Robustness of a neural network used for image classification

The effect of applying distortions on adversarial examples

Rasmus Östberg

2017
Abstract

Powerful classifiers as neural networks have long been used to recognise images; these images might depict objects like animals, people or plain text. Distorted images affect the neural network’s ability to recognise them, they might be distorted or changed due to distortions related to the camera. Camera related distortions, and how they affect the accuracy, have previously been explored. Recently, it has been proven that images can be intentionally made harder to recognise, an effect that last even after they have been photographed. Such images are known as adversarial examples. The purpose of this thesis is to evaluate how well a neural network can recognise adversarial examples which are also distorted. To evaluate the network, the adversarial examples are distorted in different ways and thereafter fed to the neural network. Different kinds of distortions (rotation, blur, contrast and skew) were used to distort the examples. For each type and strength of distortion the network’s ability to classify was measured. Here, it is shown that all distortions influenced the neural network’s ability to recognise images. It is concluded that the type and strength of a distortion are important factors when classifying distorted adversarial examples, but also that some distortions, rotation and skew, are able to keep their characteristic influence on the accuracy, even if they are influenced by other distortions.

Keywords: LeNet, Distorted Images, MNIST, Adversarial Examples
Contents

Abstract ................................................................................................................................. 2

1 Introduction ..................................................................................................................... 2
  1.1 Research Task ............................................................................................................. 2

2 Background .................................................................................................................... 2
  2.1 Image processing fundamentals ............................................................................... 2
  2.2 Image recognition and classification ......................................................................... 3
  2.3 Image classification and recognition with neural networks .......................................... 4
  2.4 The Artificial Neural Network .................................................................................... 4
  2.5 Convolutional Neural Networks .................................................................................. 5
    2.5.1 Components of the Convolutional Neural Network ............................................... 6
    2.5.2 Estimating the predictive ability of the CNN ......................................................... 6
  2.6 Robustness of neural networks for image classification ................................................ 7
    2.6.1 Adversarial examples ........................................................................................... 7
    2.6.2 The effect of image quality on classification accuracy ............................................ 8
    2.6.3 Transform-Invariant Convolutional Neural Networks for Image Classification and Search ................................................... 8
    2.6.4 Robustness methods for neural networks ............................................................... 9

3 Method and Experiment setup ...................................................................................... 10
  3.1 Building and Evaluating the LeNet model ................................................................. 10
  3.2 Applying and estimating the adversarial noise .......................................................... 11
  3.3 Choosing the distortions and varying their levels ....................................................... 12
    3.3.1 Evaluating Single distortions ............................................................................... 12
    3.3.2 Evaluating the Combination of distortions ........................................................... 13

4 Result ............................................................................................................................... 14
  4.1 Single distortion - Our findings ................................................................................... 14
  4.2 Combinations of Distortions - Our findings ............................................................... 16

5 Discussion ....................................................................................................................... 18
  5.1 The model and the dataset ......................................................................................... 18
  5.2 Affine transformations and image clipping .................................................................. 19
  5.3 The distortions .......................................................................................................... 19
  5.4 Possible explanation for differences between results ................................................ 19
  5.5 Further Research ....................................................................................................... 20

6 Conclusion ...................................................................................................................... 20

A Training the CNN ........................................................................................................... 24

B Software and Hardware .................................................................................................. 25
List of Figures

1. The accuracy given a combination of distortions. .................................................. 3
2. An example of a digit from the MNIST database.......................................................... 4
3. A basic ANN which consists of one hidden layer......................................................... 5
4. Explanation of backpropagation. .................................................................................. 5
5. Description of the components in the LeNet model. ...................................................... 7
6. Explanation why adversarial examples are possible....................................................... 8
7. Comparison how others have generated adversarial examples...................................... 12
8. Visual representation of all rotations. ........................................................................... 13
9. The order which the distortions were applied in. ......................................................... 13
10. Accuracy when applying Gaussian Blur or Contrast. ................................................... 15
11. Accuracy when applying Rotation or Skew ................................................................. 15
12. The accuracy given the combination of contrast, rotation and blur ............................. 17
13. The accuracy given the combination of contrast, rotation and skew ............................. 18

List of Tables

1. The architecture of the LeNet model................................................................. 10
2. Hyper-parameters of the LeNet model. ................................................................. 11
3. Distortion levels that were used when combining different distortions. ............ 13
4. Adversarial levels that were used when combining different distortions. ............. 13
5. The minimum and maximum levels for all distortions and the adversarial noise. ... 14
6. The most notable differences, when classifying distorted adversarial examples .... 14
7. The combination of distortions that lead the the lowest accuracy ......................... 18
8. The combination of distortions that lead the the highest accuracy ......................... 19
9. Information about used Software and Hardware ...................................................... 25
1 Introduction

Computer vision is used in many fields and in some cases, it is used to recognise objects or perhaps people. It can also be used to guide a vehicle through a terrain or in combination with facial recognition to enable a user to login into a laptop. When recognising images the computer is simply fed images which it is trying to identify. In most cases the computer can be very correct regarding image recognition, thanks to the use of neural networks. But, in recent studies [1, 2, 3] it has been proven that it is possible to ‘disguise’ the input image. Not only is the image harder to recognise, it is also recognised as another image, and this is done with high confidence. When the image is disguised as another image it is known as an adversarial example, Section 2.6.1. This kind of disguise is preserved even if the image has been printed and later photographed.

Certain distortions or defects can arise when photographing, the photograph might appear a bit different compared to the real life object. Perhaps, the photograph becomes slightly rotated or the camera might not be properly focused which results in a blurry photograph. It has already been showed that these kinds of distortions affect the recognition ability of a computer [4, 5]. Since the distortions affect the computer’s ability to recognise images, one may wonder how these distortions affect adversarial examples, and what influence they have. An adversarial example might be harder or easier to recognise, depending on what type of distortion is applied.

This thesis investigates how adversarial examples are affected by distortions that can arise when photographing. Here, the adversarial examples are constructed from the well known ‘MNIST’ database of digits, Section 2.1. The adversarial examples are therefore digits which are disguised as other digits. In this thesis, a neural network based on the so-called LeNet model, is used to classify (distorted versions of) these adversarial examples. Each distortion may affect neural network’s ability to classify, it is also possible that multiple distortions may interact and further affect the classification rate.

1.1 Research Task

In this thesis, it is evaluated how well a neural network, based on LeNet model, is able to recognise distorted adversarial examples, the model’s ability to recognise is measured with classification accuracy. The combination of distortions have to be taken into account since the interaction between distortions affect the accuracy [6]. The research task is intentionally limited in two ways regarding the distortions. First, only a few distortions are picked and used to distort the adversarial examples. Second, the distortions are applied in an arbitrary order, the order might be an important factor when distorting images, but this is not investigated here. The order of distortions is not changed throughout the whole evaluation.

The research task is divided into the following objectives.

1. How do single distortions affect adversarial examples?
2. How does the combination of multiple distortions affect adversarial examples?

2 Background

2.1 Image processing fundamentals

Image processing is a procedure used to analyse or manipulate images [7]. When processing the image we could extract information or transform the image to something else. A certain sort of image processing is image classification. This processing technique takes an image and analyses it. Depending on what we extract from the image we could make different decisions.
When transforming the image we aim to change the original image to something better. Of course, ‘better’ differs from case to case. When removing noise, e.g. with low pass filtering, a better image would be defined as less noisy version. With a low-pass filter, we aim to remove noise and make the image smoother, a common technique in image processing [7]. An image which is slightly tilted or not centred can be corrected with affine transformations. Such transformations moves or distorts the image, for example; rotation, skew or scale.

When applying different transformations we change the image into something similar. Despite the similarity between the original and the new one we might loose important information. Often, when applying affine transformation parts of the image is moved outside of the image and will be implicitly cropped [6]. Since the image is pictured/constrained in a grid of pixels, all parts that are outside the grid are also lost. Features which are placed in corners might get lost when performing rotation or skew, since the corners are moved outside the box, see the example in Figure 1. Here it is clearly seen that a larger part of the image is moved outside the original pixel grid/bounding box (red marking). The image below depicting a ‘7’ is moved outside the box, and thus resembles a ‘1’. Without the horizontal line, it is harder to distinguish the difference between 1 and 7.

![Figure 1: Affine transformations such as rotation, skew and translate, moves the digit outside the grid, thus removing the edges and perhaps important features used to process the image. Here, the leftmost image picturing a ‘7’ from the MNIST dataset is transformed by rotation, skew and translate distortions, and thereby loses important features.](image)

**The MNIST data set** The MNIST (Modified National Institute of Standards and Technology database) dataset [8] was used when Uličný [9] evaluated how well adversarial noise affected neural networks. The dataset consists of a large number of handwritten digits, ranging from 0 to 9. The whole set contains of 70,000 pairs of images and their numerical representations. The dataset is divided into two parts, a training subset and a testing subset. The two sets consist of 60,000 and 10,000 digits and numbers respectively. The training set and testing set is written by different writers, and their characteristics are therefore different. Each handwritten digit has been translated to a 28x28 grid of cells, Figure 2, where each cell describes a pixel. This presentation was created by scanning each handwritten digit and then translating each digit to a resized image. Each pixel contains a value ranging from 0 to 255. (The digits were scanned and saved without the colour scale, hence the singular value in each cell. An RGB(A) image should therefore contain a tuple of three values one for each colour channel).

### 2.2 Image recognition and classification

Images are complex objects, they are not just some random pixels a grid, they describe things [10]. In the case of handwritten characters the images describe letters or digits. We humans, are able to distinguish between one letter and another, even if they have different styles. When we were younger, we were not so sure about what features each letter have. However, we slowly learned, example by example what the character ‘B’ looks like. By closely studying each letter we find out that certain letters have certain features, and that the combination of certain features is unique for each letter. For example, the letter ‘Q’ is a ring-shaped figure with a little ‘tail’. This
means, that with the help of these feature we are able to detect letters. And this is all what image classification is about. A classifying model, for example a neural network may divide the image into smaller parts to figure out what it depicts.

2.3 Image classification and recognition with neural networks

One solution to classify images is to use Deep Learning, an approach that allows computers to learn by experience to understand different hierarchies of concepts [11]. These concepts are often connected to each other and can be described by a graph. This graph is often filled with many 'layers' of concepts, thus creating a 'deep' graph. This is the reason why it is called Deep Learning. Deep Learning is a Machine Learning Algorithm that is able to recognise patterns. A typical example of Deep Learning is the Multilayered Perceptron, a very basic version of a neural network. To recognise images, the pattern recogniser has to be trained on a large set of training data. In the case of image classification we feed a large set of image examples to the classifier. By analysing the pixel values of multiple images, a neural network is able to learn what the images represent. But, it is difficult to understand the raw pixel values in an image. A neural network solves this by breaking the image into smaller images to simplify the problem. By analysing the smaller images it is able to extract features used to classify the image itself.

To classify images the neural network needs to be trained with examples from a training set and validated with a test set. This type of training, in which the user 'trains' the network on different examples, is known as supervised learning. In this thesis a neural network is trained and used to classify different types of handwritten digits from the MNIST dataset (see Section 2.1).

2.4 The Artificial Neural Network

The artificial neural network (ANN) is an input classifier, which is inspired by how the central neural system is constructed [12]. The network learns a generic pattern with the help of training examples and can thereafter map one dataset to another. As the name suggests the network is built by smaller units named neurons. All neurons are connected in a layer-based structure, with one input layer and one output layer. All intermediate layers are hidden layers which are used to construct the intermediate output in the model. For a neuron to pass signals to other neurons they are connected with so called weights [10], Figure 3. A neuron in a neural network is equipped with an activation potential which decides if a neuron should fire a signal or not depending on a specific input signal. When a neuron receives an input, be it from another neuron or the input layer, it processes the signal and produces an output signal. The output signal is then sent to the next layer. By repeating this process the signal propagates through the whole network. When the signal reaches the last output layer it is used as the network’s final output. The output shows what class the input belongs to, but this is only possible if the network can describe the relationship between input and output.

For the network to be able to describe the input/output relationship it has to be trained. The training is focused on minimising the difference between the expected output and the actual
one [10]. By doing so, the chance of computing correct classification is increased. To estimate this difference a loss function is used when the machine is trained, the loss function guides us how to change neuron-weights to minimise the loss. Backpropagation is a technique to update the weights by sending the estimated error signal in the opposite direction, Figure 4. When the signal propagates backwards through the network the weights are updated by a weight optimiser, for example the Stochastic Gradient Descent (SGD). In short, back propagation consists of two phases. In the first phase the network takes an input and estimates its class $y$. The error/loss is estimated based on the difference between $y$ and the actual class $\hat{y}$. In the second phase the weights are updated to minimise the loss. The error signal produced in the previous phase propagates backwards in the net and the weights are updated to minimise the loss.

![Figure 3: A basic ANN which consists of one hidden layer.](image3)

![Figure 4: Both phases of the backpropagation is visualised here.](image4)

### 2.5 Convolutional Neural Networks

When using a normal neural network to classify images some requirements have to be met, stated by LeCunn [13]. The objects in each image have to be centred and normalised before fed to the network. On top of that the training set must be quite large to train the network with all variants of the same input. This is because the network lacks the ability to generalise images with respect to local invariants or smaller distortions, e.g slant or translation. Generalisation depends on finding the correct weight values for the network. The neural network classifies images depending on the structure of each pattern. Since the input varies from image to image, for example the digit ‘1’ can be drawn in different styles, the network has to be trained with an enormous size of data. In other words, a larger set of training data has to be used to find the correct weight values due to the neural network’s ability the generalise images.

To be able to distinguish between patterns in images we have to extract features. This is often hard when the images are different from each other and certain features are only found in a few variants of a specific type of image. For example; features like curves and loops are found when classifying the digits like ‘8’ or ‘6’ but not when classifying a ‘1’. More complex feature such as arcs or zigzag lines, are built by lesser/smaller features like dots or lines. For example, if
the image contains two endpoints and one line which is connected between those points, there
is a high probability that the input is the number one (1).

A neural network that is commonly used for image recognition is the CNN (Convolutional
Neural Network), since it is able to automatically extract features from the input space. In the
CNN model neurons of the visual system are connected to sub fields in the receptive field on
the input, which makes it possible to extract local features without having to specify what to
extract. When a CNN model is constructed, different layers, or filters, are connected to different
parts of the input image. Each layer is responsible to extract local features. The features are
then combined to form higher order features which are used to classify the image.

2.5.1 Components of the Convolutional Neural Network

The CNN model could be described as the combination of a feature extractor and a classifier.
In the CNN model, there are different types of layers which are the building blocks of the neural
network itself. Some are used for feature extraction and others are used for the classification. In
this section all the important layers of the CNN model are presented and summarised in Figure 5.

The layer that is used to extract features is the Convolutional layer. Each unit in this layer is
connected to a small portion of the previous layer, thus receiving input from the previous layer.
(Except for the first CNN layer which receives data from the image itself in terms of pixels and
their colour values.) This makes it possible for the model to extract features without having
to tell it what to extract. In each convolutional layer, the neurons are organised as flat planes
where all neurons in the same layer share the same neuron-weights. The neurons from one plane
perform the same feature extraction all over the image. Multiple planes are used to extract
multiple features since each plane is bound to search for a specific feature. As stated earlier,
distortions and shifts of the input can cause the position of important features to vary. In a
convolutional layer, the extracted features are also shifted or distorted in the same manner as
the input which makes the neural network robust against somewhat distorted images. But the
robustness to small variants is also possible thanks to the Pooling Layers (also known as sub-
sampling layers). To make the network able to generalise distorted input data, the positions
of the features are averaged/recalculated with a pooling layer [13]. Pooling layers are constructed
to reduce the noise in the feature space. But also, to increase the size of the next layer’s receptive
field without changing the size of the filter [5]. Each pooling layer takes a subarea from the input
from the previous layer and sub-samples it to a smaller size. The input can be sub-sampled by a
so called average pooling layer which computes the average of the input or a max pooling layer
that extracts the maximum value in the sub-area.

The classification part of the CNN is actually a normal fully-connected network which is
explained in Section 2.4. The features which are extracted by the CNN layer is fed to this neural
network for classification.

2.5.2 Estimating the predictive ability of the CNN

When training and testing a predictive model, such as the LeNet, it is crucial to estimate its
performance. Different methods are used to estimate the performance, e.g. Cross Entropy Loss
(CEL) and top-k. These measurements are based on how many correct classifications the model
can do.

The purpose of the CEL estimator is to minimise the loss when the model is still in training.
After the model is trained another estimator is used, namely the top - k estimator which works
in the following way: When the CNN classifies an input \( x \) (be it an image or a digit) it computes
the probability that the input belongs to each of the \( n \) classes. If the input’s actual class \( \hat{y} \) is
found in the \( k \)-most probable classes the classification is seen as a correct one, otherwise not.
When classifying one single input the estimator gives one of two values: 1 (correct classification)
Figure 5: In the above Figure (from Lecun [13]) all layers are used as building blocks to construct a CNN model. Here are all layers presented; convolutional, sub-sampling and the ‘normal’ or basic dense layer. A number of convolution and sub-sampling layers are used to extract features which are fed to a ANN.

or 0 (wrong classification). The average of all correct classifications of a whole dataset \(^1\) is referred to as the model’s classification accuracy which is proportional to the correctly classified classes, or digits in this case.

2.6 Robustness of neural networks for image classification

2.6.1 Adversarial examples

The adversarial examples are input data that are often miss-classified and done so with high confidence [1], even if the trained model already have a high accuracy.

Training data, explained in Section 2.3, is used to train a model to be able to understand a real-world problem, for example, to identify faces or objects depicted in images. The amount of training data generally increases the model’s ability to understand the problem. Often, we do not have enough training data to be able to construct a dataset that describes the complete feature space. Instead, we use data that cover a larger portion of the feature space, this is referred to as incomplete training data [1]. By combining the training data with the model’s ability to generalise we end up with a powerful model. The incomplete data builds the decision boundary for the model, while the complete training data builds the real decision boundary. Even if we were to collect more data we are not able to cover all input and output combinations which builds the complete training set.

The difference between the model’s decision boundary and the real boundary creates pockets, the pockets are also known as adversarial regions. The reason that adversarial examples are hard to classify is that they are moved into the adversarial regions, due to the manipulation of the input data. An input that belongs to a certain class could be moved closer to a region which forces the model to classify it as something else. To give a better understanding of how these pockets look like, see Figure 6, in this example the task is to detect spam emails. An e-mail could belong to either of two classes, spam or not spam. Suppose a trained model could distinguish between these two classes, which are separated by the model’s decision boundary (red linear line). The real decision boundary (blue) differs from the red, creating adversarial regions. The classifying model is confident when miss-classifying input data placed in any of these adversarial regions.

Adversarial examples are input data which are harder to classify, since the features from the input are moved to an adversarial region. To move the features, a special kind of noise called adversarial noise is applied and is generated by Fast Gradient Sign Method (FGSM) [1], see Equation 1. The FGSM method works in the following way. The adversarial noise is generated

\(^1\)This data set could be either the test set or the validation set.
Figure 6: In this Figure (from Papernot et.al [1]), two classes are plotted in a scatterplot. The predictive model builds the (red) decision boundary (from the incomplete training data). But the (blue) real boundary is built by the complete training data. The difference between them creates the adversarial regions, the source of adversarial examples.

by computing the gradient $\nabla_x$ (w.r.t $x$) of the loss $L$ \footnote{Read as: The loss $L$ which depends on model $\phi$, input $x$ and output $y$.}, which is estimated by the model when presented with an input $x$ and the corresponding output $y$. In other words $L$ depends on the neural network. The gradient is used to compute how the input should be changed, in order to push it into one of the adversarial regions. The direction of the change, i.e if a certain pixel value should be increased or decreased, is computed by the ‘sign’ function. The strength of the adversarial noise is adjusted by a factor of $T$. The factor is a positive float that increases the intensity of the adversarial noise, giving us the adversarial noise $R$. The final noise, $R$, is then applied to the original input $x$, thus we acquire an adversarial example.

$$R = T \times \text{sign}(\nabla_x(L(\phi; x; y)))$$

(1)

2.6.2 The effect of image quality on classification accuracy

Dodge and Karam [5] evaluated how distorted images affects the classification accuracy of neural networks. Another goal was to evaluate when a specific level of distortion decreased the accuracy of the predictive model. The selected models that were evaluated are well known networks as VGG-CNN, VGG-16 and GoogLeNet. They state that adversarial examples are not likely to happen by pure coincidence and input data is more likely to be affected by blur, contrast and JPG compression. To evaluate the selected networks they were being fed distorted samples from the ImageNet (2012) dataset. Different sub samples of the dataset were distorted by the noise, creating data with quality ranging from excellent to poor. The whole distorted dataset was thereafter fed to the network. To measure the classification accuracy when classifying the dataset they used top-1 and top-5 accuracy. In their result, they show that all networks were robust against compression and contrast based noise, meanwhile blurred images lead to worse performance, even when adding just a small amount of blur.

2.6.3 Transform-Invariant Convolutional Neural Networks for Image Classification and Search

The CNN model is protected against small affine variations (rotate, scale and translation) of the sub-features because of the shared weights and pooling layers, but when global features are distorted by said variations the CNN model is not that resistant. Xu Shen el al. [6] explains
that this is caused by the architecture of the neural network. In their paper, they develop a method to train a CNN to be able to classify distorted input images. Such a network was given the name TICNN (Transform-invariant CNN), a network which is robust against stronger distortions. The method focuses on creating a new layer which can be applied to any of the layers of the CNN. The new layer performs random transformations when the model is trained and the transformations are also applied to the feature maps between the layers. This makes it possible to train the network on the same dataset without changing it or extracting extra input-features. The transformations of the feature maps are transformed in the following order: scale, rotation and then translation. In their experiment they used the MNIST dataset. The dataset was distorted by random rotation in the interval $[−90, 90]$ degrees, scaled with a factor $[0.7, 1.2]$ and the images (with size 28x28) were translated in a 42x42 grid, thus creating a distorted version of the MNIST.

2.6.4 Robustness methods for neural networks

The thesis by Uličný [9] is one of the building blocks for this report. In this section we highlight some parts of his thesis. Uličný evaluated how robustness methods protected neural networks against adversarial examples, which were created from the well known MNIST dataset [8]. Some of the robustness methods were used to manipulate the adversarial examples and thereby make them easier to classify. Uličný used other methods to enhance his neural network’s resistance against adversarial examples, and different combinations of noise-robustness methods were used in the evaluation. The classification accuracy was measured with the top-1 and top-5 accuracy.

Applying and estimating adversarial noise Adversarial examples are created by applying adversarial noise, explained in Section 2.6.1, to normal input data. The strength of the adversarial noise is varied by the parameter $T$. Uličný varied this parameter in the interval of $[0.07, 0.35]$, thus creating different noise levels of the adversarial noise. Another important parameter is $L$, which is the loss function supplied by the neural network. By estimating the injected noise it was possible to compare two different types of noises, adversarial noise and gaussian noise. Even if the two noises managed to reach the same ‘injection level’, the adversarial noise had clearly more influence in the classification accuracy. The measurement of injected noise is done by $dist$ which is the average pixel differences for two datasets, the normal MNIST dataset $X$ and its adversarial counterpart $\tilde{X}$. The average difference was measured per pixel, for each colour band $c$, between all $n$ pairs of images; with height $h$ and width $w$, according to Equation 2.

$$dist = \frac{1}{n \cdot c \cdot h \cdot w} \sum_{k=1}^{n} \sum_{i=1}^{c} \sum_{j=1}^{h} \sum_{l=1}^{w} |X_{klij} - \tilde{X}_{klij}|$$ (2)

Robustness Methods Here we highlight some of the robustness methods used by Uličný for protecting the neural network against adversarial examples. In this section we describe how some methods make the model more resistant but also how other methods are meant to reduce the adversarial noise in the input data. Some of these methods requires the model to be retrained, other methods do not.

When the adversarial examples had been created by using FGSM method, Section 2.6.1, different robustness methods were used to evaluate how well they could protect a neural network from adversarial examples. Some of the robustness methods focused on preprocessing the images to be classified, for example using lowpass filtering, see Section 2.1, and Denoising Autoencoder, presented below. Other methods focused on strengthening the neural network itself, by making it able to classify the adversarial examples (adversarial training) or by ‘dropping’ parts of the neural network (dropout). The neural network was combined with other networks to form a so-called ensemble, which makes it possible for similar models to work together to classify the
input data. Low-pass filtering is a common noise reduction method. Here, blurring and denoising functions were used to remove adversarial noise from adversarial examples. The idea is to use a convolutional kernel to ‘roll over’ or convolve the input to produce white gaussian noise, by using blurring or denoising methods. The purpose of this noise is to push adversarial examples out from the adversarial regions. The notion of these regions is explained in Section 2.6.1. A Denoising Autoencoder, DE for short, is a special form of a neural network that can be used to reduce noise in the input space. To remove noise in the input the network has to be able to map an input to another version of said input. The DE is then able to translate noisy images to non-noisy images. The mapping is accomplished by feeding the DE normal images and noisy versions of them. Uličný [9] used a DE to reduce adversarial noise from the adversarial examples. To increase the resistance against adversarial examples the examples can be included in the training phase of the model. This is called adversarial training. When dropout [14] is applied to the model a fraction of the inputs are ‘dropped’ and ignored. Dropout is used to prevent over-fitting, but also to make the neural network better at generalisation. Uličný constructed different dropout models and used them to increase the robustness of the neural network.

**Architecture of the neural network** The neural network that was used by Uličný [9] was the LeNet, this network is of the convolutional type and it is often used when classifying images, see Section 2.3. A convolutional network (CNN) is constructed by two networks, see Section 2.5. The first network is built of a number of the convolutional layers which are responsible for extracting features from the input, these features are sent to the second sub-network which is a normal fully connected network. The fully connected network is responsible for classifying the extracted features. In total, the LeNet architecture is a CNN composed of four layers: two convolutional and two fully-connected.

The activation functions [12, 15], that were used are ReLU and the Softmax function. Where the latter one is used for the final output and the former is used for the intermediate layers. The LeNet network is described in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Kernel/Stride</th>
<th>Outputs</th>
<th>Layer Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>5x5/1</td>
<td>24x24x20</td>
<td>1</td>
</tr>
<tr>
<td>Max Pool</td>
<td>2x2/2</td>
<td>12x12x20</td>
<td>-</td>
</tr>
<tr>
<td>Conv</td>
<td>5x5/1</td>
<td>8x8x50</td>
<td>1</td>
</tr>
<tr>
<td>Max Pool</td>
<td>2x2/2</td>
<td>4x4x50</td>
<td>-</td>
</tr>
<tr>
<td>Dense</td>
<td></td>
<td>1x1x500</td>
<td>1</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
<td>1x1x500</td>
<td>-</td>
</tr>
<tr>
<td>Dense</td>
<td></td>
<td>1x1x10</td>
<td>1</td>
</tr>
<tr>
<td>Softmax</td>
<td></td>
<td>1x1x10</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: The basic architecture of a LeNet. The network consists of two pairs of convolution and pooling layers for feature extraction, and a final dense network for feature classification.

**3 Method and Experiment setup**

**3.1 Building and Evaluating the LeNet model**

The neural network was built by using Keras [16], a framework for neural networks written for Python. This framework was chosen because it serves as a more user-friendly and easy-to-use abstraction layer on top of TensorFlow [17] and Theano [18], two Python-based frameworks for creating and building neural networks. Here we used TensorFlow as a backend when classifying digits from the MNIST dataset. Thanks to the GPU version of TensorFlow we saved computation...
time. The GPU was used to train and estimate the performance of the LeNet model when fed adversarial examples.

The network was constructed according to the LeNet specification, using the parameter values shown in Table 1. The model consists of four layers in total: two convolutional, with subsampling, for feature extraction and two fully connected used for classification. In the training phase the MNIST training dataset was propagated through the network in 1000 iterations with the batch size of 120. When training, the classification accuracy was measured with cross-entropy loss (CEL), see Section 2.5.2 and Appendix A for the measured learning curve. The weights of the network were optimised with the SGD [11] and was initialised by Xavier initialisation [19]. When testing, the classification accuracy was estimated by the top-1 estimator, see Section 2.5.2. The model parameters were chosen according to Table 2.

The source code used to evaluate the LeNet model was developed in the spirit of Test-Driven Development (TDD). By writing automated tests we do not waste time to manually verify the software. We are also able to continue to develop without the fear of breaking something that already exists [20].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td>Xavier initialisation</td>
</tr>
<tr>
<td>Loss</td>
<td>Cross Entropy Loss / top-1</td>
</tr>
<tr>
<td>Batch size</td>
<td>120</td>
</tr>
<tr>
<td>Iterations</td>
<td>1000</td>
</tr>
<tr>
<td>Optimiser</td>
<td>SGD (with nesterov)</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.01</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>$1e^{-6}$</td>
</tr>
</tbody>
</table>

Table 2: Hyper-parameters of the LeNet model.

### 3.2 Applying and estimating the adversarial noise

Adversarial noise is the key ingredient when generating adversarial examples. Here, adversarial noise with different strengths was generated by the FGSM method, which was described in Section 2.6.1. The noise was then applied to each image in the testset of the MNIST dataset. By doing so, a dataset of adversarial examples was acquired. The new dataset is fed to an already trained model to evaluate the classification accuracy, which is done for different strengths of adversarial noise. Each time the model is fed the testset the prediction accuracy and adversarial noise is measured and plotted.

$$\text{dist} = \frac{1}{n \cdot h \cdot w} \sum_{k=1}^{n} \sum_{i=1}^{h} \sum_{j=1}^{w} |\hat{X}_{kij} - X_{kij}|$$  \hspace{1cm} (3)

We, as well as Uličný, measured the amount of injected adversarial noise with the average pixel difference $\text{dist}$, Section 2.6.4. In short, $\text{dist}$ measures the difference between images $X$, and the same images with adversarial noise $\hat{X}$. $\text{dist}$ is calculated in the same way as Uličný [9] did, but with a slight change. Since the MNIST digits are stored with one single colour band (grey scale), instead of three (which are used in coloured images), we can omit the ‘colourband variable’ ($c$) from Equation 2, and thus obtain Equation 3.

To verify the correctness of the adversarial noise the network’s performance was compared with the related work by Uličný [9]. In the comparison plot, Figure 7, the strength of adversarial noise is measured and plotted.  

\(^3\) His results were approximated and plotted in this graph. The result here may be a bit smoother, compared to the one in his report.
noise was increased and then applied to the test set of the MNIST dataset. The x-axis shows the estimated average pixel difference, calculated with Equation 3, and the y-axis shows the top − 1 accuracy. When comparing the two results it is seen that our ‘curve’ is not as steep as his, which is further discussed in Section 5.4.

![Diagram](image.png)

Figure 7: In the red curve we see how the MNIST dataset was classified in the thesis by Uličný. His result is compared with the one found in this thesis which results in the red line.

### 3.3 Choosing the distortions and varying their levels

The distortions that were picked to distort the adversarial examples were the following: contrast, skew, rotation and blur. These distortions are also chosen by others; For example, Dodge and Karam [5] have picked blur, noise, contrast and different types of JPEG-compressions and Kandi et.al [4] have chosen rotation as one of many distortions in their evaluation.

In this thesis, when each distortion was picked a max and min level of distortion were also picked, Table 5. The maximum value max of each distortion level was chosen so that the change was clearly visible, this was done empirically. The minimum value min for a distortion was selected so that each distortion did not change the original image. When using rotation min was 0 degrees, and with contrast min was set to 1.0. A number of values/samples n in the range of [min, max] were selected to estimate how the strength of each distortion affected the LeNet model. Each distortion that is used in this thesis is adjusted and changed to increase the level of said distortion. When using the Blur distortion we control the radius used when applying the distortion. The Contrast distortion is controlled by a factor which is relative to the original image, ‘1’ means that there is no difference and ‘2’ means that the contrast is increased by the factor of two. Regarding the Rotation distortion, the level corresponds to the angle which the image is rotated by, here the images are rotated in a counter-clockwise direction. When adjusting the Skew distortion, we change the shear angle (in radians) in counter-clockwise direction.

#### 3.3.1 Evaluating Single distortions

When evaluating the interaction between a single distortion and the adversarial examples we used five samples from the [min, max] interval, in hope to catch as many important observations as possible. For each level of distortion we gradually increase the strength of the adversarial noise to see how the distortions interacts with the noise. The accuracy of the neural network was measured with the top − 1 estimate. All four distortions are presented in Figure 8, here it is seen how they affect the the visual presentation of a digit. As a reference a non-distorted digit is also plotted.

---

Kandi et.al [4] explains that the direction of a rotation does not matter when distorting images. The accuracy loss is almost the same when rotating further than 180 degrees. This is the reason why we choose to rotate the adversarial examples in one direction.
Figure 8: In this example the digit '1' is distorted by different distortions. For each column the adversarial noise is increased, from $dist \approx 0$ to $dist \approx 0.4$, which are measured by Equation 3. From top to bottom; no distortion, gaussian blur, contrast, rotation and skew.

3.3.2 Evaluating the Combination of distortions

When choosing a larger number of distortion levels one end up with many more interactions to test, the evaluation time grows exponentially as the number of levels are increasing. When combining the distortions and evaluating their interaction with adversarial examples the number of samples, picked from the $[\min; \max]$, was decreased from five to three. To make it more concrete, the levels are referred as none, medium and strong, the precise meaning of each level of distortion is shown in Table 3. The amount of injected adversarial noise, measured with $dist$, see Section 2.6.1, is also referred as (none, medium and strong) presented in Table 4. The distortions were applied to the adversarial examples in an arbitrary order shown in Figure 9, the order was fixed and did not change throughout the evaluation. For each combination of distortion, the classification accuracy of the neural network was measured with the top-1 accuracy.

![Distortion Diagram]

Figure 9: The order which the distortions were applied in.

<table>
<thead>
<tr>
<th>Distortion</th>
<th>None</th>
<th>Medium</th>
<th>Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blur</td>
<td>0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Contrast</td>
<td>1</td>
<td>1.75</td>
<td>2.5</td>
</tr>
<tr>
<td>Rotation</td>
<td>0</td>
<td>17.5</td>
<td>35</td>
</tr>
<tr>
<td>Skew</td>
<td>0</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 3: Distortion levels that were used when combining different distortions.

<table>
<thead>
<tr>
<th>Distortion</th>
<th>None</th>
<th>Medium</th>
<th>Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial noise</td>
<td>0.0066</td>
<td>0.207</td>
<td>0.407</td>
</tr>
</tbody>
</table>

Table 4: Adversarial levels that were used when combining different distortions.
4 Result

The task was to evaluate how different distortions affected the classification of adversarial examples constructed from the MNIST dataset. When the LeNet model was evaluated top-1 was used to estimate its ability to classify. To understand how the distortion affects the accuracy the basic ‘no distortion’-reference is also plotted, which show the performance when no distortion is applied. First it was investigated how each individual distortion affected the accuracy of our model, Figure 10 and 11, Section 4.1. Afterwards the interactions between multiple distortions were taken into account and plotted in Figure 10, Section 4.2. As mentioned in the introduction, we set out to answer the following research questions:

1. How does a single distortion affect adversarial examples?
2. How does the combination of multiple distortions affect adversarial examples?

4.1 Single distortion - Our findings

In this section we present how each distortion affected the model regarding classification accuracy. The most notable things regarding the first research task are presented in Table 6, they are then discussed in detail and plotted in Figure 10 and 11. These plots shows the how the top-1 accuracy changes due to different distortion levels but also due to the estimated adversarial, mentioned in Section 2.6.1. In this section, it is shown that adversarial examples that have been previously classified with roughly 45% accuracy can be classified with accuracy up to about 75% thanks to the contrast distortion. It is also shown that the classification rate of examples, that have been classified with 0% accuracy, can be increased to about 10% by all distortions except contrast.

The strength of each distortion varied through the evaluation, all distortion levels are referred as none, medium and strong as defined in Table 3.

<table>
<thead>
<tr>
<th>Distortion</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial noise</td>
<td>≈0</td>
<td>≈4</td>
</tr>
<tr>
<td>Blur</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>Contrast</td>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>Rotation</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Skew</td>
<td>0</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 5: The minimum and maximum levels for all distortions and the adversarial noise.

Applying Gaussian Blur  The gaussian blur did not increase the accuracy for adversarial examples with no adversarial noise. Gaussian blur with around 0.1 radius increased the accuracy, but only for examples with none-medium adversarial noise. A strong gaussian blur seemed to increase the classification rate of adversarial example with high adversarial noise, the accuracy from 0% to about 9%. As the gaussian blur was increased, the accuracy dropped (from nearly

<table>
<thead>
<tr>
<th>Original accuracy</th>
<th>Increased to</th>
<th>By which distortion</th>
</tr>
</thead>
<tbody>
<tr>
<td>≈45%</td>
<td>65% - 75%</td>
<td>contrast</td>
</tr>
<tr>
<td>≈0%</td>
<td>≈11.5%</td>
<td>rotation</td>
</tr>
<tr>
<td>≈0%</td>
<td>≈7%</td>
<td>skew</td>
</tr>
<tr>
<td>≈0%</td>
<td>≈9%</td>
<td>gauss</td>
</tr>
</tbody>
</table>

Table 6: The most notable differences, regarding classification rates, when classifying adversarial examples with different distortions.
Figure 10: Performance plots when applying Gaussian Blur and Contrast to the adversarial examples.

80% to about 10%-50%; see Figure 10). It could be possible that the gaussian blur have removed some of the adversarial noise but also the original features, in a similar way that features are removed as stated in Dodge and Karam [5]. Zhou et.al [21] state that since MNIST contains handwritten digits, the most important features are the strokes and the edges along the digits. When applying a de-focus blur the edge information is getting weaker and thus making the digits harder to recognise. In their work, they show that the de-focus blur is a distortion that makes the digits hard to recognise. Zhou et.al also stated that the distortions have more effect on low-level information of an image. Dodge and Karam report a similar result: the blur distortions seem to remove textures or smaller features which are used to identify the image.

Applying Contrast  The contrast distortion can increase the accuracy from 45% up to 77%. In fact, the distortion managed to increase the classification accuracy of almost all adversarial examples, the only exception is when strong adversarial noise was applied to the adversarial examples, Figure 10. Dodge and Karam [5] have reported that a decrease regarding contrast does not affect the classification of images, except when the contrast have been lowered to almost nothing. All networks that they evaluated had some resilience against contrast. This is different from what was done here since the contrast was increased and not lowered.

Applying Rotation  Figure 11 shows that a strong rotation increased the accuracy of the model when applied to examples with medium-strong adversarial noise. When the adversarial examples
were rotated below 10 degrees the distortion did not manage to influence the accuracy, but it did when they were rotated above 10 degrees. When increasing the rotation the classification accuracy is gradually changing, images that were classified with high confidence are classified with lower confidence and vice versa. Kandi and Mishra [4] have shown that the classification rate of rotated digits is symmetrical around 180 degrees, in other words the accuracy is not dependent on the direction of rotation.

Applying Skewness  The skew distortion influenced the classification accuracy, in a manner similar to the gaussian blur. A gap, regarding accuracy, could be seen when comparing none and strong skew when no adversarial noise was used. A skew of 0.1 made it easier to classify none-medium adversarial examples, and a stronger distortion of 0.4 made it easier to classify stronger adversarial examples.

4.2 Combinations of Distortions - Our findings

In this section we present our findings when evaluating the interaction between the distortions see Figure 12 and 13. The levels of distortions and adversarial noise are presented in Table 4 and Table 3. The x-axis shows the accuracy measured by top-1, the y-axis shows the rotation. The size of each data point shows the contrast, where a larger size means higher contrast. Finally the colour of each data point shows how much blur or skew that was used in each combination. In Figure 12 we show rotation, contrast and blur and in Figure 13, rotation, contrast and skew. For each level of adversarial noise (none, medium and strong) the interactions between distortions are summarised below.

With Adversarial Noise set to: None  A medium-strong Contrast distortion increased the accuracy of the model when the medium-strong gaussian blur is applied, but was only able to nudge the accuracy without the influence by blur. The rotation made it harder to classify the digits, it did not matter whether another distortion was applied or not. All levels of the skew distortion decreased the classification accuracy, which is further decreased when applying rotation. The influence from the skew distortion resembles the one from blur, even they are completely different distortions.

With Adversarial Noise set to: Medium  The influence by the contrast distortion depends on the amount of blur, the accuracy was increased when increasing the contrast of medium-strong blurred adversarial examples. But the contrast did not change the accuracy when the examples were skewed. The network was more confident when classifying rotated adversarial examples but only when they were not affected by the strongest blur or skew. When applying strong skew, the examples were very hard to classify, examples with none-medium skew were classified with about 25% to 40% confidence.

With Adversarial Noise set to: Strong  When applying strong blur to adversarial examples they were easier to classify. However, when rotated by 30 degrees they were harder to classify and the non-blurred images were made easier to recognise. The influence from the contrast distortion was weakened the more the blurred examples were rotated. In almost all cases, rotation combined with skew made the examples easier to classify. The contrast distortion managed to change the accuracy, this is clearly seen when rotation is gradually increasing. The increased contrast made examples harder to classify, but when rotated by 35 degrees the influence was not as noticeable.

Combination of distortions: An overview  When evaluating the combination of distortions it is interesting to see and compare two extremes. As shown in Section 4 the main factor is the adversarial noise, therefore it is only necessary to investigate which distortions lead to the highest
and lowest accuracy scores for each adversarial level. The results are presented in Table 7 and 8. When inspecting the tables some trends could be clearly seen. One of the trends is that the contrast was set to none when the accuracy was low. On the other hand, when the accuracy was maximised, the contrast was never set to none. Even when combined with other distortions the rotation distortion influences the adversarial examples in its characteristic way, see Figure 11. In other words, medium-high rotation leads to higher accuracy but only when the adversarial noise is strong. The strong blur increased the accuracy but only when the adversarial strong noise was applied. The skew-distortion follows the same pattern as the blur distortion, which is thinkable since the single-evaluations are also similar, Figure 11.
Figure 13: Here, we plot each combination to get a clearer look on how the combination affects the prediction accuracy.

Table 7: For each adversarial noise level we present the distortion combinations that lead the lowest top-1 accuracy scores.

<table>
<thead>
<tr>
<th>level of adversarial noise</th>
<th>level of blur</th>
<th>level of contrast</th>
<th>level of rotation</th>
<th>level of skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>2.0</td>
<td>1.0</td>
<td>35.0</td>
<td>0.4</td>
</tr>
<tr>
<td>medium</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.4</td>
</tr>
<tr>
<td>strong</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

5 Discussion

5.1 The model and the dataset

To answer the research question, we used the MNIST data set which consist of uncoloured handwritten digits. It could be interesting to see if coloured images change the classification
Table 8: For each adversarial noise level we present the distortion combinations that lead the highest top-1 accuracy scores.

<table>
<thead>
<tr>
<th>Level of adversarial noise</th>
<th>Level of blur</th>
<th>Level of contrast</th>
<th>Level of rotation</th>
<th>Level of skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0.0</td>
<td>1.75</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>medium</td>
<td>0.0</td>
<td>1.75</td>
<td>17.5</td>
<td>0.0</td>
</tr>
<tr>
<td>strong</td>
<td>0.0</td>
<td>2.5</td>
<td>35.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

accuracy, to test this one can use two versions of the same dataset. Other datasets can also be used to build adversarial examples, for example the Labeled Faced of the wild [22] for facial recognition or CIFAR10 for image recognition.

The chosen neural network was the LeNet model, a network which is often used in combination of the MNIST dataset. The results produced here might be very unique to the model, we therefore suggest to explore other models. When classifying digits, none of the robustness methods mentioned by Uličný [9] were used, these methods might have helped when classifying distorted adversarial examples.

5.2 Affine transformations and image clipping

When applying distortions the image gets transformed and some information might get lost, since the image is clipped by the distortion, see Section 2.1. In the case of rotation, the corners are clipped and removed. This might be a problem, since the removed parts might contain important features which are used by the adversarial examples, as well as the model. It could therefore be discussed if we really have evaluated how the affine transformations affect adversarial examples, since these examples have been clipped. Meier et.al [23] Kandi et.al [4], Lokesh et.al [24] solved the ‘clipping problem’ by re-sizing the image before applying a distortion. Each image is then centred on a larger background, which allows the image to be rotated, moved etc. without removing edges or corners. To us, it is currently unknown whether the clipping is important or not when crafting adversarial examples. This could be left for further investigation in a future work.

5.3 The distortions

For each distortion, the strength was adjusted within a given range of values, Section 3. The levels were selected depending on what we thought was enough to answer the research question, in hindsight, the exact level for each distortion type could have been selected with some sort of support. The levels might not be relative to each other which can lead to a twisted conclusion, but the relatively wide range of values have probably caught the most probable values. When the distortions were applied to the adversarial examples, they were done so in a specific order, Section 3. The order was arbitrary and all distortions were applied in the same manner for all examples in the evaluation. It is possible that a different order would have lead to a different classification accuracy. Of course, there are other possible distortions to include, the most obvious one is image translation where the digit is moved around in the image. But this require that the image is moved in all four directions but also with different distances.

5.4 Possible explanation for differences between results

The difference between the result in this thesis and the one from Uličný, Figure 7, could depend on the training of the LeNet model. Uličný did not clearly specify how each hyper parameter was chosen, for example, the number of iterations and the batch size used when training the model. The lack of information of training parameters could be one reason to the difference when comparing our results. In the plot it is seen that the adversarial noise, generated by Uličný,
have a greater impact on the MNIST dataset, the accuracy drops much quicker compared to the performance that was achieved in this thesis. One could imagine that we have achieved the ‘adversarial effect’, only with a slight difference.

It is said that the neural network’s ability to generalise is taken advantage of when the adversarial examples are created. Assuming this is true, we can also assume that a network with a better ability to generalise is also able to classify adversarial examples with a higher accuracy. However, the classification of distortions is also affected by the generalisation ability of the neural network. The network’s accuracy is, generally speaking, increased with more training data, this is not the case of the adversarial examples. We then arrive at the following conclusion: A stronger model’s performance might be solely degraded by adversarial examples that interacts with distortions, rather than the distortions. Therefore, when comparing our result we might end up with a twisted conclusion, since the distortions might degrade Uličný’s model, assuming that our model have a better ability to generalise.

5.5 Further Research

More distortions: We have only experimented with a few types of distortions. Translation and warp distortions have not been evaluated together with adversarial noise, these types of distortions may affect adversarial examples and could be included in a future work. Another possible distortion is motion blur which could occur when the image is moved attached on a moving object.

Other predictive models: In this thesis we have evaluated the LeNet model which is fed distorted adversarial examples, but this is only one of many of neural networks. There are other networks which could be included in future work, for example; VGG16, Xception or Inception V3.5

Adversarial regions and distortions: It is possible to determine whether the input is an adversarial example or not [1], this is based on measuring how close the input is to the adversarial regions found in the feature space. We have shown that some distortions are able to increase the classification accuracy of adversarial examples, which could be explained by the fact, that some examples have been moved away from the adversarial regions. To determine if this is the actual case, one can measure the distance from each adversarial region to any given input and type of distortion.

6 Conclusion

Different combinations of distortions were applied to images which are intentionally hard to recognise, also known as adversarial examples. Here, it is shown that the distortions do indeed affect the network’s ability to recognise the adversarial examples. The result can be used as a stepping stone for how to further explain how the interaction between adversarial examples and distortions works. It was shown how each single distortion affects the classification rate of adversarial examples, but also how the interaction between distortions changed the accuracy. In the evaluation, it was shown that the classification accuracy depends on the type of distortion(s) and the strength of said distortion(s). Some distortions dramatically increased the accuracy meanwhile others did not make such a difference. A distortion’s characteristic influence on the model’s accuracy is still visible, even though it has been mixed with other distortions, for example, rotation and contrast.

Most of these are listed on https://keras.io/applications/, with Keras it possible to use already trained models instead of training them yourself.
How does a single distortion affect adversarial examples? In this thesis it is shown that each distortion is able to affect adversarial examples, in terms of classification accuracy. All distortions managed to make a notable impact on the classification accuracy. The most notable one is contrast since it did not decrease the accuracy at all and in some cases increased the accuracy by 30 percentage points. The other distortions, rotation, skew and blur, managed to increase the classification accuracy of adversarial examples that were classified with 0% confidence, however the accuracy did not increase when the contrast noise was applied. In almost all cases, all distortions apart from contrast decreased the accuracy when the accuracy was previously high.

How does the combination of multiple distortions affect adversarial examples? It is shown that all distortions are able to interact with each other. The result shows that the type and strength of the distortion affects the classification rate when interacting with other distortions. The distortions that left a larger impact on the classification accuracy are blur and skew. These two distortions managed to decrease the accuracy by 60 percentage points of examples that were classified with high confidence. Even though blur and skew were combined with other distortions they were able to increase the accuracy when it was previously low. The low accuracy was increased from $\approx 0\%$ by 10 percentage points, but only when a rotation of 35 degree’s was applied. In some cases, accuracy of $\approx 25\%$ was increased with $\approx 15$ percentage points due to the increase in contrast.
References


A Training the CNN

When training the neural network CLE (Cross Entropy Loss) was used to estimate and minimise the loss. The accuracy that was measured in the training phase is plotted in the following plot. For each epoch the accuracy was increased. The plot shows that the model reaches about 98% accuracy on the training set of MNIST. Even though the model is almost 100% correct it does not manage to classify some adversarial examples.
B Software and Hardware

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 16.10</td>
</tr>
<tr>
<td>Python</td>
<td>3.5</td>
</tr>
<tr>
<td>cudnn</td>
<td>5.1.10</td>
</tr>
<tr>
<td>Cuda compilation tools</td>
<td>release 8.0, V8</td>
</tr>
<tr>
<td>Keras</td>
<td>2.0.0</td>
</tr>
<tr>
<td>Tensorflow (gpu version)</td>
<td>1.0.1</td>
</tr>
<tr>
<td>GTX nvidia 660 (2GB)</td>
<td>Driver Version: 375.39</td>
</tr>
</tbody>
</table>

Table 9: Software/Hardware and corresponding version that were used when evaluating the LeNet model.