University of Sheffield

Adversarial Attacks on Neural Networks

Kamil Topolewski

Supervisor: Robert M Hierons

This report is submitted in partial fulfilment of the requirement
for the degree of BSc Computer Science by Kamil Topolewski

May 11, 2022

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Signature: Kamil Topolewski

Date: May 11, 2022

Abstract

In recent years, machine learning has been rapidly advancing and becoming ubiquitous in our everyday world. Implementing Deep Neural Networks in areas like image recognition and malware detection enabled the creation of innovative applications. Despite its exceptional performance, it is criticized for its lack of ability to rationalize its predictions. What is more, they are vulnerable to adversarial inputs - small intentional perturbations of an image. Despite extensive research in this field, very few projects presented an effective method of preventing against adversarial attacks. In this project, the Deep k-Nearest Neighbour is examined - a hybrid classifier of Neural Network with the k-nearest neighbour algorithm that estimates the nonconformity of predictions. This work proposes contributions, which improve its performance, widen possibilities and adapt to real-life applications.

Contents

1 Introduction **5**
- 1.1 Motivation . . . 5
- 1.2 Report Structure . . . 5

2 Background **7**
- 2.1 Image Classification . . . 7
- 2.2 Machine Learning . . . 8
- 2.3 Artificial Neural Networks . . . 8
  - 2.3.1 Perceptron . . . 8
  - 2.3.2 Activation functions . . . 9
  - 2.3.3 Feedforward Neural Networks . . . 10
- 2.4 Convolutional Neural Networks . . . 12
  - 2.4.1 Convolution . . . 12
  - 2.4.2 Pooling . . . 13
  - 2.4.3 Batch Normalization . . . 14
- 2.5 Adversarial Attacks . . . 14
  - 2.5.1 Adversarial Attacks Categories . . . 15
  - 2.5.2 Fast Gradient Sign Method . . . 16
- 2.6 Defense strategies against Adversarial Attacks . . . 16
- 2.7 Deep k-Nearest Neighbours . . . 17
- 2.8 k-Nearest Neighbours . . . 17
  - 2.8.1 Approximate Nearest Neighbors Algorithms . . . 17

3 Design **19**
- 3.1 Dataset . . . 19
  - 3.1.1 Dataset pre-processing . . . 19
- 3.2 CNN Model . . . 19
- 3.3 Neurons' Activations . . . 20
- 3.4 Adversarial Examples . . . 20
  - 3.4.1 Adversaries Class . . . 21
- 3.5 Approximate Nearest Neighbors Algorithms . . . 21
- 3.6 Deep k-Nearest Neighbours . . . 22

4 Evaluation **25**
4.1 Baseline . . . 25
4.1.1 CNN Model . . . 25
4.1.2 Neurons' Activations . . . 25
4.1.3 Adversarial Attacks . . . 26
4.1.4 Approximate Nearest Neighbors Algorithms . . . 26
4.1.5 Deep k-Nearest Neighbours . . . 28
4.2 Experiments . . . 31
4.2.1 Alternative Approximate Nearest Neighbours Algorithms . . . 31
4.2.2 Alternative Deep Neural Network Architecture . . . 33
4.2.3 Evaluation of different dataset . . . 35

5 Discussion **37**
5.1 Advantages of the Deep k-Nearest Neighbours . . . 37
5.2 Limitations . . . 37

6 Conclusion **38**

A Fashion MNIST Results **39**

Chapter 1

Introduction

In the last decade, thanks to deep neural networks (DNNs), we have seen a significant improvement in machine learning algorithms. Since GPU (graphics processing units) were developed, computational powers increased thousands of times over ten years, which made deep learning extremely powerful. They are used in a variety of fields like weather predictions, Gmail smart sorting and different types of recognition.

1.1 Motivation

With the incoming rapid growth of deep learning, attacks on this machine learning could potentially have enormous consequences. State-of-art neural networks are so accurate at certain tasks that numerous companies use them to automate their work. For instance, some banks use them to process checks, post offices for package addresses, and self-driving cars for detecting road signs. A potential threat arises - what if an attacker could purposely make a DNN misclassify an image? What is more, executing it with an undetectable difference to a human observer?

The above-explained phenomenon is known as adversarial attacks[1]. They are inputs intentionally designed to cause misclassification of an image. By adding a relatively small amount of perturbation to an image, almost invisible to a human eye, neural networks (NN) may completely misrecognize an object. This attack can be performed at many machine learning algorithms: deep learning as well as simple linear classifiers[2].

In this project, I will focus on detecting adversarial attacks and ways of preventing them.

1.2 Report Structure

- **Chapter 2: Background.** Includes an overview of the concepts and literature study carried out for the purpose of the project.
- **Chapter 3: Design.** Explains steps taken to develop the Deep k-Nearest Neighbours and the structure built for its evaluation.

- **Chapter 4: Evaluation.** Provides testing of the baseline model and introduces novel approaches.
- **Chapter 5: Discussion.** Section explaining the limitations and future work.
- **Chapter 6: Conclusion.** A summary of the results and conclusions carried out in the project.

Chapter 2

Background

For the purposes of this project, a literature survey was carried out on fields relevant to the subject.

2.1 Image Classification

One of the primary tasks in computer vision is image classification. It refers to extracting information about an image. Usually, we try to assign a label to input from a fixed span of categories. This functionality has a variety of real-life applications like recognizing road signs, object identification in a satellite image or reading handwritten characters (Figure 2.1).

For humans, image classification is a trivial task to perform. Our brains can effortlessly adapt to variations which are highly challenging for a computer algorithm. The main difficulties are:

- **Viewpoint.** To the camera, an object can be perceived differently, depending on its orientation. Regardless of rotation in any dimension, it still has to be classified correctly into the same category.
- **Intra-class.** Categories may be broad in their representation. "Car" label should be attached to a Micro vehicle as well as a Van, even though they look very distinct from each other.

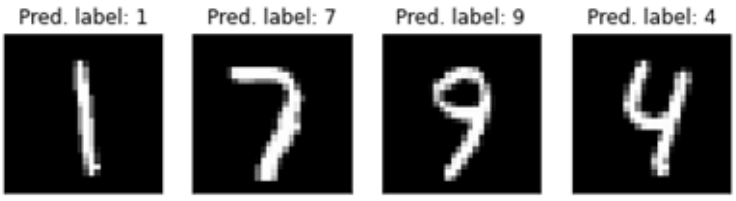

Figure 2.1: Image classification of MNIST handwritten digits dataset.

- **Deformation.** The same object can appear different due to real-life damage or deformation caused by the poor quality of an image.
- **Occlusion.** Detecting all features may be an issue due to occlusion, only a limited portion of an object may be visible to the camera.
- **Scale.** Same class objects generally exhibit many variations in their size. Dependent on the camera placement, the magnitude can change substantially.

All of these factors contribute to the difficulty of creating an algorithm to classify new images.

2.2 Machine Learning

Arthur Samuel was one of the pioneers of Machine learning (ML) with his research on the game of checkers[3]. The Samuel Checkers-playing program, created in 1959, was one of the first self-improving algorithms. It was a demonstration of computer capabilities which later became the fundamentals of Artificial Intelligence (AI).

The primary concept of ML is the ability to improve without getting specific instructions. They usually apply a set of techniques which analyze an extensive amount of data. ML can improve automatically with the use of data and experience.

Since the image data for image processing is unstructured, ML has been found efficient for its classification.

2.3 Artificial Neural Networks

Artificial Neural networks (ANN) are an essential part of ML algorithms that are capable of solving complicated problems. They are composed of artificial neurons and layers, inspired by the human brain, trying to imitate biological neurons signalling each other. The main difference between ML and Neural Networks (NN) is the ability to detect patterns by themselves during training. NNs are built from layers which allow solving complex problems by breaking them down into simpler ones. Using Perceptrons, clustering and classification, they are able to interpret images.

2.3.1 Perceptron

Motivated by earlier research on a mathematical model of the human brain[4], Frank Rosenblatt proposed the idea of a perceptron[5]. Perceptron is the first mathematical representation of an artificial neuron, which gives ANN the ability to learn. These neurons calculate a binary output based on a bias, input values and their weights. A simple perceptron model is presented in Figure 2.2. And mathematically perceptron can be defined with the formula 2.1.

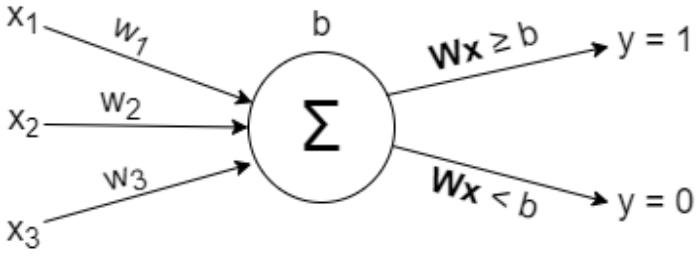

Figure 2.2: Model of perceptron.

$$y = \begin{cases} 1, & if \quad \mathbf{Wx} \geq b \\ 0, & if \quad \mathbf{Wx} < b \end{cases} \tag{2.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight vector, b is bias and y is output.

Assuming a neuron output is a 0 or 1 decision, high-level intuition behind the neurons' parameters could be explained as follows:

- **Inputs.** Representations of factors on which the decision depends.
- **Weights.** Describe significance of each input.
- **Bias.** A parameter that moves the decision boundary away from the origin.

A slight alteration in the biases or weights could change the output of perceptron from 0 to 1, which can result in poor performance of a model. That is why NN uses activation functions.

2.3.2 Activation functions

Activation functions are necessary and fundamental for every NN. On each layer of the model, different activation functions can be used, which will impact the performance of NN. They not only solve Rosenblatt's perceptrons limitations but also add non-linearity to our model[6]. Output from such a neuron can be defined with formula 2.2.

$$y = f(\mathbf{Wx} + b) \tag{2.2}$$

where f is the activation function, $\mathbf{W}$ is weights vector, $\mathbf{x}$ is inputs vector and b is bias.

Following activation functions have been used in this project:

ReLU

Rectified Linear Unit(ReLU) is one of the most common and effective activation functions in ANN. It consists of two linear fragments, therefore, it retains properties of linear functions,

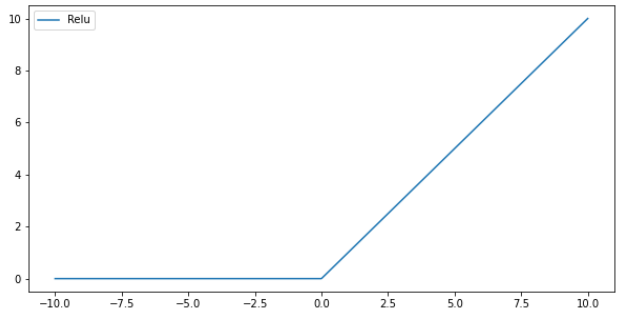

Figure 2.3: Graph representation of the ReLU function

which helps optimizations with gradient descent methods[7]. For every input, ReLU checks a threshold value: if it is below 0, it takes the value of 0, whereas values bigger or equal to 0 remain the same. ReLU function can be represented with the formula 2.3 and a graph 2.3.

$$f(x) = max(0, x) = \begin{cases} x, & if \quad x \geq 0 \\ 0, & if \quad x < 0 \end{cases} \tag{2.3}$$

Softmax

Softmax is a popular activation function for multiclass classification. It takes a vector of numbers and outputs probabilities, which are scaled relative to each value in the vector. The softmax function is often used to compute the normalized output vector of a model. These values then define the probability of the input belonging to each class, often referred to as confidence of the predictions. Softmax function is represented with equation 2.4

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad for\ i = 1, 2, \ldots, n \tag{2.4}$$

where z represents values of the output vector and n is the number of elements in output vector.

2.3.3 Feedforward Neural Networks

On their own, neurons are only capable of making a single decision. Therefore, taking the human brain as inspiration, we connect neurons together. This way, they form a column, which we call a layer. However, a single layer of neurons would be able to only classify linearly solvable problems. The answer to this issue is adding additional layers, called hidden layers, between input and output, which creates Feedforward Neural Network (FNN). Feedforward refers to the lack of loops between the nodes and information moving only forward. A sample depiction of such network is shown in Figure 2.4. FNN with multiple hidden layers is called Deep Neural Network (DNN), a term coined by Rina Dechter[8].

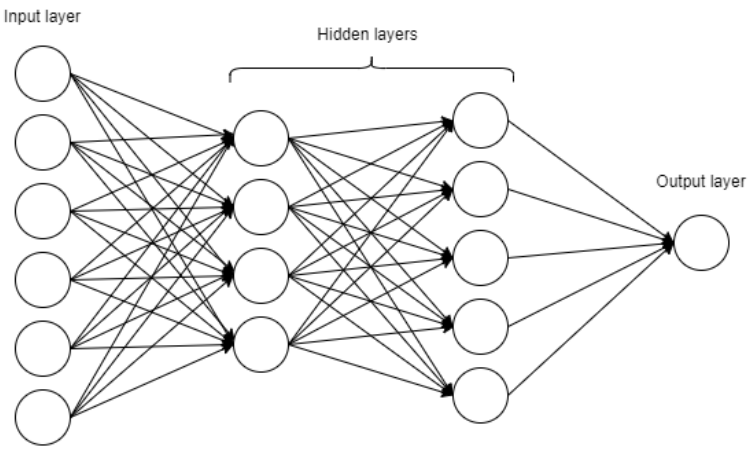

Figure 2.4: A sample structure of a NN.

Forward-propagation

Provided input into the first layer, DNN can propagate the data forward to calculate a prediction. Each layer receives data from the previous layer and processes it with respect to its activation functions, weights and biases. The last layer outputs a prediction, which for image classification, is a probability distribution using softmax activation. However, the forward-pass is not enough for a model to learn. The fundamental concept that allows NN to learn is through back-propagation.

Back-propagation

Back-propagation is the core concept necessary for training a NN. As training, we refer to a process of adjusting weights and biases. These parameters are usually initialized with random values, therefore, the first prediction of the model will be completely random. After a prediction is made, we calculate a loss function, which measures an error of that prediction in respect to true label. The most common loss function for the image classification model is cross-entropy. Given output probabilities for all classes, cross-entropy calculates divergence between the output and true class distribution, according to formula 2.5. Hence, as predicted probability diverges from the true label, the cross-entropy loss increases. Whereas loss equaling zero would indicate a faultless prediction.

$$L = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.5}$$

where M is the number of classes, y is the label of class c for an observation o and p is probability of observation o belonging to class c.

Given a loss, weights and biases can be adjusted by minimizing the loss function. Finding the explicit minimum of a loss function efficiently is not feasible due to the enormous amount

of parameters[6]. Therefore, NNs try to find a local minimum of a loss function using an optimization algorithm called gradient descent. The intuition behind gradient descent is iteratively moving towards the negative gradient of the function. The size of these iterative steps is called the learning rate. Since the slope of the function at each point is different, the gradient has to be recalculated at each step. This way NNs eventually arrive at a minimum, even though it is not necessarily a global minimum. A high learning rate allows to move faster through the function, however, we risk overshooting. A small learning rate is more precise but computationally more demanding. To efficiently locate a minimum adaptive moment estimation (Adam) can be used[9]. Adam is an optimizer that allows to adaptively change learning rate during the training with respect to current weights and biases. The algorithm applies an idea of momentum, computed by exponentially moving averages of a gradient. In practice, DNNs usually update the weights and biases after a defined number of predictions is made. This set of predictions is called a batch. This approach avoids taking noisy steps, due to more stable gradient descent convergences. The last hyperparameter essential to a training process of a NN is the number of epochs. It defines the number of iterations the entire dataset will be processed during training. It has to be large enough to sufficiently learn, however, prevent overfitting of training data. Overfitting refers to a concept when a model is too closely aligned to the training data, causing underperformance on unseen data.

Taking advantage of hidden layers, DNNs can process hierarchical representation of data in increasing abstraction. This approach allows for solving very complex problems[10], although it is computationally demanding and relies on a sufficient training dataset. Technological progress in recent years has contributed to exponential data growth and computer power improvements, which allowed to train DNNs faster and perform with very high accuracy.

2.4 Convolutional Neural Networks

One of the classes of NN is Convolutional Neural Network(CNN), which is well-known for its high performance in image classification. The name comes from a linear operation between vectors called convolution. They were inspired by findings in the biological vision field on mammals' visual cortex[11]. The research discovered that some neurons are active when exposed to watching lines at specific angles. Whereas some neurons would respond to a line moving in an explicit direction. The intuition behind the CNN can be interpreted as learning features in increasing abstractions. The first layer could detect edges, whereas the next one might build representations of features using those edges. The fundamental adjustments introduced by CNN are convolution and pooling.

2.4.1 Convolution

The role of the convolution layer is to identify multiple patterns within the sub-region of the input using receptive fields. Convoluting is a linear operation between input and matrices, called kernels or filters. Computing dot products between the kernel-sized segments of the

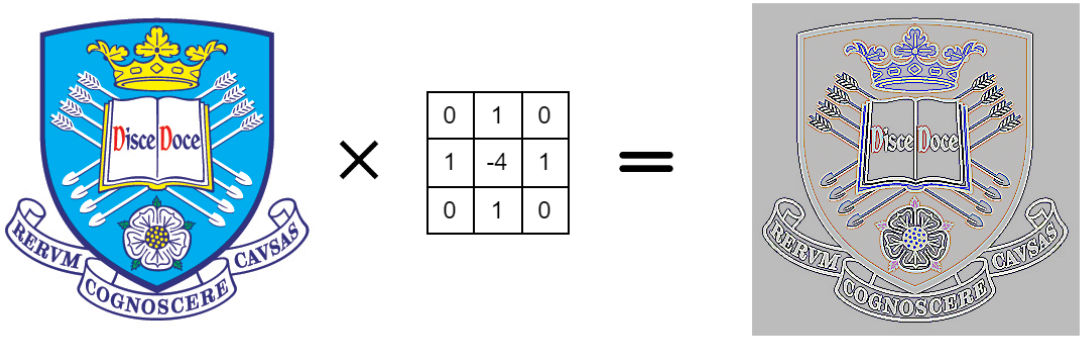

Figure 2.5: Application of 3x3 edge detection filter to an image.

input yields an activation map. Each kernel is meant to capture a specific feature of an image. For instance, a filter emphasizing vertical lines in an image is presented in Figure 2.5.

The convolution has three main parameters that can be adjusted: filters amount, kernel size and stride.

- **Filters amount.** The number of filters will decide on the dimensionality of the output space, therefore has to be adapted to computational power. Layers *deeper* into the network tend to have more filters, because the level of abstraction is higher, which makes it easier to capture features.
- **Kernel size.** It is a tuple specifying the height and width of the 2D kernel matrix. Greater size kernels help reduce volume size and learn larger spatial filters.
- **Stride.** It is a tuple which sets the number of pixels the filter should move. For instance, if the stride is (2,2), the filter is applied to the current location and takes two-pixel steps before it is applied again.

2.4.2 Pooling

Processing raw images is computationally intensive due to their size. Picture taken by an iPhone 11 has a resolution of twelve megapixels, which transfers to twelve million data points. Furthermore, most photos consist of three colour channels, which results in 36 million inputs. The role of the pooling layer is to reduce this amount, avoiding the loss of features essential to accurate prediction.

One of the types of pooling is max pooling. This technique uses a kernel (similar to convolution kernels) to extract the maximum value from a portion of an image. Moving across the entire image, max pooling builds a reduced representation of an image. A single max pooling operation is presented in Figure 2.6. On top of decreasing the size of the photo, it extracts the main features.

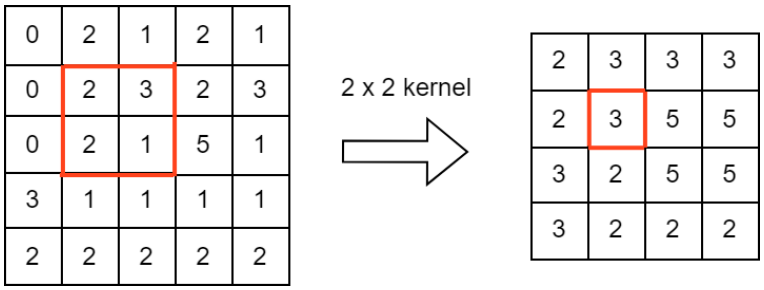

Figure 2.6: Max pooling a 5x5 image with a 2x2 kernel.

2.4.3 Batch Normalization

If the distribution of the images activations changes substantially during the training, CNN might encounter the internal covariate shift[12]. Batch Normalization alters the activations to prevent significant distribution change, which might happen due to parameter adjustments from each batch. Otherwise, CNN might experience overfitting and diminishing gradients for a high learning rate, which decelerates the training.

2.5 Adversarial Attacks

As mentioned in Chapter 1, adversarial examples are malicious images created with the purpose to fool DNN.

Many classes and state-of-art models of machine learning have been shown to be vulnerable to adversarial attacks[13][14]. When first discovered, speculative explanations were suggesting that is due to the extreme nonlinear nature of deep networks or a form of over-fitting. However, it was discovered that even linear models fail against adversarial attacks[15]. In many cases, models with different architectures, trained on different sets of data, can misinterpret the same image[16]. This proves that it is not a 'bug' within a particular machine learning algorithm, but an intrinsic blind spot. We expect models not to be much affected by the small noise, since they do not change the image category. However, a small amount of precisely calculated, but hardly perceptible, perturbation added to an image, causes complete misclassification.

In the figure 2.7, the DNN correctly classifies the raw image (most left picture) as a panda with 58% confidence. The middle image represents a distortion generated by an adversarial attack. It may appear as random noise, however, it is a carefully computed structure meant to fool the DNN. The distortion is added to the raw image to generate an adversarial example (most right picture). To a human observer, pandas' images are indistinguishable. In fact, on a basic computer monitor, these two images are exactly the same. A standard monitor can display only up to 8 bits of colour resolution, and DNN works with 32. In the image on the right, the distortion happens to be too small to affect those 8 bits. After executing the Adversarial Attack, the model misclassifies the image as a gibbon with 99% confidence.

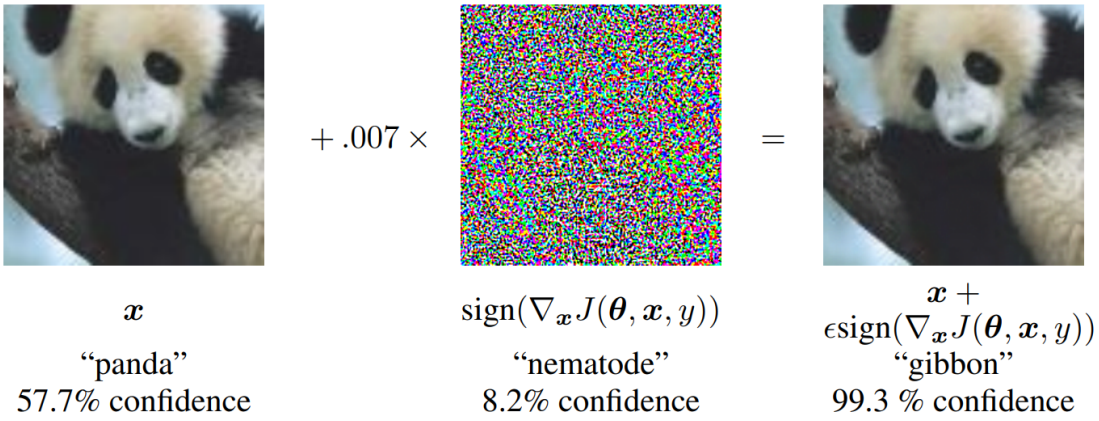

Figure 2.7: FGSM Adversarial Attack applied to an image of panda[2].

2.5.1 Adversarial Attacks Categories

Based on the attacker's outcome intentions, adversarial attacks can be categorized into the following groups:

- Targeted - attackers force the model misclassification to some specific target prediction.
- Non-targeted - the only goal is to cause predicting incorrect label.

Additionally, attacks are split into groups based on the knowledge that attackers have of the underlying neural network model:

- White box - the attacker is all-knowing. He has information on the model's type and structure, and all of the parameters and weights.
- Black box - the attacker has no knowledge of the underlying model. He only has access to the inputs and outputs of the algorithm.

The last distinction is based on how data is provided to the model:

- Digital attacks - attacker has access to each data point entering the algorithm. Which are arrays of float32 values. That allows the attack to be very specific and precise. An attacker can carefully change tiny details in a JPG or PNG file and feed it into the machine learning algorithm.
- Physical attacks - this usually corresponds to an attack in the physical world where the input is an image captured by a sensor (like a camera). That involves placing a real object in a physical environment. For example, an attacker sets a sign next to a road which is later captured by a self-driving car. Despite attacks usually involving precisely calculated perturbations, it has been discovered that adversarial attacks can be successfully performed on real-life inaccurate sensors[17].

This project will focus on digital, white-box, non-targeted attacks. A variation that is the most heavily researched at the moment. A popular example of a white-box non-targeted attack is the Fast Gradient Sign Method.

2.5.2 Fast Gradient Sign Method

One of the first and most popular attacks is the Fast Gradient Sign Method (FGSM) [2]. It takes advantage of gradient descent - an optimization algorithm for minimizing the loss function. As explained in section 2.3.3, gradients indicate the directions in which the local minimum is placed. Then CNN executes back-propagation to determine how to adjust the weights. FGSM uses the same concept, except using maximum loss. Then back-propagates from the output layer to the input image and alters the pixels. In other words, instead of adjusting the weights to reduce the loss function, we change the image to maximize the error. An adversarial image is constructed according to the formula 2.6 and visually in Figure 2.7. Epsilon is a parameter controlling the intensity of the perturbation added to the adversarial image. High ϵ increases the probability of misclassifying an image, although the perturbations are more noticeable.

$$X_{adv} = X + \epsilon.sign(\nabla_X J(\theta, \mathbf{x}, \mathbf{y})) \tag{2.6}$$

where:

ϵ - parameter controlling the intensity of distortion
∇ - the gradient
J - the loss function
$\mathbf{x}$ - the model's input image
$\mathbf{y}$ - the model's output predictions

2.6 Defense strategies against Adversarial Attacks

Trying to solve the problem of adversarial attacks is difficult for many reasons. Due to its open-ended nature, finding a defence against one attack is not sufficient. This can be observed by following the latest research papers proposing different solutions. These studies usually prevent against a particular attack, however, most of them are later exploited with new adversarial examples. For instance, using a second neural network, trained specially to detect adversarial attacks[18], but that was later bypassed by adaptive attacks[14]. Or using defensive distillation which was considered a complete problem solution[19], however only for a year, when a new attack was discovered, which was able to fool it, with a 100% success rate[20].

A promising defence strategy, introduced by Nicolas Papernot and Patrick McDaniel in 2018, is the Deep k-Nearest Neighbours[21]. Instead of using DNNs' softmax output as credibility, the algorithm measures the nonconformity of the prediction with the training data.

2.7 Deep k-Nearest Neighbours

Deep k-Nearest Neighbours(DkNN) introduces a notion of credibility, which defines certainty in the prediction made by a CNN. Softmax predictions (explained in Section 2.3.2) are not a reliable metric of prediction's confidence[22]. Proof of that thesis can be seen in the Figure 2.7, where a CNN assigns a 99.3% confidence to an incorrect prediction of an adversarial example, whereas only 57.7% for the original image with the correct label. Furthermore, a softmax function returns a probability distribution over all of the classes, therefore, given a random noise, it still may output very high confidence. This also happens for out-of-distribution data - images that do not belong to the dataset and do not represent any of the classes.

The core assumption behind DkNN is that for the same class, the 'thinking process' of a CNN that leads to the prediction should be similar. Hence, CNN's intermediate computations of an unseen image of class c should be closely related to intermediate computations of images of class c observed during the training. In practice, intermediate computations refer to neuron activations of the NNs' layers and comparison is done using the k-nearest neighbours search. Then an analysis of found neighbours labels is conducted, which establishes the nonconformity of intermediate calculations of DNN. Then nonconformity of a sample is compared with a nonconformity of a calibration set, which yields the final credibility.

2.8 k-Nearest Neighbours

K-Nearest Neighbours(k-NN) is a fundamental machine learning algorithm that can be used in supervised classification. Given an input, k-NN searches for a specified number of samples (k) closest to the query by comparing distances. Standard k-NN has many advantages. It is instance-based - learns the training examples and then classifies new data points based on a similarity measure, therefore does not require any additional training. Furthermore, we can add new data points to the dataset at any point. However, it becomes very inefficient on large datasets, which is the case for DkNN. As the number of data points and features increases, it becomes significantly slower, what we call the curse of dimensionality[23]. This occurs due to inputs' distances becoming less distinctive and search area increasing. For a naive k-Nearest Neighbours, the time complexity of a single query is $\mathcal{O}(n\ d)$, where n is the number of samples in the index, and d is their dimension. To reduce the computational complexity of k-NN, we can use data reduction or approximate nearest neighbours algorithms.

2.8.1 Approximate Nearest Neighbors Algorithms

Locality-Sensitive Hashing

Locality Sensitive Hashing (LSH) is a well-known family of algorithms used in similarity searches for high-dimensional data. A system for finding and ranking images from Google called VisualRank uses LSH and PageRank, for analysis and comparison[24]. In order to find the nearest neighbours, LSH distributes data points into buckets. This is achieved using a hash function, although it is unlike a cryptographic hashing function. Instead of

minimizing collisions between outputs, it maximizes them. Hence, given a query data point, LSH produces a hashcode based on which query is distributed into a bucket. Then, finding the nearest neighbours only among the bucket is immensely more efficient than the entire data set.

Vector Encoding Using Trees

Another technique used for Approximate Nearest Neighbors is tree-based algorithms [25]. At each intermediate node in the tree, the algorithm uses random projection[26] to reduce the high-dimensionality vectors. The random projection method chooses two arbitrary data points and then splits by the hyperplane equidistant from them. The hyperplane becomes a decision boundary that separates the data space into two subspaces of one less dimension. This procedure splits the dataset recursively until we reach the specified number of nodes. Then we end up with a binary tree representing the entire data space. The assumption is that similar data points likely belong to the same node (no hyperplane separates them). For a given query, the algorithm traverses the tree from the root to the final node. On each intermediate node it decides which side of the hyperplane to explore further until it reaches a leaf (node with no children), at which it executes the final search.

Graph based k-NN

Graph-based methods execute a local search over a proximity graph. A proximity graph is a set of nodes connected by edges, constructed by measuring the distances between each vector. These edges represent connections between nodes that are the most similar to each other. Then we can use the graph for finding the nearest neighbours of a query by following the algorithm:

1. Choose an arbitrary node and add it to an array of candidates.
2. Look at all nearest neighbours (nodes connected by an edge) of the first untried candidate in the candidate array.
3. Add all of these neighbours to the candidate array.
4. Sort the candidate array by similarity to the query.
5. Discard candidates in the candidate array where the index is bigger than specified number of neighbours.
6. If all candidates in the array have been tried, return to the candidate array, otherwise return to step 2.

Chapter 3

Design

In this section, system design and development will be discussed. The implementation was designed in order to recreate the original DkNN research[21] and provide a structure for agile experiments using classes.

3.1 Dataset

The images used for this study are taken from the MNIST dataset[27]. It is a substantial database of grayscale handwritten digits from 0 to 9. Digits are ideal for pattern recognition and MNIST provides predefined labels, which are necessary for supervised learning. Due to redundant features, it is a perfect dataset to validate the implementation of DkNN and experiment on.

3.1.1 Dataset pre-processing

Achieving high accuracy DNN relies on a consistent format of input images. MNIST dataset provides images with pixel-level [0, 255], although most CNN work with float points, therefore dataset is normalized to values [0, 1]. Scaling images also undermine the impact of illumination's differences. Then the dataset is split into three subsets:

- Training set - 60,000 (86%) of images is devoted to training the DNN.
- Test set - 9250 (13%) is used towards testing.
- Calibration set - 750 (1%) is committed to calibrating DkNN nonconformity.

3.2 CNN Model

For training neural networks, the TensorFlow framework is used - an open-source software library for developing machine learning models. Choosing the correct model architecture is very important for high accuracy. However, in order to recreate the performance from the

Layer	Filters	Kernel size	Strides	Padding	Activation
Conv 1	64	8x8	2x2	1	ReLU
Conv 2	128	6x6	2x2	0	ReLU
Conv 3	128	5x5	1x1	0	ReLU
Dense	-	-	-	-	Softmax

Table 3.1: DNN architecture of the model used in original DkNN paper[21].

original paper, the same model architecture was used. Detailed architecture is presented in Table 3.1.

3.3 Neurons' Activations

Extracting neuron activations is essential for approximate nearest neighbours. This is done using a built-in Keras function. The dimension of the output of a convolutional layer can be described with Formula 3.1. The 3-dimensional output activations have to be converted into a 1-dimensional vector. The final size of the vector is the product of the output dimensions.

$$(x_2, y_2) = (Ceil\frac{x_1 + p - k_1 + 1}{s_1}, Ceil\frac{y_1 + p - k_2 + 1}{s_2}, n) \tag{3.1}$$

where:

(x_1, y_1) - dimensions of the previous layer's image dimensions.
(k_1, k_2) - the convolution kernel matrix.
(s_1, s_2) - the stride matrix.
p - the number of padded pixels in each axis.
n - the number of filters.
Ceil - ceiling round a decimal.

3.4 Adversarial Examples

To generate the adversarial dataset, FGSM adversarial attack was applied to the original MNIST test set. This technique is described in Section 2.5.2. Perturbations are created using a Tensorflow built-in library and added to the MNIST dataset. Six levels of perturbation intensity ε were chosen to capture how the performance changes progressively. Since the test set consists of 9,250 data points, 55,500 adversarial attacks will be generated across all ε . Since the FGSM attack relies on the model's architecture, for each model separate adversarial examples need to be computed. Perturbating this amount of images is time-consuming. To streamline the process, adversarial examples for each model and epsilon are stored using python's pickle module.

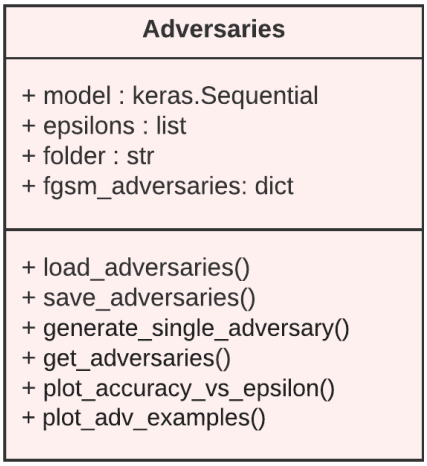

Figure 3.1: Parameters and functions of Adversaries class.

3.4.1 Adversaries Class

During testing and experiments, adversarial examples are used numerous times. To manage them more efficiently, the Adversaries class was created, and each set of adversarial examples is initialized as its objects. Figure 3.1 shows the parameters and functions of the class. The class provides an implementation for generating FGSM attacks. However, this process is time-consuming since the data have to go through a similar process as images during training. Therefore, functions for loading and saving adversarial examples have been created. Adversaries class also provides the ability to plot adversarial examples, thus the visibility of perturbation can be examined. The second plot function allows visualizing how the accuracy of the model changes with respect to the increasing epsilon.

3.5 Approximate Nearest Neighbors Algorithms

In the DkNN paper, Multiprobe Locality Sensitive Hashing[28] has been chosen as the approximate nearest neighbours algorithm. Despite LSH's (introduced in Section 2.8.1) immense time complexity improvement over standard k-Nearest Neighbours search, it is still not efficient enough for fast indexing and query for such high-dimensional data. LSH samples a locality-sensitive hash function, then for each unique hash, it creates a bucket and lastly arranges data into these buckets. Multiprobe LSH is based on standard LSH, however, it uses multiple hash functions. Then buckets represent tuples of cells corresponding to each hash function. The hashing process is repeated several times, producing a specified number of tables. Then the algorithm queries multiple tables at the same time and searches for candidate buckets. Data points in candidate buckets are then compared with the query with

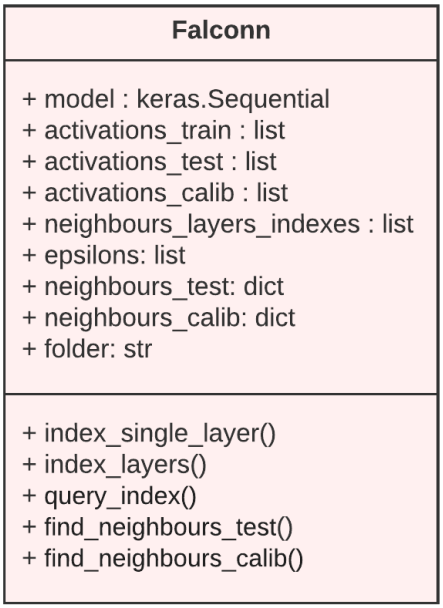

Figure 3.2: Parameters and functions for classes of approximate nearest neighbors algorithms.

angular distance.

For implementing Multiprobe LSH, Falconn library has been chosen[29]. Falconn is an LSH library developed by Ilya Razenshteyn and Ludwig Schmidt. In order to efficiently test different settings of approximate nearest neighbours algorithms, a class for each algorithm is declared. Figure 3.2 shows Falconn class structure. It provides an implementation of indexing the training dataset, which is necessary for nearest neighbours search. The class also includes functions for querying the test and calibration sets using the index. The number of nearest neighbours k has been set to 75, identical to the original DkNN.

3.6 Deep k-Nearest Neighbours

The Deep k-Nearest Neighbours algorithm (Introduced in Section 2.7) adds an additional procedure executed after prediction is made by the CNN. A test image is forward-propagated by the CNN, and activations of neurons on each layer are extracted. For each layer's activations, the approximate nearest neighbours algorithm is used to find k-nearest neighbours in the activations dataspace of the training dataset. Neighbours of each layer obtained by the function are put into a multi-set. Then using the multi-sets we can compute the nonconformity of the prediction, based on how many neighbours' labels match the label predicted by the model. Nonconformity of a single image can be defined with formula 3.2.

$$\alpha(x, j) = \sum_{\lambda \in 1..l} |i \in \Omega_\lambda : i \neq j| \tag{3.2}$$

where:

x - the input image
j - the true label of image x
Ω - the multi-set of training points' labels
l - the number of model's layers

To obtain the credibility of the new prediction, the algorithm compares the new prediction's nonconformity to the nonconformity of the predictions of the calibration set. A calibration set is a small subset of the original dataset of images. After the model is trained, calibration images go through the same DkNN process as test images, and nonconformities for true labels are computed. Then DkNN calculates the empirical p-value for each class, which is determined by dividing the number of calibration's nonconformity measures bigger than the test input's by the size of the calibration set. The equation for computing p-values is shown in the formula 3.3.

$$p_j(x) = \frac{|\{\alpha \in A : \alpha \geq \alpha(x, j)\}|}{|A|} \tag{3.3}$$

where:
x - the input image
j - the true label of image x
$\alpha(x, j)$ - the nonconformity of the input x for the label j
A - a set of nonconformity values of the calibration dataset

After computing empirical p-values for all classes, the highest one determines the label predicted by the DkNN and its credibility as the empirical p-value itself. The full algorithm of the DkNN explained in this section, taken from the original paper[21], is shown in the Algorithm 1.

For each Deep k-Nearest Neighbours model, an instance of DkNN class is initialized. Figure 3.3 shows the parameters and functions of the class. This class provides all necessary operations for the DkNN Algorithm 1. Furthermore, it implements additional functions allowing to visualize and compare performance. This includes a bar diagram comparing the correctness of neighbours on different layers and a reliability diagram. The latter visualizes accuracies with respect to predictions' credibilities (or confidence for softmax) and the number of data points in each category.

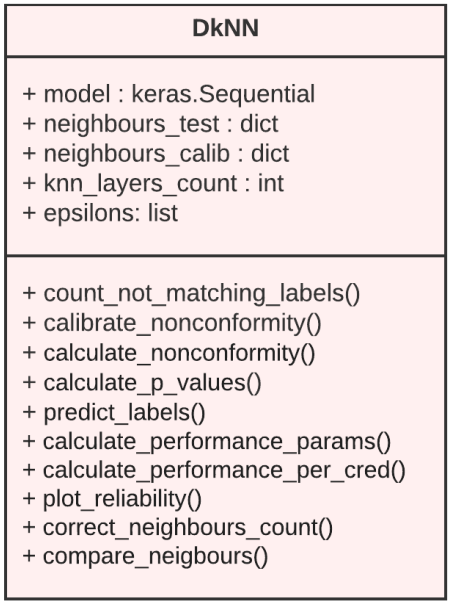

Figure 3.3: Parameters and functions of DkNN class.

Algorithm 1 - Deep k-Nearest Neighbour[21]

Input: training data (X,Y), calibration data (X^c,Y^c)
Input: trained neural network f with l layers
Input: number k of neighbors
Input: test input z

// Compute layer-wise k nearest neighbors for test input z
for each layer λ 1..l **do**
    $\Gamma \longleftarrow$ k points in X closest to z with LSH tables
    $\Omega_\lambda \leftarrow \{Y_i : i \in \Gamma\}$ ▷ Labels of k inputs found
end for
//Compute prediction, confidence and credibility
A = $\{\alpha$ (x,y) : (x,y) $\in$ (X^c,Y^c) ▷ Calibration
for each layer j $\in$ 1..n **do**
    $\alpha(z, j) \longleftarrow \sum_{\lambda \in 1..l} |i \in \Omega_\lambda : i \neq j|$ ▷ Nonconformity
    $p_j(z) = \frac{|\{\alpha \in A : \alpha \geq \alpha(z,j)\}|}{|A|}$ ▷ empirical p-value
end for
prediction $\leftarrow argmax_{j \in 1..n} p_j(z)$
confidence $\leftarrow 1 - max_{j \in 1..n, j \neq prediction} p_j(z)$
credibility $\leftarrow max_{j \in 1..n} p_j(z)$
return prediction, confidence, credibility

Chapter 4

Evaluation

In this section, baseline model is evaluated and novel approaches are introduced.

4.1 Baseline

A baseline DkNN was first implemented according to the the original DkNN paper[21]. This will allow for testing of the implementation and better comparison with novel approaches. Parameters that were not specified in the research, will be set experimentally in order to achieve results as similar as possible.

4.1.1 CNN Model

The CNN model suggested in the original paper is relatively simple, due to high complexity of k-nearest neighbours search. It consists of three convolutional layers with ReLU activation and a single dense layers with Softmax activation. Detailed architecture has been presented in Table 3.1.
Following hyper-parameters have been set:

- Optimizer - Adam, learning rate of $\alpha = 10^{-3}$.
- Loss - Categorical crossentropy
- Batch size = 500 has been found to be most optimal.
- Epochs = 8.

Batch size and epochs have not been implicitly specified in the paper. However, using the hyper-parameters above, similar accuracy of 99.1% was achieved (99.2% in the research).

4.1.2 Neurons' Activations

Neurons' activations are collected on all of the layers for each training sample. On the first layer, an image of dimension (28,28) is convoluted by 64 filters, therefore producing 64 new

Layer	Output dimension	Number of neurons
Conv 1	(14, 14, 64)	12,544
Conv 2	(5,5,128)	3,200
Conv 3	(1,1,128)	128
Dense	(10)	10

Table 4.1: The baseline CNN neurons outputs.

representations of the data. However, the image size is smaller due to the application of (8,8) kernel with stride (2,2) and 2 pixels per image axis. Using formula 3.1, after the first convolution the output has dimensions of (14,14,64), which is equal to 12,544 neurons per image. The rest of the outputs sizes is presented in Table 4.1

4.1.3 Adversarial Attacks

As mentioned the Section 3.4, the adversarial dataset is created via an FGSM attack. The main parameter of the attack is epsilon, which specifies the intensity of the perturbation. Six intensities have been chosen: [0.025, 0.05, 0.075, 0.1, 0.125, 0.15]. The ϵ of 0.0 means it is the original image, no perturbation was added. Figure 4.1 shows a sample of five images for epsilons [0.025, 0.1, 0.15]. For the lowest epsilon, the model is already misclassifying one of the digits. That digit is labelled as '9' and the raw image is similar to the digit '9'. Therefore vital property of adversarial attacks can be observed - closely related representations of labels are more likely to be misclassified. It is more demanding to cross the decision boundary for a label more distinct from other classes. Applying the highest ϵ fools the baseline network in all five cases. As the ϵ grows the perturbations are more noticeable, although a human can still accurately recognize the digit without difficulty.

The baseline model has been tested on entire adversarial dataset for each epsilon, its accuracy is shown in Figure 4.2. The model's accuracy of 99.1% for original images drops to 13% for the $\epsilon = 0.15$. Furthermore, the average softmax confidence of these predictions is 84%. This shows how effective and dangerous adversarial attacks are.

4.1.4 Approximate Nearest Neighbors Algorithms

Using the Falconn library activations of training samples are indexed. Indexing takes the longest time of the entire DkNN procedure, although can be done independently before querying the test set. The disadvantage of the Falconn library is the inability to download the index for future use. Then calibration, test and adversarial test sets are queried for neareast neighbours within training set dataspace. As explained in Section 3.5, Multi-probe LSH uses multiple hash functions in parallel and builds tables. The number of tables is set to 50 tables, which provides similar time efficiency as other approximate nearest neighbours algorithm compared in experiments (Section 4.2.1).

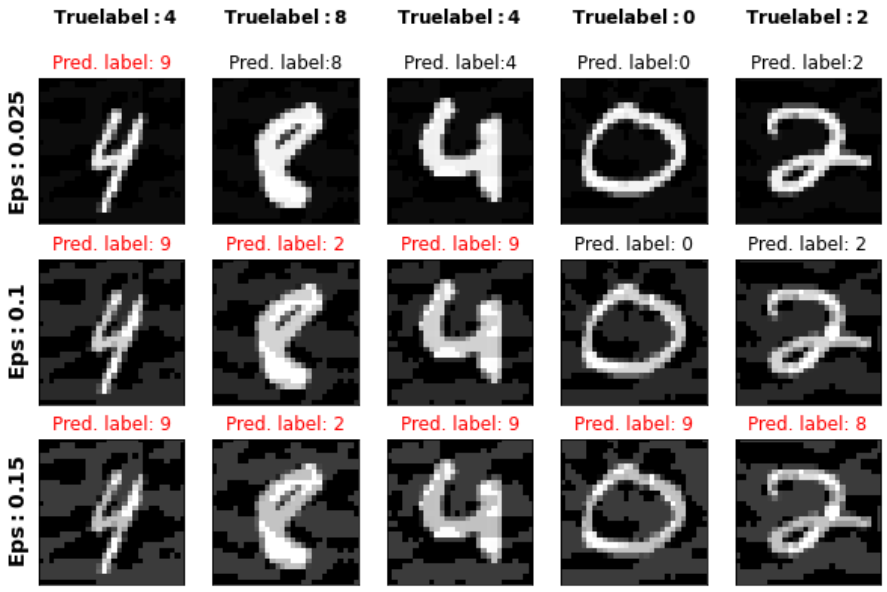

Figure 4.1: Sample set of FGSM advesarial examples, with increasing ϵ with each row. Each column represents different sample.

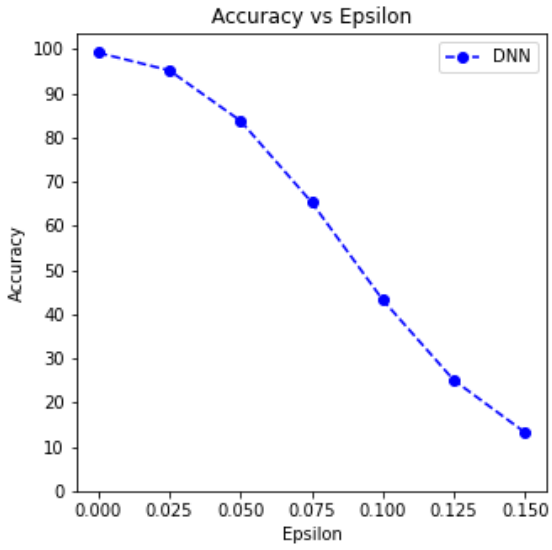

Figure 4.2: Accuracy of baseline model with respect to epsilon.

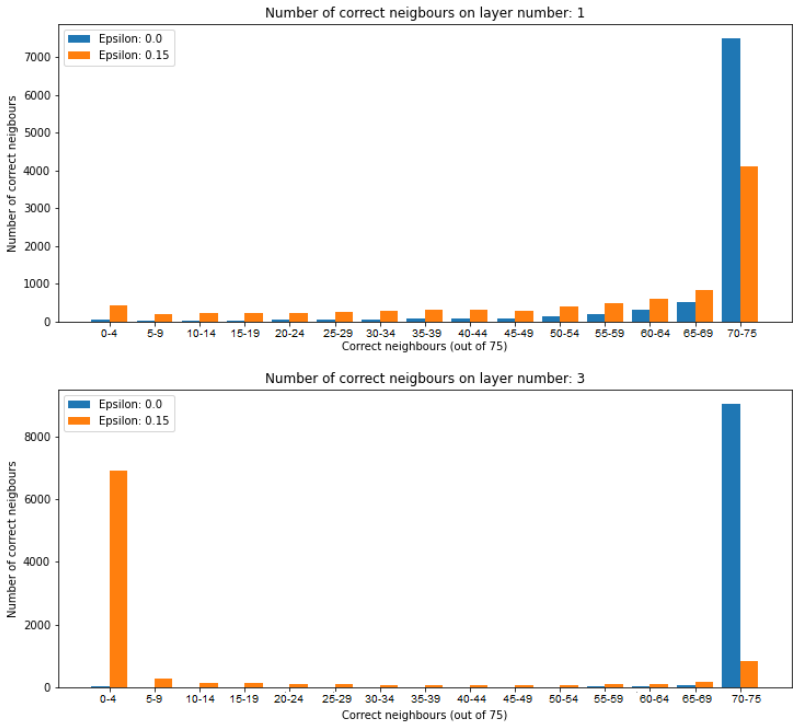

Figure 4.3: Correctness of neighbours on the first (the top figure) and third layer (bottom figure).

4.1.5 Deep k-Nearest Neighbours

Nonconformity

Using the neighbours found by the approximate nearest neighbours algorithm, DkNN calculates the nonconformity of calibration neighbours. As explained in Section 3.4.1, this is done by checking how many found neighbours do not belong to the correct label. Blue bars in top Figure 4.3 show the correctness of neighbours found for the non-adversarial dataset on the first layer. Around 80% of the images have between 70-75 correct neighbours, and the rest data points lie close to the highest category. Only 5% of the dataset has more than half of neighbours misclassified. Orange bars represent adversarial examples for $\epsilon = 0.15$. Around 44% of the samples are in the highest category. The other half of adversarial examples is almost evenly distributed throughout the rest categories classes. Therefore, executing the nonconformity on only the first layer is not enough to assign low credibility to adversarial examples. In bottom Figure 4.3 the same comparison is made on the third layer. In this case, contrast between datasets is definite. For $\epsilon = 0.0$ 98% of samples have the highest amount of neighbours, whereas for ϵ= 0.15 76% belong to the lowest category. Characteristic of higher contrast between neighbours' correctness in higher layers of CNN has been observed throughout the whole project.

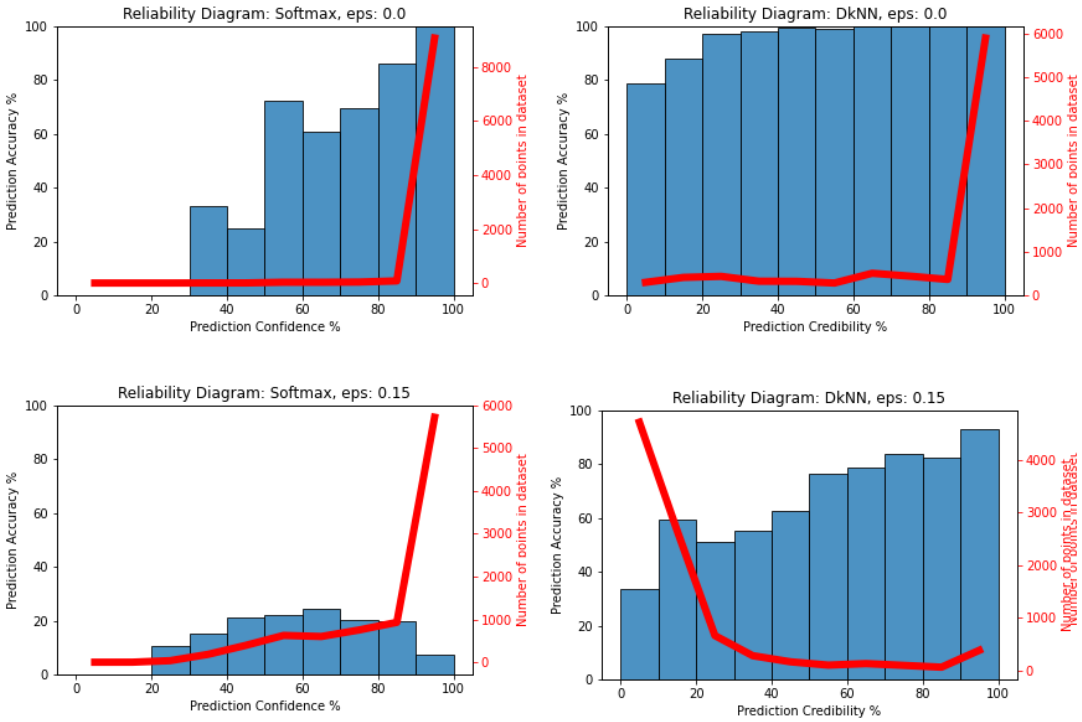

Figure 4.4: Reliability diagrams of CNN's Softmax confidence and DkNN's credibility.

Credibility

With the nonconformity of calibration data and approximated nearest neighbours of the test dataset, DkNN can predict credibility predictions using formula 3.3.

The reliability diagram in Figure 4.4 shows the performance of DkNN and Softmax on the original test set. Bars correspond to the left y-axis and represent the average accuracy of the model's predictions binned by their estimated credibility (DkNN) or confidence(Softmax). The red line corresponds to the right y-axis and indicates the number of data points belonging to each credibility category. One can expect that model's accuracy should increase linearly with credibility (or confidence) estimations. Ideally, the reliability diagram should approach a linear function. For the original images, the Softmax gives 90-100% confidence for almost all of the predictions. The accuracy rises with the confidence of the model, which can be interpreted as the correct linear relation, however, the accuracy below 90% has only several or no data points. For $\epsilon = 0.15$, where the images are no longer straightforward to classify, the accuracy-confidence relation does not hold. In fact, the model assigns 90-100% confidence to most of the predictions that are only 7% accurate.

For the DkNN and $\epsilon = 0.0$, the linear relation is more noticeable. The model assigns the lowest credibility to predictions which have an average accuracy of 78%. The distribution of samples is more even than Softmax, although most of the examples and in the 90-100% credibility category as well. The highest contrast between DkNN and Softmax model is visible

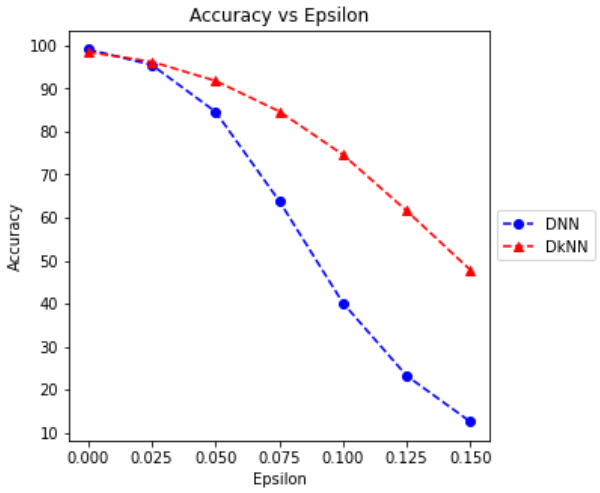

Figure 4.5: Accuracy of DkNN and baseline model with respect to epsilon.

for adversarial samples. DkNN's diagram not only approaches a linear function but also classifies most of the examples with extremely low credibility. Majority of these predictions have a 33% accuracy. Therefore, even though the overall accuracy dropped significantly, DkN is aware that most of these predictions are possibly incorrect. The accuracy of the model with the 90-100% credibility is 93%, thus the lowest credibility samples could be discarded and then the rest of predictions would have relatively high success rate.

Accuracy of DkNN

Interestingly, DkNN has a better overall performance on the adversarial examples. The method of comparing the intermediate computations shows much higher resilience from the adversarial attacks. In the Figure 4.5 accuracies of DkNN and baseline model with respect to epsilon are shown. For ϵ of 0.15, the DNN model has an accuracy of 13% which is only slightly better than assigning the label by random. The DkNN model for the same set of adversarial examples shows a sizeable improvement achieving an almost 50% success rate. An important value of accuracy is for the $\epsilon = 0.0$ because the new algorithm should not significantly compromise the performance of the non-adversarial data. The accuracy of baseline DkNN is 98.5% which is lower than standard DNN's 99.1%. Therefore, if no adversarial MNIST data is expected in the dataset, it is more effective to exclude the DkNN algorithm.

Out-of-distribution samples

Another drawback of Softmax predictions is its performance on out-of-distribution samples. These samples are images taken from a different dataset and do not represent any of the model's classes. Since the standard CNN returns a probability distribution over all of the classes, there is a substantial chance that the image looks similar to one of the labels and high accuracy will be assigned. To test this, the model has been evaluated on the Fashion MNIST

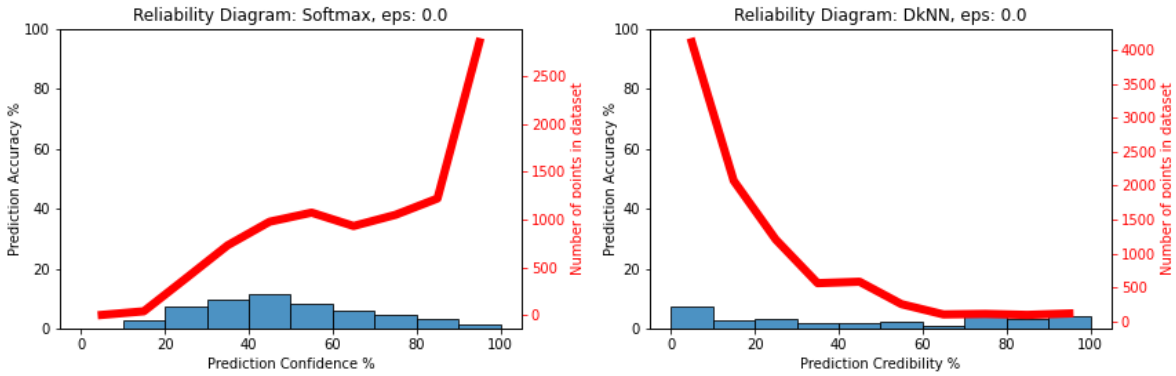

Figure 4.6: Reliability diagrams of CNN's Softmax confidence and DkNN's credibility for out-of-distribution dataset.

dataset. These images have an identical format as the handwritten digits but represent clothing articles. Therefore, there is no class overlaps and all of the predictions made will be incorrect. Figure 4.6 shows reliability diagrams for Softmax and DkNN tested on Fashion MNIST. As anticipated, standard CNN shows 90%-100% confidence for most of the samples, despite the that none of the images matches any of the classes. On the other hand, the DkNN model correctly assigns the lowest credibility to most of the data points. Only 5% of out-of-distribution images are assigned with credibility above 60%. Ideally, well-calibrated DkNN should assign 0% credibility to all out-of-distribution images. Nevertheless, this proves the intuition behind DkNN works in practice. It sets low credibility to examples that do not match its 'decision-making process' during training.

4.2 Experiments

In the original paper[21], experiments were only conducted using the same model and one approximate nearest neighbours algorithm. Therefore, these are the areas I will focus on examining and improving.

4.2.1 Alternative Approximate Nearest Neighbours Algorithms

The Multiprobe Locality Sensitive used in the original paper is an algorithm made for high dimensional data. However, there are many alternatives like vector encoding using trees or graph-based k-NN. These algorithms are used for finding similarities in images, music and many others[24]. However, not all of these techniques are suitable and adapted to such high dimensional data. Some of them would take hours to index the training examples and query the dataset. Therefore, I will cover two algorithms with which I had the best results.

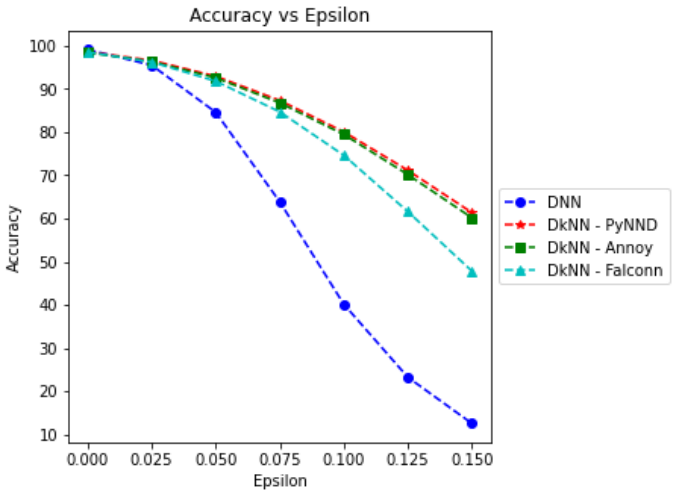

Figure 4.7: Accuracy of DkNN's with different approximate nearest neighbours algorithms DkNN and baseline model with respect to epsilon.

Annoy

Annoy is a python library from Spotify[30], which implements the graph-based vector encoding using trees (Explained in Section 2.8.1). However, it introduces multiple improvements for high-dimensional data. At each split of the tree, it examines both nodes and builds a priority queue. With this strategy, the algorithm is more likely to find the correct nearest neighbours. The second novel implementation is building multiple trees simultaneously, called a forest. Then the algorithm searches the forests in parallel also using a priority queue. Using the queues ensures that the nodes with the highest likelihood of being the closeset neighbour are searched first. Annoy is not the most optimal choice for indexing and querying. The creators of the library claim it works as efficiently as other well-known algorithms, although only up to thousand dimensions, whereas the DkNN model has 12,544 on the first layer. Its performance is slower than multiprobe LSH or graph-based k-NN. However, its strong advantage is the approach for storing indexes on the disc. Annoy's index takes only 3 GB of space and uses a fraction of the system's RAM during runtime in comparison to other algorithms. Furthermore, loading the file happens almost instantly. Incredible preprocessing performance is the result of decouping loading indexes from creating them. This also allows for using static files as indexes and distributing them between processes.

The results of using Annoy as the approximate nearest neighbours algorithm for DkNN are shown in Figure 4.7 (dark green function). Annoy noticeably improves the baseline algorithm performance. An important property is that the difference in accuracy increases with the epsilon. The improvement is most likely coming from the more accurate approximation of the nearest neighbours. This assumption is based on the observation of the correctness of neighbours for non-adversarial examples. Nearest neighbours found by Annoy library for the raw data were substantially more accurate.

PyNNDescent

PyNNDescent is a python library implementing Nearest Neighbour Descent for graph-based k-NN (Explained in Section 2.8.1). This library allows to store the neighbours' index, however, it takes several minutes to load and takes over 8.5 GB of memory on the disc. For a training dataset of a larger scale, the file size might cause a craash of system's random-access memory (RAM). The advantage of this library is the query time which is 200% faster than Falconn and 300% faster compared to Annoy.

The result of implementing PyNNDescent in DkNN is shown in Figure 4.7 (red function). The accuracy is comparable with Annoy, although slightly better and increases with the epsilon. Minor, however vital, improvement is for original data achieving 99.88%, which narrows the difference with the standard CNN model.

4.2.2 Alternative Deep Neural Network Architecture

State-of-art CNNs achieve an accuracy of 99.8% on the MNIST dataset[31]. Therefore, the baseline's model accuracy of 99.1%, does not vary significantly. However, the MNIST dataset is very primitive in comparison to nowadays images. In the original DkNN research[21], the model was tested on the SVHN dataset - real-world RGB images of the street view house numbers. The accuracy achieved by the research was only 90.6%, whereas state-of-art CNNs achieve over 96%[32]. In order for DkNN to have any real-life application it has to be adaptable to more complex CNN's architecture.

To enable comparison with the results from the previous experiments, the new model will be tested on the same dataset. The model chosen for experiments will be kaggle.com MNIST competition winner[33]. The CNN's architecture is shown in the Table 4.2. Following hyper-parameters have been set:

- Optimizer - Adam, learning rate of $\alpha = 10^{-3}$.
- Loss - Categorical crossentropy.
- Batch size = 500.
- Epochs = 8.

The main issue preventing the application of the DkNN algorithm to this model, is the number of neurons. The previous CNN already struggled with the high-dimensionality, and the new model is substantially more complex. This is especially significant due to the curse of dimensionality (Explained in Section 2.8). Therefore, the original DkNN Algorithm 1 will be extremely inefficient. The novel approach is to reduce the number of data points by performing an approximate nearest neighbour search only on a subset of layers.

Firstly, which layers to exclude from the search have to be identified. The first and second convolutional layers cannot be used, due to their extremely high dimensionality. The most efficient layer would be MaxPooling, which downsamples the previous layer, resulting in

Index	Layer	Filters	Kernel size	Activation	Output dim	No. of neurons
1	Conv	64	(3,3)	ReLU	(26, 26, 64)	43,264
2	Conv	64	(3,3)	ReLU	(24, 24, 64)	36,864
3	MaxPooling	-	(2,2)	-	(12, 12, 64)	9,216
4	BatchNorm	-	-	-	(12, 12, 64)	9,216
5	Conv	128	(3,3)	ReLU	(10, 10, 128)	12,800
6	Conv	128	(3,3)	ReLU	(8, 8, 128)	8,192
7	MaxPooling	-	(2,2)	-	(4, 4, 128)	2,048
8	BatchNorm	-	-	-	(4, 4, 128)	2,048
9	Conv	256	(3,3)	ReLU	(2, 2, 256)	1,024
10	MaxPooling	-	(2,2)	-	(1, 1, 256)	256
11	BatchNorm	-	-	-	(256)	256
12	Dense	-	-	ReLU	(512)	512
13	Dense	-	-	Softmax	(10)	10

Table 4.2: DNN architecture of kaggle MNIST winner[33].

fewer neurons to index. The combination of the pooling layer after the convolutions allows the model to learn more global features of the digits by merging local features. Therefore, if an adversarial example was able to confuse the CNN on one of these convolution-maxpool stacks, the DkNN algorithm should detect it. The batch normalization layer has the same amount of neurons, however, it performs alternation of mean and scaling of the activations to prevent internal covariate shift (explained in Section 2.4.3). This is a useful property for training the CNN and will also improve the DkNN performance. It will ensure that activations ranges are very similar thus their distribution between batches should be preserved, resulting in more accurate neighbours search. Therefore, the approximate nearest neighbours algorithm will be applied to all Batch Normalization layers. The dimensionality of these layers is smaller than the neurons of the baseline model, therefore, despite complex architecture, the indexing and querying of the neighbours will be more efficient.

The accuracy of the new model is shown in Figure 4.8. It achieves a 99.5% success rate on the raw MNIST dataset, which is an improvement from 99.1% for the baseline model. Interestingly, the model shows much higher resilience to adversarial attacks. Despite a similar accuracy for $\epsilon = 0.0$ of 0.4 percentage points between models, the difference grows to 30 for $\epsilon = 0.15$. This is most likely due to the improved calibration of the model. The decision boundaries are further apart which prevents misclassification for small perturbations.

Following the same procedure as the baseline model, reliability diagrams in Figure 4.9 were created. The new model's Softmax performance is similar to the baseline model. Once again, almost all predictions are given between 90% and 100% for both raw MNIST data and adversarial examples. A minor difference is the overall higher accuracy of those predictions, however, for $\epsilon = 0.15$ and the highest confidence, less than 40% of datapoints were classified correctly. The reliability for DkNN for raw images is almost identical to the baseline model, although only from credibility below 30% the accuracy decreases below 99%. Alike Softmax,

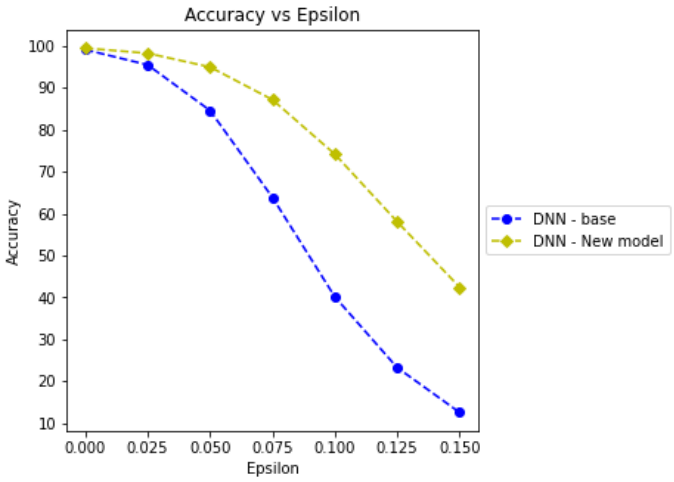

Figure 4.8: Accuracy of the new and baseline model with respect to epsilon.

most images are classified in the highest reliability category. A significant improvement, in comparison to the baseline DkNN, is visible for the adversarial examples. Apart from the overall higher accuracy, the model assigns 90-100% credibility to a substantial number of perturbated images. Most importantly, these predictions have an accuracy of 100%. In the baseline model, in order to achieve accuracy above 90%, almost all predictions would have been discarded, because a very limited subset was given credibility over 30%. Whereas, the new model would be able to classify over half of the test set with at least 90% accuracy.

In Figure 4.10 a comparison between both models have been presented and all settings. Two graphs with the highest accuracy represent the new model performance using PyNNDescent approximate nearest neighbours. The black plot represents a novel approach applied on the MaxPooling layer, whereas purle on the Batch Normalization layer. The two curves are almost overlapping, but there is a slight improvement with application on the latter. The mean accuracy for $\epsilon = 0.15$ is 87% which is a substantial increase from the best performing baseline algorithm with a success rate of 61%.

4.2.3 Evaluation of different dataset

To support the findings, all evaluations performed on the MNIST dataset have been also tested on the Fashion MNIST. This dataset of clothing articles is more challenging than handwritten digits, which results in a worse performance of both CNN and DkNN. However, the overall observations still hold. The softmax function assigns high accuracy regardless of provided samples. Whereas DkNN credibility shows a linear relationship between credibility and accuracy. The diagrams for the evaluation of Fashion MNIST are presented in Appendix A.

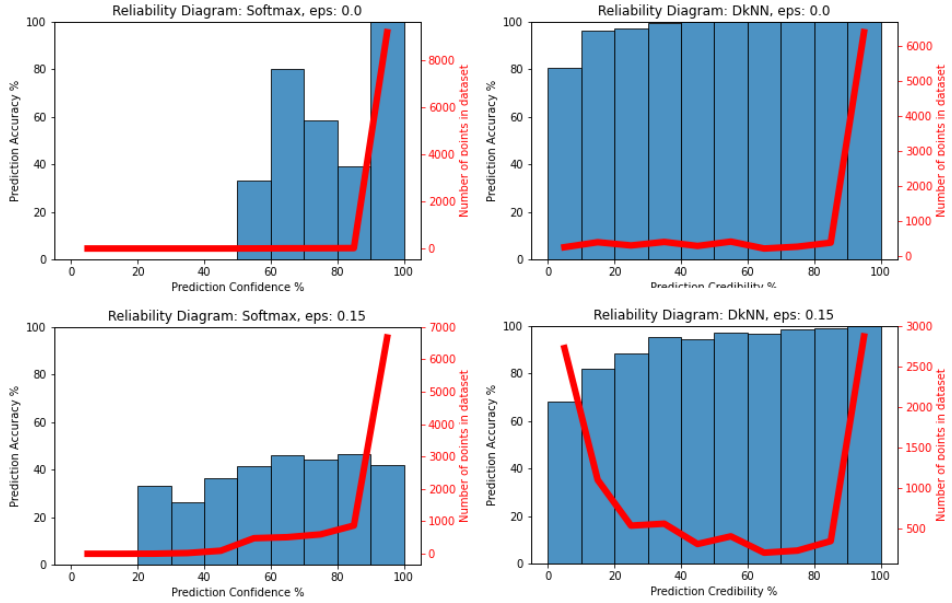

Figure 4.9: Reliability diagrams of CNN's Softmax confidence and DkNN's credibility.

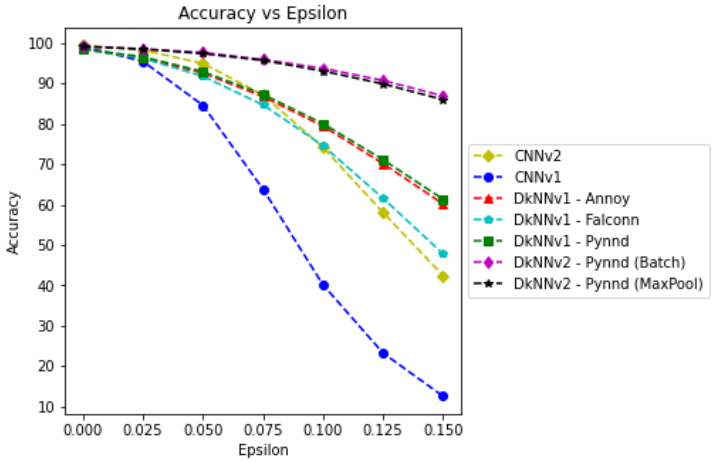

Figure 4.10: Accuracies of both models and all settings with respect to epsilon.

Chapter 5

Discussion

5.1 Advantages of the Deep k-Nearest Neighbours

Using the Deep k-Nearest Neighbours as a credibility measurement for MNIST data seems very reliable. Even for intense perturbation, by discarding the lowest credibility estimates, the model has reliable accuracy. Successful application of the DkNN to a state-of-art MNIST model proves that it is possible to implement for more complex architectures.

Another vital advantage of using the DkNN is the ability to rationalize the prediction made by the model. Standard CNNs behave like black boxes, it produces an output, although it is hard to interpret what led to that decision. Using the DkNN, the nearest neighbours found on each layer can be examined. Then a prediction can be rationalised by inspecting training examples on which the CNN based its output. For instance, a doctor might be able to interpret a disease prediction made by a CNN on an MRI image, based on the closest matching scans that the DkNN found within its training examples.

5.2 Limitations

The project has been limited to examining the performance of the Deep k-Nearest Neighbours only on one type of Adversarial Attack. In theory, the attacker would have to recreate the 'decision-making process' in order to successfully attack the model. Therefore, the big advantage of DkNN is its universality to many adversarial attacks. However, specific attacks to fool the DkNN classifier has been discovered, achieving an incredible success rate [34][35]. Another limitation has been the dataset. The MNIST handwritten digits and Fashion MNIST are primitive datasets consisting of only 784 pixels, which is a tiny fraction of a photo taken by a camera. For a standard size photo, the performance of the algorithm might change drastically. However, only several approximate nearest neighbours algorithms have been tested in this project and most of them were not adapted for such high dimensional data. Perhaps there exists (or could be developed) an algorithm suitable for approximating the nearest neighbours efficiently for data with even more features. This would allow for the implementation of the DkNN in more demanding models with more complex architecture.

Chapter 6

Conclusion

In this project, an in-depth examination of Deep k-Nearest Neighbours has been conducted. DkNN is an algorithm which assigns reliable credibility to its prediction. The Neural Network model from the original research was found inefficient for more complex datasets. Therefore, a new model has been proposed and evaluated with high performance, using state-of-art architecture and the newest approximate nearest neighbours. The implementation of the new model was feasible, due to alterations made to the core algorithm, which extracts neighbours only on a subset of layers. With its high performance, it is a reliable model for FGSM adversarial attacks and out-of-distribution inputs. Despite a low average accuracy on heavily perturbated FGSM images, it assigns low credibility to the incorrect predictions. Therefore, the incorrectly labelled data can be discarded and the model performance will remain stable. Furthermore, the algorithm can be used to rationalize the prediction made by the CNN. The similar training examples give an insight into what data points the model was relying on.

Appendix A

Fashion MNIST Results

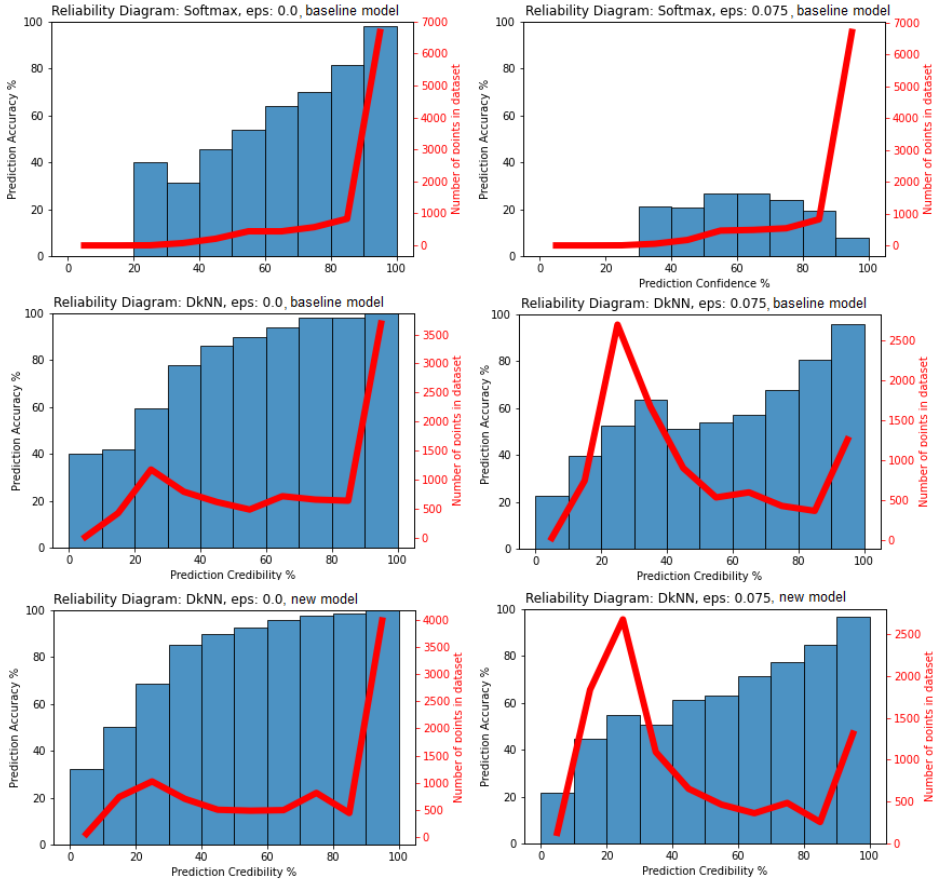

Figure A.1: Reliability diagrams of CNN's Softmax confidence and DkNN's credibility on Fashion Mnist.

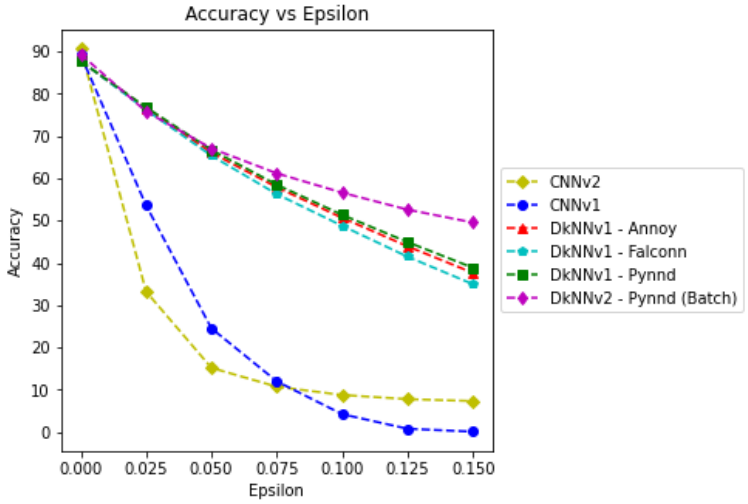

Figure A.2: Accuracies of both models and all settings with respect to epsilon.

Bibliography

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[4] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity.," 1943.

[5] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," 1958.

[6] M. A. Nielsen, "Neural networks and deep learning," *Determination Press*, 2015.

[7] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018.

[8] R. Dechter, "Learning while searching in constraint-satisfaction-problems.," 1986.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[10] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016.

[11] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," 1962.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[13] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[14] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," 2020.

[15] P. Harder, F.-J. Pfreundt, M. Keuper, and J. Keuper, "Spectraldefense: Detecting adversarial attacks on cnns in the fourier domain," 2021.

[16] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent advances in adversarial training for adversarial robustness," 2021.

[17] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2016.

[18] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," 2017.

[19] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," 2015.

[20] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2016.

[21] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," 2018.

[22] Y. Gal, "Uncertainty in deep learning," 2016.

[23] V. Pestov, "Is the k-nn classifier in high dimensions affected by the curse of dimensionality?," 2011.

[24] Y. Jing and S. Baluja, "Visualrank: Applying pagerank to large-scale image search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1877–1890, 2008.

[25] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," 2016.

[26] J. Sharma and S. Navlakha, "Improving similarity search with high-dimensional locality-sensitive hashing," 2018.

[27] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[28] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," 2015.

[29] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[30] "Annoy library." https://github.com/spotify/annoy.

[31] S. An, M. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for mnist digit recognition," 2020.

[32] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," 2013.

[33] I. Kassem, "Mnist: Simple cnn keras," 2022.

[34] C. Sitawarin and D. Wagner, "On the robustness of deep k-nearest neighbors," 2019.

[35] X. Li, Y. Chen, Y. He, and H. Xue, "Advknn: Adversarial attacks on k-nearest neighbor classifiers with approximate gradients," 2019.