non-robust model.

8.4 Expanding the Horizons of the Attacks

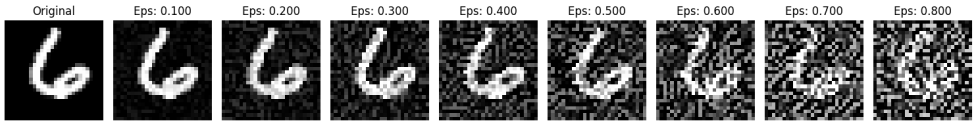
Until now, the models have been evaluated on adversarial test sets generated using the Projected Gradient Descent (PGD) attack with the l_∞ norm and an ϵ value of 0.3. This specific attack configuration was also used during the adversarial training phase, ensuring that the models were exposed to this type of perturbation during their development. While this approach has proven effective in enhancing the models' robustness against known adversarial attacks, it is crucial to understand how well these models generalize to different types of adversarial noise. A comprehensive evaluation will provide deeper insights into the models' resilience and practical applicability in real-world scenarios where adversarial threats can vary significantly.

Models resistance to different ϵ values, l_2 and l_∞ bounded PGD attacks To conduct a comprehensive assessment of the adversarial robustness of our models, we performed two supplementary experiments. Firstly, we examined the resilience to l_∞-bounded attacks for various values of ϵ. Secondly, we evaluate the resistance of our model to attacks that are bounded in l_2 norm as opposed to l_∞ norm.

For all the PGD attacks, we use 40 iterations while the step size $\alpha = \frac{4}{3} \cdot \epsilon/40$, which guarantees sufficient movement freedom inside the ϵ-ball. For a visual comparison of how much the image is corrupted by the different values of epsilon, see Figure 8.7.



(a) PGD l_2 attack effect with different ϵ



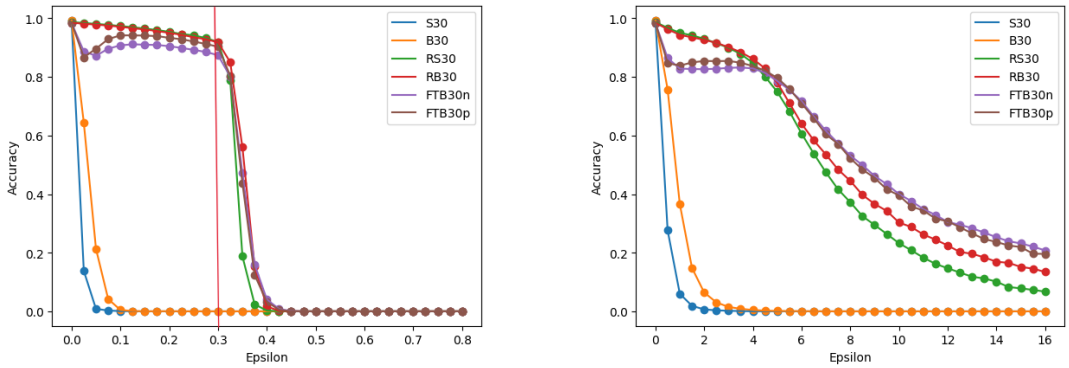
(b) PGD l_∞ attack effect with different ϵ

Figure 8.7: Comparison of adversarial effects on a single image using PGD l_2 and l_∞ attacks with varying ϵ values. (a) Shows the impact of the l_2 norm attack, while (b) displays the effect of the l_∞ norm attack.

Figure 8.8 presents the accuracy of adversarially and non-adversarially trained models against PGD adversaries generated from the test set with different perturbation strengths and norms.

In case of the PGD l_∞ attacks (Figure 8.8a), it is evident that for ϵ values below 0.3, all the robust models achieve high accuracies. However, as ϵ exceeds 0.3, these robust models encounter a significant drop in accuracy. Conversely, non-robust models show a steep decline in accuracy at much smaller ϵ values, even those below 0.1.

Turning to the PGD l_2 attacks (Figure 8.8b), robust models display resilience even at high ϵ levels, such as $\epsilon = 6$, where the decline in accuracy is much slower due to the attack's difficulty in finding effective adversarial examples. Similarly to the l_∞ attack, non-robust models suffer a sharp decrease in accuracy at smaller ϵ values.



(a) PGD l_∞ attack

(b) PGD l_2 attack

Figure 8.8: Accuracy of the adversarially and non-adversarially trained models against PGD adversaries generated from the test set with different perturbation strength and norms, using the change class untargeted loss (CU 5.1).

From this analysis, it is clear that l_∞ attacks are much stronger than l_2 attacks, as they cause a much steeper decline in accuracy. Moreover, the fine-tuned models display a small decrease in accuracy with smaller ϵ values, especially at the beginning, which may indicate a weakness present in the fine-tuning approach. Additionally, it may be the case that the trained models, having been trained with $\epsilon = 0.3$, have memorized the specific noise for the PGD attack. A further study of the effects of different attacks should be conducted in future work.

8.5 Conclusions

Effectiveness of Adversarial Training Adversarial training has proven to be highly effective in enhancing the robustness of our models against adversarial attacks. The significant reduction in attack effectiveness across various configurations and the gain in accuracy underscores the efficacy of this approach.

Fine-Tuning as an Efficient Approach Fine-tuning pre-trained models has shown to be a highly efficient method for achieving robustness. As evidenced by the results, fine-tuned models (FTB30n and FTB30p) exhibit similar levels of robustness and performance improvements as models trained from scratch (RS30 and RB30). This approach significantly reduces the computational effort and time required for training, leveraging the pre-existing knowledge of the models and adapting them to specific tasks with minimal additional training.

Impact on Default Test Set Accuracy While adversarial training leads to a decrease in accuracy on the default test set, this is compensated by a significant increase in accuracy on the adversarial test set. The robust models exhibit a trade-off, where the slight reduction in performance on clean data is balanced by a marked improvement in resilience against adversarial examples. This indicates that the models are better equipped to handle real-world scenarios where adversarial perturbations are prevalent.

Effect of Different Norms and Epsilons Our analysis of the models' resistance to different ϵ values and norms reveals important insights. PGD l_∞ attacks are much stronger than l_2 attacks. Robust models show resilience even at higher ϵ levels for l_2 attacks, whereas l_∞ attacks cause a more significant drop in accuracy at lower ϵ values.

Memorization of Specific Noise There is a possibility that models trained with $\epsilon = 0.3$ value, may have memorized the noise pattern for the PGD attack used during training, even if random starts have been used. This potential memorization suggests that the models could be more vulnerable to other types of adversarial noise not encountered during training. Future work should include a comprehensive study of the effects of different attacks and training configurations to ensure the models' robustness is generalized and not overly specific to certain adversarial conditions.

CHAPTER 9

A Deeper Analysis of Robust Prototype-based Neural Networks

While previous chapters have centered on the interpretability and overall performance of these models, this chapter aims to provide a detailed analysis of the effects of fine-tuning for robustness against adversarial attacks and the stability of the autoencoder component. Specifically, we examine the changes in the spatial distribution of adversarial and non-adversarial examples in the latent space and assess the autoencoder's ability to maintain consistent reconstructions under adversarial conditions. This comprehensive approach offers insights into the models' internal adaptations and their robustness to adversarial attacks. The scripts used for this deeper analysis can be found in the repository [1] accompanying this thesis.

9.1 A Deeper Look into the Fine-tuning Effects

In this section, we aim to explore whether fine-tuning the models to enhance their robustness to adversarial attacks causes any significant changes in the spatial distribution of adversarial and non-adversarial examples in the latent space, as well as the prototype vectors. Understanding these changes can provide insights into how the model's internal representations adapt during the fine-tuning process.

9.1.1 Latent Space Visualization and Analysis

Principal Component Analysis (PCA) [19] is a statistical technique used for dimensionality reduction. It transforms the original high-dimensional data into a new coordinate system where the greatest variances in the data are captured by the first few principal components. This method helps in simplifying the complexity of the data while retaining its essential

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

patterns and structures, making it easier to visualize and analyze. In this analysis, PCA is employed to project the prototype vectors and encoded examples into a 2D space for better visualization and understanding of the effects of fine-tuning on the model's internal representations.

We delve into the distribution of both non-adversarial and adversarial examples before and after fine-tuning, examining how these distributions shift and what these shifts imply about the model's robustness. By reducing the dimensionality of the data using PCA, we can achieve a clearer visualization of these distributions.

Two important notes should be considered during this analysis. First, for the RB30p model, while the prototypes appear to move in the 2D projection due to changes in the encoding of the test examples, the prototypes themselves remain fixed in the high-dimensional space. Second, the adversarial test set was generated using the Projected Gradient Descent [4] (PGD) attack method with the l_∞ norm (see Section 3.2.4). The specific attack configuration parameters are: $\epsilon = 0.3$, $\alpha = 0.01$, 40 iterations, and random starts. This attack aimed to change the class in an untargeted way, following the CU loss configuration (see 5.1).

9.1.1.1 Non-adversarial Examples Distribution

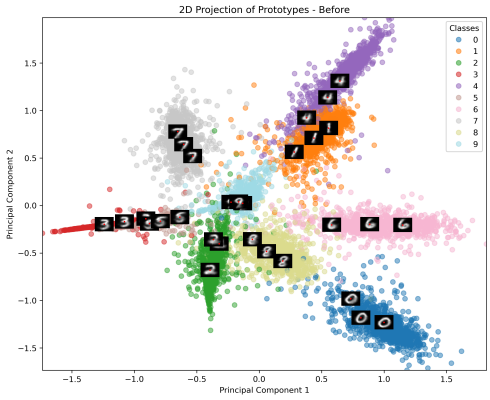
In this subsection, we analyze the distribution of non-adversarial examples and prototype vectors in the feature space before and after fine-tuning. By visualizing these distributions, we aim to understand how fine-tuning affects the model's internal representations and its ability to generalize to non-adversarial data.

Figure 9.1 illustrates the evolution of the feature space arrangement due to fine-tuning. Initially, before fine-tuning (Figure 9.1a), the prototypes and examples are tightly clustered. However, both after fine-tuning with all components unfrozen (Figure 9.1b) and freezing the prototypes (Figure 9.1c) we observe that the examples tend to be more dispersed and overlapped around their respective class prototypes, which may be the reason why these models experienced a accuracy decay on the default test, as shown in the previous chapter (see Table 8.2).

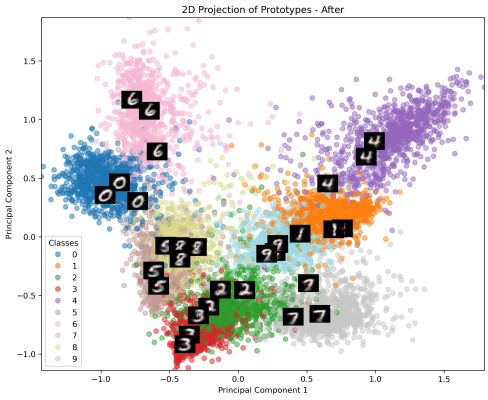
9.1.1.2 Adversarial Examples Distribution

In this subsection, we investigate how the adversarial examples and prototype vectors are distributed in the latent space before and after fine-tuning. This examination provides insights into the model's improved generalization and robustness against adversarial attacks as a result of the fine-tuning process.

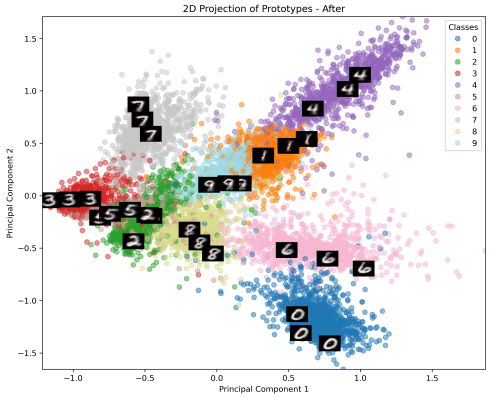
Figure 9.2 shows the changes in the feature space arrangement for the adversarial test set. Before fine-tuning (Figure 9.2a), the adversarial test set is scattered and does not cluster well around the prototypes, highlighting the model's vulnerability to adversarial perturbations. The dispersed distribution indicates that the non-robust model fails to generalize effectively when exposed to adversarial attacks. After fine-tuning with nothing frozen (Figure 9.2b) and with prototypes frozen (Figure 9.2c), the adversarial test set becomes better clustered around



(a) B30 - Before Fine-Tuning



(b) FTB30n - After Fine-Tuning with Nothing Frozen



(c) FTB30p - After Fine-Tuning with Prototypes Frozen

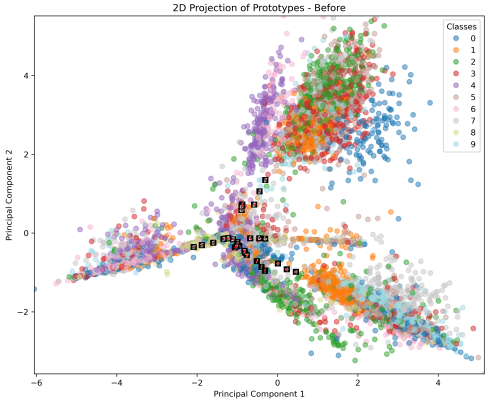
Figure 9.1: 2D distribution of the encoded examples and prototype vectors from the test set.

the prototypes, suggesting that the model has learned more robust features. This improved clustering indicates that the fine-tuned models are better at handling the adversarial test set, leading to enhanced performance and stability.

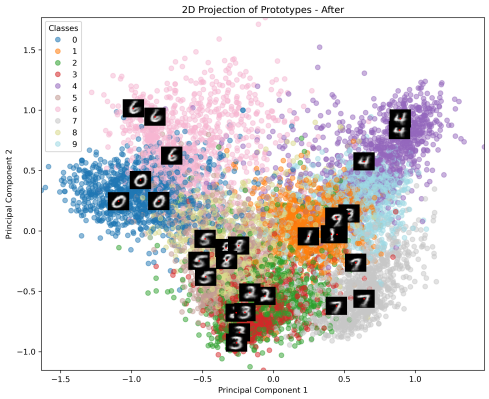
9.2 Autoencoder Robustness Analysis

In this section, we shift our focus to examining the robustness of the autoencoder component in our prototype-based neural network. The robustness of the autoencoder is crucial in ensuring the integrity of the representations it generates, which in turn affects the overall performance and reliability of the models. This section aims to analyze the autoencoder's ability to maintain stable and consistent reconstructions under adversarial attacks.

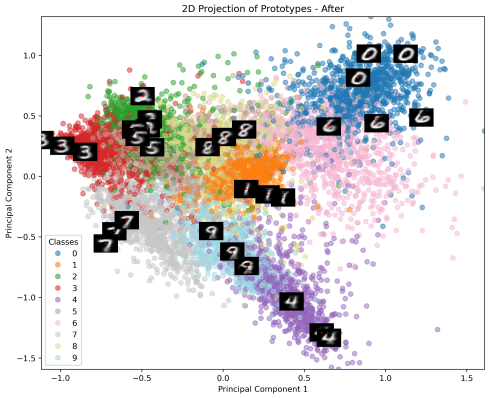
To assess this, we analyze the autoencoder's ability to remove noise by measuring the difference between the reconstruction of an original example $g(f(x))$ and its corresponding



(a) B30 - Before Fine-Tuning



(b) FTB30n - After Fine-Tuning with Nothing Frozen



(c) FTB30p - After Fine-Tuning with Prototypes Frozen

Figure 9.2: 2D distribution of the encoded adversarial test set and prototype vectors.

adversarial example $g(f(x'))$. This difference reflects the degree to which the model can eliminate noise from the adversarial example.

Autoencoder ability to remove noise It is important to note that while training the robust models through adversarial training, the modified autoencoder loss ($\hat{A}$ 8.1) was used with the adversarial examples. This approach forced the reconstruction of an adversarial example x' to match the non-adversarial example x, making the autoencoder learn to remove adversarial noise. As shown in Figure 9.3, the RB30 model is able to remove the adversarial noise from the adversarial examples, decoding them into more clear images that resemble the non-adversarial ones, while the non-robust version B30, decodes the adversarial examples into corrupted images. Other robust models such as RS30 show similar behavior (see Figure 8.1).

In Table 9.1, the mean decoding difference over the examples in the adversarial test set is

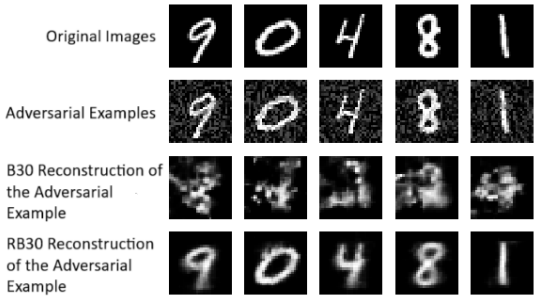


Figure 9.3: Example of the reconstruction of B30 model and RB30 model of adversarial examples generated from randomly selected images from the original test set.

presented. The results show that the robust models (RS30, RB30, FTB30N, and FTB30P) achieve significantly lower mean decoding differences compared to the non-robust models (S15, S30, and B30). Specifically, the RS30 and RB30 models present the lowest mean reconstruction differences, indicating their high effectiveness in maintaining stable reconstructions under adversarial attacks. The fine-tuned models FTB30N and FTB30P also demonstrate good robustness. On the other hand, the non-robust models show higher mean decoding differences, which highlights the importance of robust training techniques in preserving the integrity of the reconstructed outputs even in the presence of adversarial perturbations.

Model	**S15**	**S30**	**B30**	**RS30**	**RB30**	**FTB30n**	**FTB30p**
MDD	0.0631	0.0736	0.061	0.0043	0.0041	0.0083	0.0076

Table 9.1: Mean Decoding Differences (MDD) across various models, indicating the effectiveness of the modified autoencoder loss in preserving decoding stability under adversarial attacks. The adversarial examples were generated using the PGD l_∞ attack with $\epsilon = 0.3$, $\alpha = 0.01$, and 40 iterations, with random starts. The used adversarial loss configuration is change class untargeted loss (CU 5.1).

9.3 Conclusions

Effect of Fine-tuning on Latent Space Distributions Fine-tuning the models to enhance their robustness to adversarial attacks results in significant changes in the spatial distribution of adversarial and non-adversarial examples in the latent space. PCA visualizations reveal that, in non-robust models, the non-adversarial examples are tightly clustered around their respective class prototypes. However, in robust models, the non-adversarial examples, while still clustered, show increased overlapping with other class examples. This overlapping in the robust models likely contributes to the observed accuracy decay on the default test set (see Chapter 8).

Improved Clustering of Adversarial Examples Post Fine-Tuning The distribution of adversarial examples shows a marked improvement in clustering around prototypes after fine-tuning. Initially, adversarial examples are scattered and fail to cluster well, demonstrating

the model's vulnerability. After the fine-tuning, both with and without freezing prototypes, the adversarial examples cluster more closely around the prototypes, indicating that the models have learned more robust features and are better equipped to handle adversarial attacks.

Autoencoder's Robustness to Adversarial Attacks The robustness of the autoencoder is vital for ensuring the integrity of its generated representations. Robust models exhibit a significantly lower mean decoding difference, highlighting their effectiveness in maintaining stable reconstructions despite adversarial perturbations. The autoencoder, trained with a modified loss function during adversarial training, successfully learns to remove adversarial noise, resulting in clearer and more accurate reconstructions of adversarial examples. Furthermore, having a low mean decoding difference also indicates that adversarial and non-adversarial examples have similar encodings, which likely contributes to the robust models' high resilience against the proposed adversarial attacks.

CHAPTER 10

General Conclusions and Future Work

In this chapter, we present the most relevant contributions derived from the work carried out throughout the project and the results obtained, along with proposals for future research.

10.1 Technical Contributions

The principal contributions of this bachelor thesis are as follows:

Identification and Resolution of Interpretability Issues: The research identifies and addresses interpretability challenges within the selected prototype-based neural network architecture. We propose modifications to achieve a more balanced class representation by the prototypes and simplify the decision process to make predictions based on the similarities between the prototypes and the inputs. These changes enhance the clarity and reliability of the model's explanations.

Adversarial Attack Framework: A comprehensive framework for executing adversarial attacks on prototype-based neural networks has been proposed. The study demonstrates that prototype-based neural networks are vulnerable to such attacks. However, it is also observed that models with greater capacity and improved interpretability exhibit increased robustness against these adversarial perturbations. This observation opens the door for future research to explore how enhancing model capacity and interpretability can further improve robustness.

Development of Defensive Strategies: The research identifies and implements defensive strategies to counter adversarial attacks. By adapting the adversarial training procedure to maintain prototype interpretability while enhancing robustness, the study shows that significant improvements in model robustness can be achieved both by training robust networks

from scratch and through fine-tuning over a few epochs. Fine-tuning is notably more efficient, requiring less computation and time, while yielding results similar to those obtained from training from scratch.

Evaluation of Model Robustness Against Various Adversarial Attacks: The models were subjected to various adversarial attacks, revealing that while the proposed defenses enhance robustness, they do not provide absolute protection. The models still exhibit vulnerabilities to different perturbation strengths and norms not encountered during the adversarial training, highlighting the need for further research in this area.

Open Access to Research Outputs To support reproducibility and further research, all scripts and trained models from this study are made available on the project's GitHub repository[1]. This ensures that the research community can access, replicate, and build upon our work.

10.2 Personal Contributions

This project has been pivotal in enhancing my skills and knowledge across several dimensions.

Firstly, it has served as an excellent introduction to the field of research. Through this project, I have learned how to set objectives and adapt to challenges, reinforcing many concepts from my bachelor's degree, particularly in non-conventional deep learning architectures and training approaches.

Additionally, the project has significantly improved my organizational and communication skills. Regular discussions with my advisor have helped refine my approaches and ensure the project's success. This process has enhanced both my technical skills and my ability to clearly articulate complex ideas.

Moreover, I find this line of research incredibly fascinating and essential for increasing trust in AI-based decision systems. Achieving the proposed objectives in this field has been deeply satisfying. I am proud of my contributions and the valuable insights gained, which will undoubtedly benefit my future endeavors, especially in research.

10.3 Future Work

Here we summarize some future research lines derived from this study.

Explore Different Adversarial Attacks While this study primarily focuses on PGD attacks using l_2 and l_∞ norms, future research could benefit from exploring a broader range of adversarial attacks. The robustness gains achieved by the proposed defense methods are impressive. However, evaluating the adversarially-trained models against a wider variety of

[1] https://github.com/jon1ran/PrototypeDNN-Robustness

adversarial attacks would provide a more comprehensive understanding of the true robustness offered by these techniques. This broader evaluation would help in obtaining a more realistic assessment of the effectiveness of the defense strategies in real-world scenarios.

Investigate Alternative Defense Methods Building on the insights from this study, future research should explore alternative defense methods beyond adversarial training. Our experiments have demonstrated that adversarial training improves robustness against adversarial attacks but often results in a trade-off, reducing performance on non-adversarial inputs. Identifying and developing other defense strategies could help mitigate this issue and provide more balanced performance, enhancing robustness without compromising accuracy on clean data.

Explore Formal Interpretability Metrics In future work, more formal metrics to evaluate model interpretability should be explored. While qualitative insights have laid the groundwork, standardized quantitative metrics are crucial for rigorous assessment. Alvarez et al. [7] and Nauta et al. [20] have proposed criteria such as fidelity, diversity, and grounding. However, a combination of metrics might be necessary to capture the complexity of interpretability. Future research should focus on refining existing metrics and developing new ones to quantify comprehensibility, stability, and purity of model explanations.

Bibliography

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, February 2014. arXiv:1312.6199 [cs]. See pages 8, 9.

[2] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial Examples: Attacks and Defenses for Deep Learning, July 2018. arXiv:1712.07107 [cs, stat]. See page 8.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples, March 2015. arXiv:1412.6572 [cs, stat]. See pages 10, 41.

[4] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks, September 2019. arXiv:1706.06083 [cs, stat]. See pages 10, 35, 39, 42, and 52.

[5] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A Survey on Neural Network Interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, October 2021. arXiv:2012.14261 [cs]. See page 12.

[6] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, pages 3530–3537, New Orleans, Louisiana, USA, February 2018. AAAI Press. See pages 12, 15, 16, 27, 28, 29, 30, and 33.

[7] David Alvarez Melis and Tommi Jaakkola. Towards Robust Interpretability with Self-Explaining Neural Networks. In *Advances in Neural Information Processing Systems*, volume 31, page 10, Montréal, Canada, 2018. Curran Associates, Inc. See pages 12, 59.

[8] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. See pages 12, 13, and 19.

[9] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, April 2014. arXiv:1312.6034 [cs]. See page 13.

[10] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity Checks for Saliency Maps, November 2020. arXiv:1810.03292 [cs, stat]. See page 13.

[11] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV), June 2018. arXiv:1711.11279 [stat]. See page 13.

[12] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks, November 2013. arXiv:1311.2901 [cs]. See page 13.

[13] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object Detectors Emerge in Deep Scene CNNs, April 2015. arXiv:1412.6856 [cs]. See page 13.

[14] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable Convolutional Neural Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, Salt Lake City, UT, June 2018. IEEE. See pages 13, 14.

[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-Precision Model-Agnostic Explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. Number: 1. See page 14.

[16] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives, October 2018. arXiv:1802.07623 [cs]. See page 14.

[17] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. See pages 17, 24, 27, and 33.

[18] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, August 2003. See page 28.

[19] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 374(2065):20150202, April 2016. See page 51.

[20] Meike Nauta, Jörg Schlötterer, Maurice Van Keulen, and Christin Seifert. PIP-Net: Patch-Based Intuitive Prototypes for Interpretable Image Classification. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2744–2753, Vancouver, BC, Canada, June 2023. IEEE. See page 59.