Universiteit Leiden
The Netherlands

Bachelor Computer Science

Understanding and Discussing Weaknesses in Deep Neural Networks

Vera Schous

Supervisors:
Michael Lew
Walter Kosters

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

09/09/2019

Abstract

Deep Neural Networks have grown very popular over the years since they became more feasible in the 2010s. However, they are still quite hard to comprehend. Artificial intelligence and machine learning have become buzzwords, but are they really the solution to most current computer science problems as so many people seem to think or are they actually easily deceivable? In this paper we will look at manipulation of images and the effects these cause on accuracy. This should give us some insights into these mystical models. We have used rotations, added noise and inverted the colours of the images. Adding noise reduces the accuracy the most and interesting patterns arose for rotation.

Contents

1 Introduction **1**
- 1.1 Thesis overview . . . 1

2 Neural Networks **2**
- 2.1 Definitions . . . 2
- 2.2 VGG16 . . . 4
- 2.3 ResNet . . . 5
- 2.4 Xception . . . 8

3 Methods **10**
- 3.1 TensorFlow . . . 10
- 3.2 Python . . . 10
- 3.3 Dataset . . . 11
- 3.4 Libre Office . . . 11

4 Related Work **12**

5 Experiments **13**
- 5.1 Structure of research . . . 13
- 5.2 Rotations . . . 13
  - 5.2.1 Detailed approach . . . 14
  - 5.2.2 Results . . . 14
- 5.3 Distortions . . . 17
  - 5.3.1 Detailed approach . . . 17
  - 5.3.2 Results . . . 18

6 Contributions **19**

7 Conclusion **20**

8 Future research ideas **20**

References **21**

A Additional Figures **22**

1 Introduction

Since the beginning of programming Artificial Intelligence has grown to be a buzzword. What makes AI intelligent? AI often works with neural networks, but what is a Neural Network and how do they work? Neural networks are based on the networks of neurons in the brain. Similar to networks in the brain, these (artificial) neural networks are able to learn from mistakes and get 'smarter' along the way. One can teach a neural network to learn a certain pattern and recognise images and the category they belong to.
We will start with the basics of Neural Networks in order to reveal our problem and goal. Deep neural networks are deeper (have more layers) than basic neural networks. These networks are often compared to black boxes because people do not know how they work specifically, only that they do and are able to learn from data in a certain way using different layers.
In order to be able to comprehend more of the internal functioning of a deep neural network one can test its power by changing the input and analysing the difference in output. This is the core of this thesis. We will be manipulating colour images and discuss the differences in accuracy these manipulations cause.
In contrast to current existing research we will compare the performance of three state of the art CNNs to each other for added noise and inverting the colours. Randomly rotating images has been done before. However, we will rotate our images for a full 360 degrees in steps of 10 degrees. Thus, our focus for rotation is on what degree of rotation still has the highest accuracy in order to be able to understand how these CNNs work.

1.1 Thesis overview

Section 2 includes the definitions needed for this thesis and a small introduction to the history of neural networks; Section 3 is about the methods that have been used; Section 4 discusses related work that has already been published; Section 5 describes the experiments we have done and their results; Section 7 concludes this thesis. Section 8 suggests future work ideas that have come to light in writing this thesis. After this there is an appendix with additional figures.

This bachelor thesis is part of the LIACS bachelor Computer Science at Leiden University. The supervisors for this bachelor thesis are Michael Lew and Walter Kosters.

2 Neural Networks

In 1943 a computational model for neural networks was created, based on the neural network of the brain (biological neural network). This paved the way for the use of Artificial Neural Networks (ANNs) in computer science. Typically, neurons are organised in layers. Different layers can perform different types of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, the layers can be entered multiple times or even "skipped".
A Convolutional Neural Network (**CNN**, also abbreviated as ConvNet) is a special kind of multi-layered neural network used for recognising of visual patterns from pixel images.
Deep learning is part of a family of machine learning methods based on ANNs. Deep Neural Networks (**DNNs**) and Convolutional Neural Networks are Deep learning architectures. DNNs are called deep due to the amount of layers used in their architectures.
For the purpose of this thesis we will look at three Deep Convolutional Neural Networks (**DNNs**) and compare their accuracy in several experiments. The three networks we will look at are **VGG16**, **ResNet** and **Xception**. These models will be discussed in the order they were created. All three CNNs can be found trained and ready for testing with Keras and TensorFlow. An illustration of a relatively simple CNN can be seen in Figure 1.
ImageNet is a large image database created for use in visual object recognition software research. There is an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**), where (deep) Convolutional Neural Networks compete to achieve the highest percentage of correctly classified objects and scenes in images from this database. This annual contest was started in 2010. The ImageNet database was one of the first large databases with classified objects that could be used in object recognition.

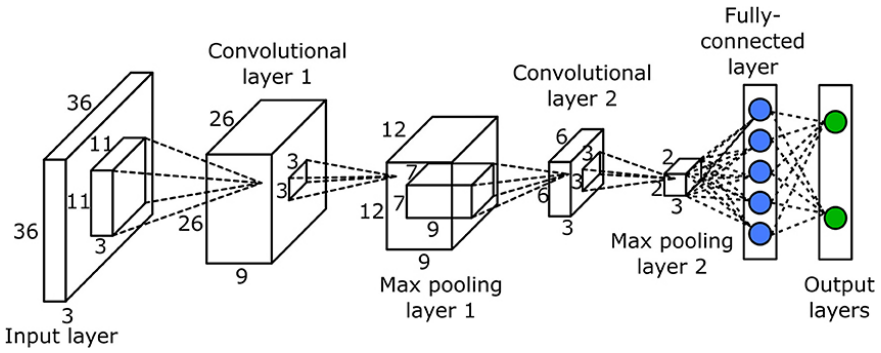


Figure 1: Example of a simple Convolutional Neural Network [HMPK].

2.1 Definitions

Before we look at the three DCNNs mentioned above, we will describe some of the basic terminology used in building Neural Networks.
Channels are a different view of the image as a whole, placing emphasis on certain aspects. This is also referred to as the width of the layers. An example is the red, green and blue channels of a RGB image.

Kernel/filter/feature detector are multiple names for the same small matrix of weights, the kernel moves over input data, performing operations that are specified by the architecture. The size of the kernel determines the size of the output of a certain layer, if the kernel is 3×3 the output is 3×3 as well (unless stride and/or padding was used).
Stride is used to skip some of the slide locations of the filter. A stride of 1 means to pick slides one pixel apart, which is just a standard convolution. A stride of 2 means picking slides 2 pixels apart, skipping one slide in between everytime, downsizing the output by a factor 2. An example can be seen in Figure 3.

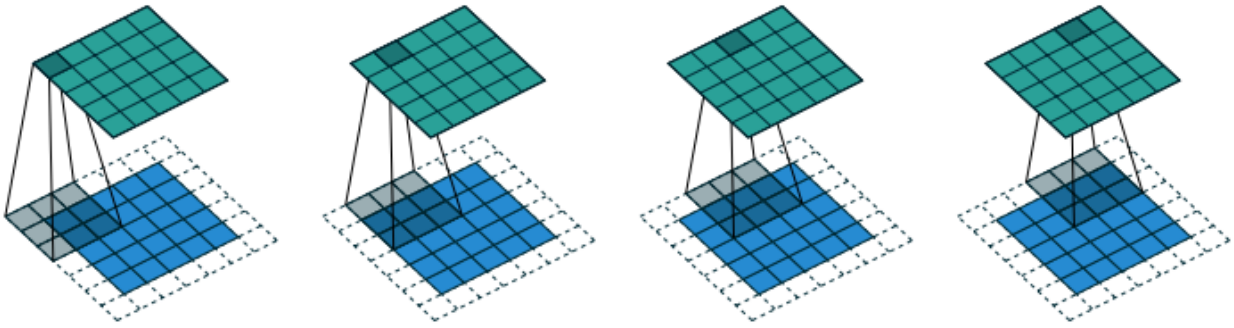


Figure 2: Example with padding 1, the gray area is the kernel [DV18].

Padding is used to protect against shrinking outputs and loosing information on the corners of an input image. One can add a padding around the edges of the input; these are "fake pixels (usually of value 0). In doing this, the kernel can allow the original edge pixels to be at the centre of the kernel, thus gathering as much information from them as is accumulated from the middle pixels. The size of the padding describes how many rows of pixels are added around the input. Figure 2 shows an example for padding 1.
Fully-Connected layers create an output the size of the weighted product of every single input feature, for input of 3×3 and 5×5 this gives $3 \times 3 \times 5 \times 5 = 225$ weight parameters.
Min/Max pooling is selecting the minimum or maximum value respectively of an area of values. An example of max pooling can be seen in Figure 3.
Average pooling is summing the values and dividing it by the total number of values; this method is not implemented frequently.

Figure 3: 3×3 max-pooling with 1×1 strides on 5×5 input [DV18].

2.2 VGG16

The Visual Geometry Group (VGG) [SZ15] of Oxford had a new idea. They investigated the effect of the convolutional network depth on its accuracy. They greatly increased depth using very small (3×3) convolutional filters and achieved a depth of 19 layers. This increase in depth was only feasible due to the use of the very small convolutional filters in all layers. During training, the input of the VGG network was a fixed 244×224 RGB image. The 3×3 filters have a small receptive field, which is the smallest possible size that can capture the difference between left/right, up/down and middle. Another building block is max-pooling layers which are used for spatial pooling; this is performed with 2×2 pixel windows with stride 2. The last basic part are Fully-Connected (FC) layers, three after each other at the end of the network. The first two have 4096 channels each and the third has 1000 channels (corresponding to the number of classes) used for the ILSVRC classification. For this experiment we only needed 100 classes to be classified. The VGG made many configurations (A-E) which varied from 11 to 19 weight layers. For the experiments in this thesis we used VGG16, configuration D, which can be seen in Figure 4. The number of channels are combined into the layer names with conv3 indicating the 3×3 sized convolutions. In this figure the components that are the same for all 6 configurations are made bold and all the parts that can vary per specific configurations are in red. There are many parameters used for achieving this large depth, but at this time it was similar to the number of weights in a more shallow (less deep) net.

CNN Configuration D
16 weight layers
input (224x224 RGB image)
conv3-64
conv3-64
maxpool
conv3-128
conv3-128
maxpool
conv3-256
conv3-256
conv3-256
maxpool
conv3-512
conv3-512
conv3-512
maxpool
conv3-512
conv3-512
conv3-512
maxpool
FC-4096
FC-4096
FC-1000
soft-max

Figure 4: VGG16 network configurations D, based on Table 1 in the paper [SZ15].

While this was one of the first CNNs to use small 3×3 convolutional layers it is necessary to point out that stacking two of these small layers has an effective receptive field of a 5x5 convolutional layer. Three such small layers have a 7×7 effective receptive field. This stacking ability makes the classification function more precise and decreases the number of parameters. A 3×3 sized convolutional layer is the smallest to capture direction, but 3×3 convolutional layers are used as a linear projection onto the space of the same dimensionality [SZ15] (when the number of input and output channels is the same).

As this was one of the first deeper CNNs this was the first proof that increasing depth leads to better performance on accuracy.

During the training of these networks images had a fixed size but were randomly cropped, randomly

flipped horizontally and even underwent random RGB colour shifts. More on this can be read in Section 5. Another variation was introduced during the training stage of this CNN: "scale jittering", a process where the size of the images was varied. This seemed to be helpful for capturing statistics of different sizes of images, leading to better overall accuracy.
These VGG networks were entered in the ILSVRC challenge of 2014 and lost to GoogLeNet (also known as Inception) by a breath.
We have chosen VGG for its simplicity in structure and high accuracy nonetheless. There are 6 configurations of VGG published in [SZ15] of which we use one of the two versions with 16 layers. The version we have chosen has only 3x3 convolutional layers, whereas version C has 3 1x1 convolutional layers instead of the last conv3-256, conv3-512 and conv3-512. We have chosen this version because it is as good as VGG19, as can be seen in Figure 5, but faster in classification. We will from this point refer to VGG16 as simply VGG.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Figure 5: VGG configuration comparison for multiple test scales and train "scale jittering". Configuration D is VGG16 and configuration E is VGG19, this Figure is Table 4 in [SZ15].

2.3 ResNet

The official name of the ResNet version we use is ResNet50 (ResNet is short for Residual Network, 50 is the number of layers [HZRS15]) but for the sake of simplicity we will call it ResNet in this thesis. Residual translates to "an internal aftereffect of experience or activity that influences later behaviour." The Residual blocks influence later behaviour.
This deep convolutional neural network competed in the ILSVRC in 2015, like VGG many versions were submitted (e.g., 152, 110, 50 layers). With this architecture [HZRS15] were able to train a Neural Network with as much as 152 layers which. It achieves a top-5 error rate of 3.57% which beats

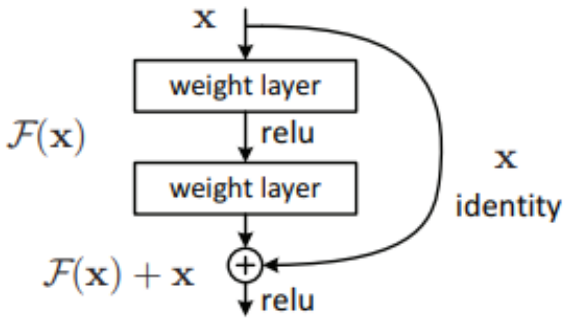


Figure 6: ResNet building block [HZRS15].

human performance on this dataset.
This architecture is thought to be able to learn edges (low), shapes (medium) and objects (high-level features) in different levels. The first paper on ResNet [HZRS15] showed that just adding more layers, which was thought to be the solution to getting higher accuracy, actually got worse when layers are added naively. A solution that ResNet introduced was adding more layers of a special kind: identity mapping layers. This suggests that a deeper model should produce no higher training error than its counterpart with less layers (shallow counterpart). Identity functions are however hard to learn, thus residual layers are introduced as seen in Figure 6. As can be seen, two layers are "skipped". This was the first introduction to this new form of architecture with skip connections. Such skip connections are also known as gated units, gated recurrent units or residual block. For these residual layers we will need to pay close attention to input size changing over different layers. There are at least two ways of making sure the input is being adjusted to fit the next layer. The first is that the skip connections still perform identity mapping, but with extra zero entries padded to increase the dimensions. The second option is to make the projections match the dimensions using a 1×1 convolution. Downsampling, making the input smaller, can be done with stride 2.
There is still a limitation of the amount of layers that can be stacked using the residual block, the paper states that with 1202 layers the error % actually goes up compared to 110 layers. Thus, having over 1000 layers is a problem, and this is hypothesised to be due to overfitting.
In Figure 7 we see how a different amount of layers for ResNet changes the accuracy compared to other networks. In Figure 8 one can see how VGG and ResNet architectures can be compared with each other. ResNet34 is used here as comparison since ResNet50 would not fit on the page and this figure shows VGG19 instead of VGG16. Thus the difference in number of layers of the ResNet and VGG versions we have used is even larger than this image suggests.

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Figure 7: ResNet comparing the number of layers and accuracy [HZRS15].

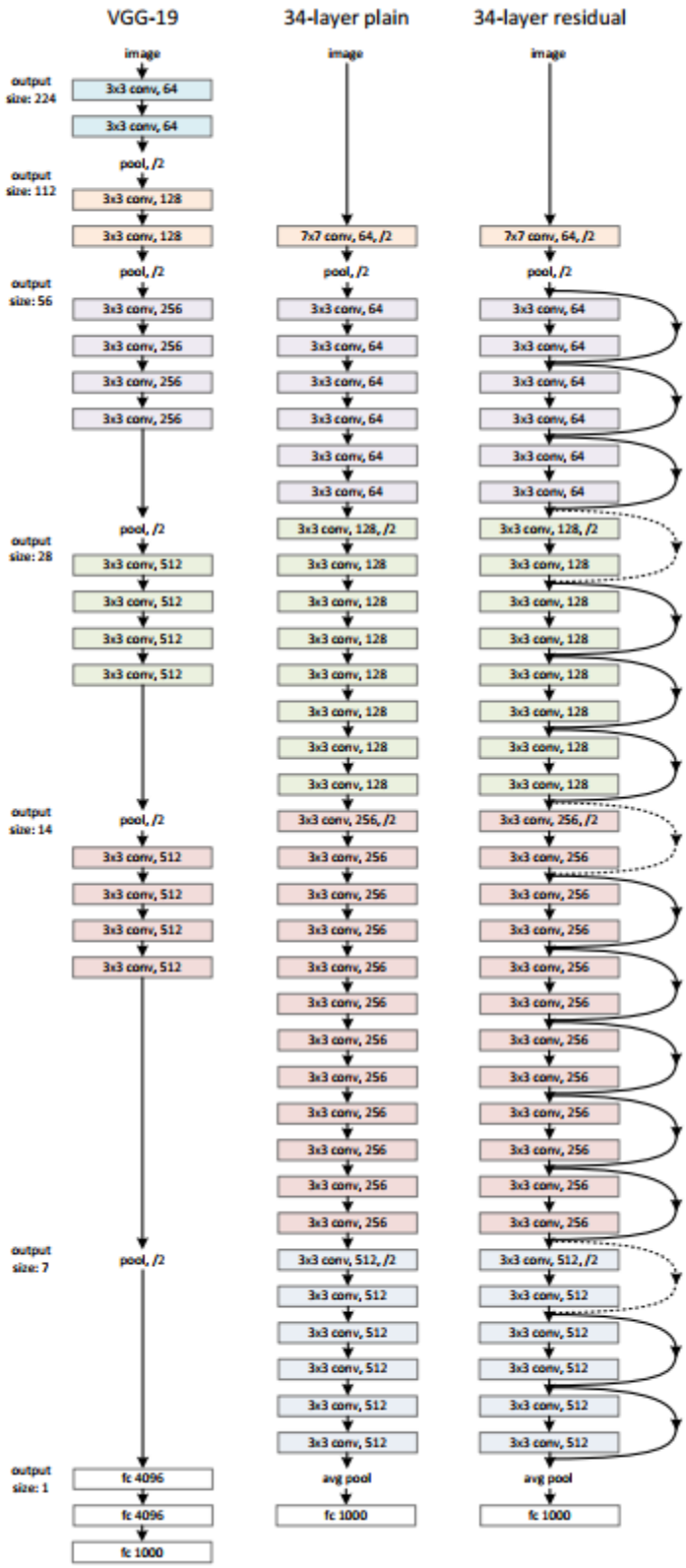


Figure 8: Comparing VGG19 and ResNet34 [HZRS15]

2.4 Xception

Inception [SLJ+14], as mentioned before, was the winner against VGG (and others) in the 2014 ImageNet challenge. Since then, many CNNs use the concept of an inception module in their own model, creating a family of inception based models. Inception was a new style of network in 2014, because of these inception modules. In Figure 9 we see this building block.

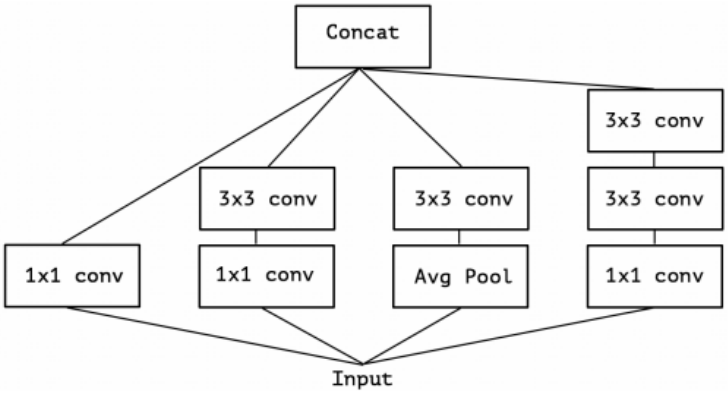


Figure 9: Inceptionv3 module [Cho17].

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

Figure 10: Comparing the performance of all three CNNs mentioned [Cho17].

Inception modules are convolutional feature extractors, empirically proven to be able to learn better representations with less parameters.

Xception was proposed by Fran cois Chollet, the creator and chief maintainer of the Keras library [Cho]. Xception stands for "Extreme Inception". Xception is an alteration of the Inception architecture [SLJ+14] which replaces the standard Inception modules with depthwise separable convolutions, "separable convolution" for short. A separable convolution is a depthwise convolution (a spatial convolution performed *independently for each channel*) followed by a pointwise convolution (a convolution projecting the channels output onto a new channel space). Since Xception is inspired by Inception, most of the experiments in the Xception paper are a comparison with Inception.

A convolutional layer attempts to learn filters in a 3D space, with a width and a height and a channel dimension. Thus, it has the task to simultaneously look at cross-channel correlations and spatial correlations. The inception module was the first step towards making this easier. Cross-channel correlations and spatial correlations are independent enough to look at them separately. Using multiple 1×1 convolutions we map cross-channel correlations and with regular 3×3 and 5×5 convolutions we map spatial correlations.

Xception, the concept of stacking separable convolutions, is not only based on the inception architecture but also has residual connections (as used in ResNet). The Xception architecture has 36 convolutional layers forming the feature extraction part of the network; this is followed by a logical regression layer. The 36 convolutional layers are spread over 14 modules, which all have linear residual connections connecting them.

In Figure 10 we can see the networks we discussed so far in comparison to each other. As expected, since the ImageNet competition is held each year, the accuracy becomes higher with each new network that is introduced.

The flow of the Xception architecture can be seen in Figure 11. This is the complete description of the network. We enter the flow diagram through the entry flow, the middle flow is next and repeated 8 times before entering the last (exit) flow. Note that all Convolution and Separable-

Convolution layers are followed by batch normalization, which is not included in the diagram. All SeperableConvolution layers use a depth multiplier of 1.

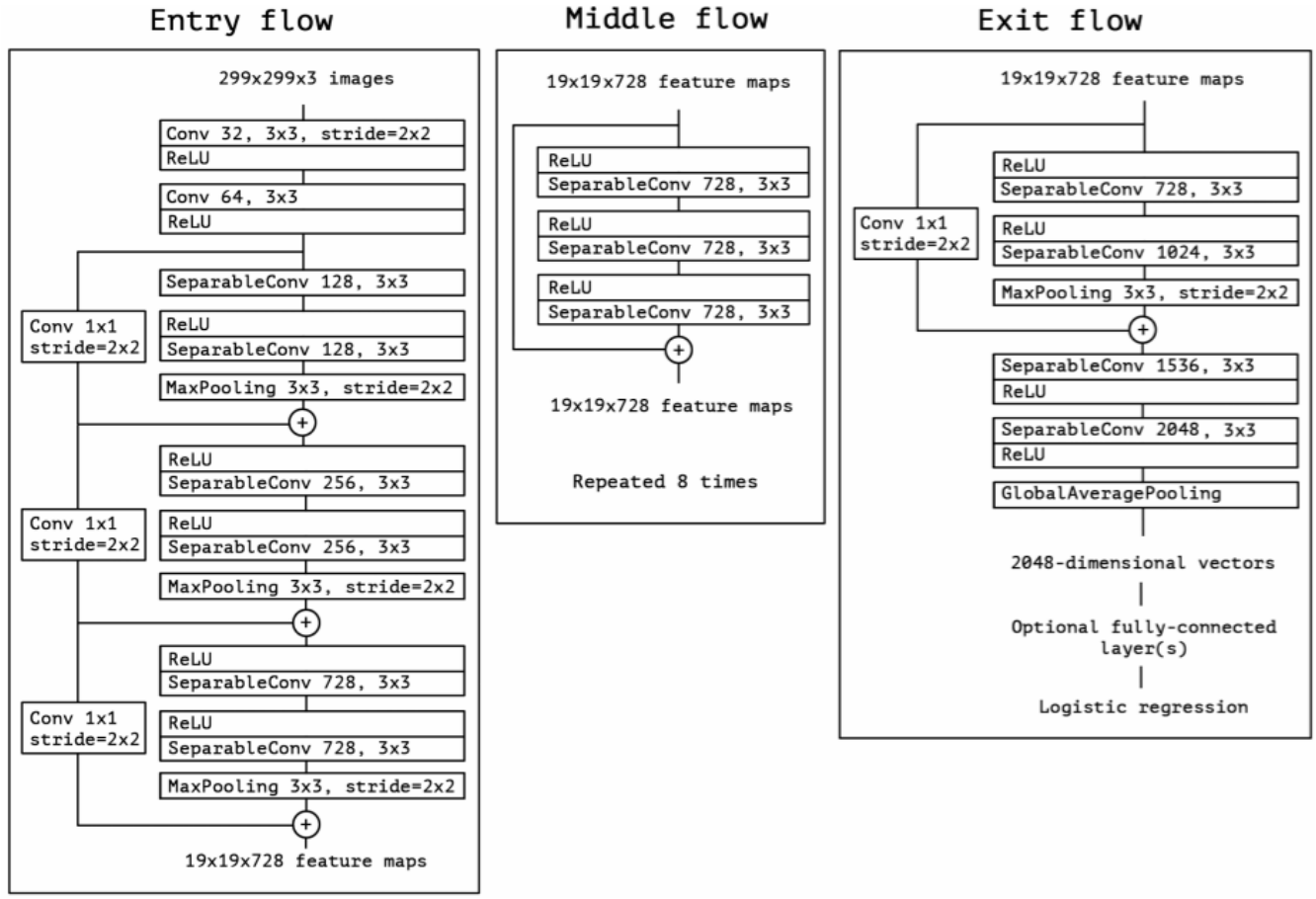


Figure 11: Xception: flow of the network [Cho17].

3 Methods

In this chapter we will look at the different kind of methods, tools and libraries that were used. We will also discuss why they were chosen and what we accomplished using these methods and tools.

3.1 TensorFlow

There are several platforms which one can use to create machine learning models. TensorFlow [Tea] has been around since november 2015 and was created by Google, with the initial goal of internal use by Google.
TensorFlow is open source and easy to use. For this project we have chosen to use TensorFlow with Python; more on this in the next section.
We have chosen TensorFlow because it is aimed at research and it works well on images. Caffe [Jia] was an alternative we looked into, but it felt more difficult to start with and less intuitive. It has also been called "cumbersome for big networks (GoogLeNet, ResNet)", which are exactly the CNNs we wanted to use in this thesis.
Keras is a Python based library which can be integrated into TensorFlow. We have chosen to use this library as well since it expands the functionality of TensorFlow greatly.
Some of the important libraries are listed in the next section, since they are extensions of Python.

3.2 Python

Python is the programming language we used, Python3 is used for TensorFlow and Keras. Some of the libraries we used are listed below, with their useful aspect(s) listed next to it:

- JupyterLab: a web-based user interface for Project Jupyter, a nonprofit organisation. This was used for running the models and collecting the data used in our results.
- matplotlib: used to make graphs and charts.
- NumPy: a library for Python, adding support for large, multi-dimensional arrays and matrices, accompanied by a collection of high-level mathematical functions to operate on these arrays.
- OpenCV-Python: used for (basic) operations on images.
- Pandas: used for dataframes and easy data manipulation.
- SciKit-image: used for rotation of images with mirrored corners, more on that in section 5.
- SciKit-learn: a python based library which is designed to interoperate with NumPy and SciPy; it is also built on matplotlib. SciKit is short for SciPy Toolkit and is a toolkit for data analysis.
- SciPy: a library for Python used for scientific computing and technical computing. SciPy contains modules for optimisation, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. SciPy builds on the NumPy array object described above.
- tqdm: used for showing a progress bar when running the models.

3.3 Dataset

CIFAR is the abbreviation of the Canadian Institute for Advanced Research. The CIFAR-100 dataset [Kri] consists of 60000 32×32 colour images in 100 classes containing 600 images each. There are 500 training images and 100 test images per class. We have enlarged the images to be either 224×224 (for VGG16) or 299×299 (for ResNet and Xception) so that they match the expected input of the networks we use. A subset of CIFAR-100 images can be seen in Figure 12.

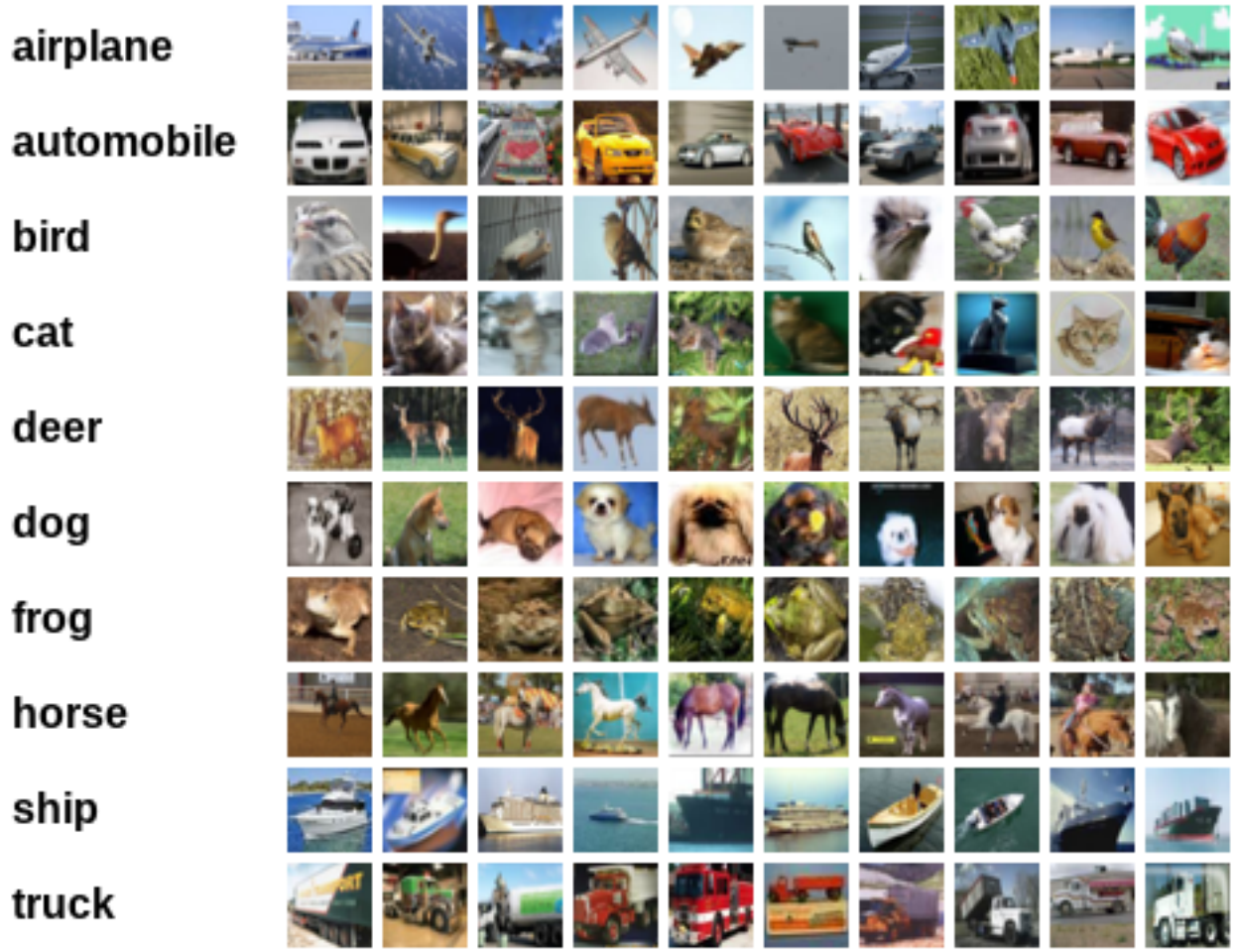


Figure 12: A subset of CIFAR images in their original size [Kri].

3.4 Libre Office

Libre Office was used for structuring and analysing the results, obtained using the TensorFlow image analysis algorithm, even further. The results were put together and normalized (for rotations) using Libre Office Calc. Most graphs were also made using Libre Office Calc.

4 Related Work

Deep neural networks have been shown to be vulnerable to adversarial examples (very small perturbations of the input having a dramatic impact on the predictions) [DMM18]. In this paper the authors researched newly trained rotation-equivariant CNNs. These networks are significantly less easily tricked by translations, rotations and the combination of both. Many groups had the same idea and did independent research into this same area. This paper examines three different CNNs: Group Equivariant CNNs (G-CNNs [CW16]), Oriented Response Networks (ORNs) and Harmonic Networks (H-Nets). G-CNNs use mathematically defined symmetry groups, H-Nets use complex-valued filters limited to circular harmonics, and ORNs actively rotate their filters during convolutions.
The experiments were done using three datasets: MNIST with simple CNNs with seven 5×5 convolutional layers vs H-Net vs G-CNN, CIFAR-10 with (G-)ResNet-44 and ImageNet with (OR-)ResNet-18.
On the first two cases the G-CNN version of the networks performed best. On the ImageNet there was no G-CNN version and the OR-ResNet outperformed the original in all but simple translation. Thus, G-CNNs overall outperform their original non-equivariant counterpart. This paper put the alternative CNNs to the test for translation and rotation but does not focus on the degrees of rotation and the effect this has on results. It simply states an overall higher accuracy with these new type of CNNs when random geometric manipulations of input data have occurred. In our thesis the focus is on the effects of different degrees of rotation on three different state-of-the-art architectures.

Another group of researchers from MIT, Massachusetts [ETT+18] researched the same manipulations of images (rotation, translation, R+T). They used the same three datasets as the group from Paris and discussed the benefit of increasing accuracy vs. the cost of increased training. Their idea is to introduce random transformations (rotation, translation) into the training data, then take a majority vote of the predicted label and use this as the actual classification. The images have a maximum amount of pixels they can move (5% of the image size) and a maximum of 15 degrees of rotation to both sides.
For the MNIST dataset they used a simple CNN, for CIFAR-10 they used a standard ResNet and for ImageNet they used ResNet-50. They had five variations of training these models.
Adding random transformations to the training data increased the general robustness but does not completely fix the problem of lower accuracy. The use of classification using majority voting can bring the accuracy up even more. This paper mainly focuses on ResNet, we will broaden the scope to two other CNNs using CIFAR-10s larger alternative CIFAR-100 and more degrees of rotation.

5 Experiments

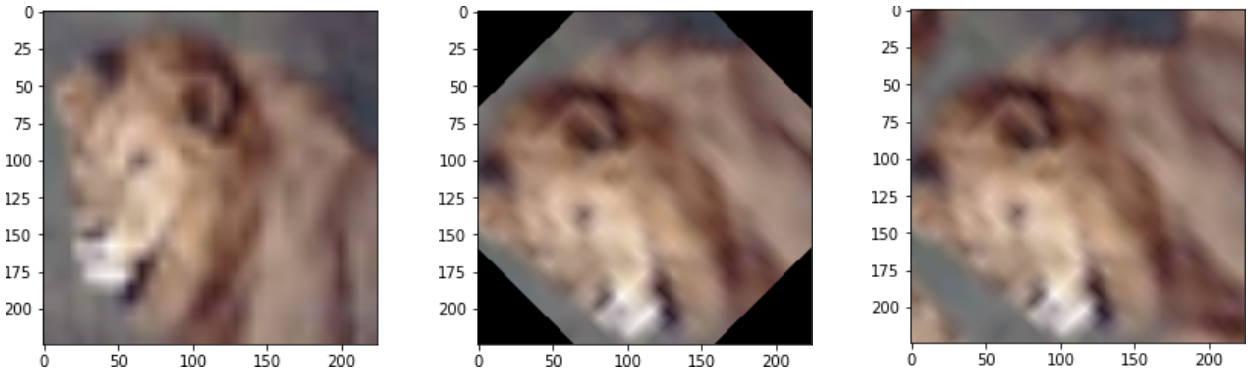
All images are preprocessed using multiple methods in order to see what makes a difference in the average classification of the 10000 testing images of the Cifar-100 dataset. All experiments are done on these 10000 testing images for all three deep convolutional neural networks: VGG16, ResNet and Xception.
First we started with manually researching a small set of images, to get an idea of what the experiments might point out. The first thing that stood out was that some CNNs have trouble recognising images that are upside down, while others seem to be able to recognise them nonetheless. In this section we will first look at the broad structure of how we did the research and then we will look more in depth into the different manipulations of images and the results on accuracy using these different manipulations.
We have chosen to look at rotation since it is one of the most commonly occurring manipulations of images in the gathering of data. When we look up images of planes they might not show a plane that is perfectly aligned in the way it would be recognised best. Thus looking into what degree of rotation tricks the state of the art CNNs most is very interesting.
As for distortions, images may not be the best quality (have noise) and it is interesting to see what noise does to the accuracy of predictions. An image with inverted colours does not occur randomly, however, it is interesting to research what changes in colours do to the accuracy of images and how different CNNs react to images in unusual colours schemes.

5.1 Structure of research

We started out by looking at pictures manually, rotating them or changing contrast in them and looking for differences in accuracy. This expanded towards using a dataset and writing a small algorithm that can manipulate images; more on this in the next sections. After having done the research on rotation of ResNet, VGG and Inception we found that the results of Inception were so unusual and unusable that we decided to look for a new CNN. Inception was thus exchanged for Xception and the results gathered using this network were more in line with what we came to expect after having researched ResNet and VGG. Inception classified almost everything as a web_site, as seen in Figure 22 in the appendix, which simply cannot be true as we know from VGG and ResNet. Results gathered from this earlier research with Inception can be found in the appendix in Figures 20 and 21, with VGG results for comparison. The results of the three other networks can be found under the corresponding image manipulations in Figures 14, 15, 16 and 17, and Tables 1, 2 and 3.

5.2 Rotations

The first experiment is rotation of each image with 10 degrees steps and the aim is to find the classification label that corresponds to the no rotation state (0 degrees, example of a Lion in Figure 13a). Each image is rotated using the skimage.transform rotate function [SI]; the default setting of this rotation function leaves the corners of the rotated image black as seen in Figure 13b. We have chosen to reflect the corners in order to make them blend in more with the image, which can be seen in Figure 13c.
The first label (0 degrees) was taken as the "truth" and for all other rotations the prediction



(a) The original (b) Rotation of 45 degrees (c) Rotation 45 degrees with reflected corners

Figure 13: Lion, img[66] of CIFAR 100 dataset.

corresponding to this label was taken as a result. All 36 results per image were saved and an average was calculated over all 36 rotations of the 10000 test images.

5.2.1 Detailed approach

When doing experiments with rotation we first load the CIFAR100 dataset using

$$keras.datasets.cifar100.load\_data(label\_mode = "fine").$$

We then resize the images to a Keras compatible format. We then load in the models we want to use (VGG16, Xception and ResNet50) and set up those models with the weights Imagenet has provided; this is integrated in TensorFlow and will be downloaded the first time a model is used. The models we used are pretrained and available for testing in Keras.
After the preparations are done we can rotate the images with reflected corners as described in the previous section. We use the model.predict(image) function to get all predictions for the images in CIFAR100 and their rotations. From the array of predictions we take the first top prediction of each image as the truth and find this label and its accuracy in all rotations of this image. We save the results of the label, the model and the accuracy corresponding to the "true" label.
Using this data we can calculate the average accuracy of all images per degree of rotation. This data was normalized so that the 0 degrees rotation has a 100% accuracy score; this way we are able to compare the different models and their peaks and low points.
We have also plotted multiple accuracies of the images we rotated in one image as can be seen in Figures 20, 23 and 24 in the appendix.

5.2.2 Results

The results were then normalized so we can compare the three networks. The average results of all images can been seen in Figure 14 for VGG16, in Figure 15 for ResNet and in Figure 16 for Xception. In order to be able to compare the rotational accuracies in the most precise way we have plotted a graph of all the differences of these three CNNs as seen in Figure 17.
On inspection we see that VGG and Xception are most similar, their difference in accuracy has not

been higher than 0.08 during the full rotation. However, ResNet and Xception are most similar on the highest peaks (90, 180, 270 degrees). This is due to the fact that VGG scores the highest accuracy on these quarter rotations. The peak of VGG at 180 degrees is due to random flipping input data horizontally in the training stage, resulting in a better recognition of images that are upside down.

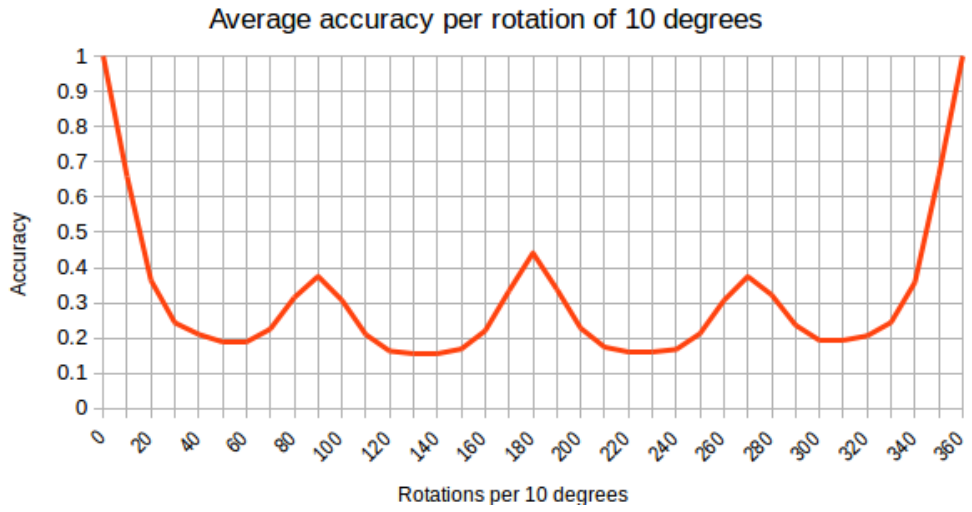


Figure 14: VGG16 average for rotation.

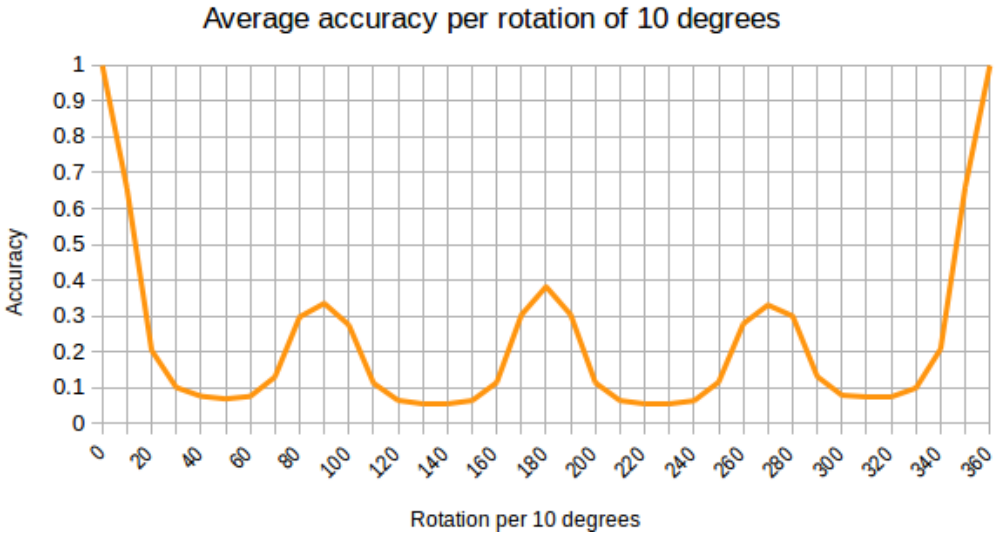


Figure 15: ResNet average for rotation.

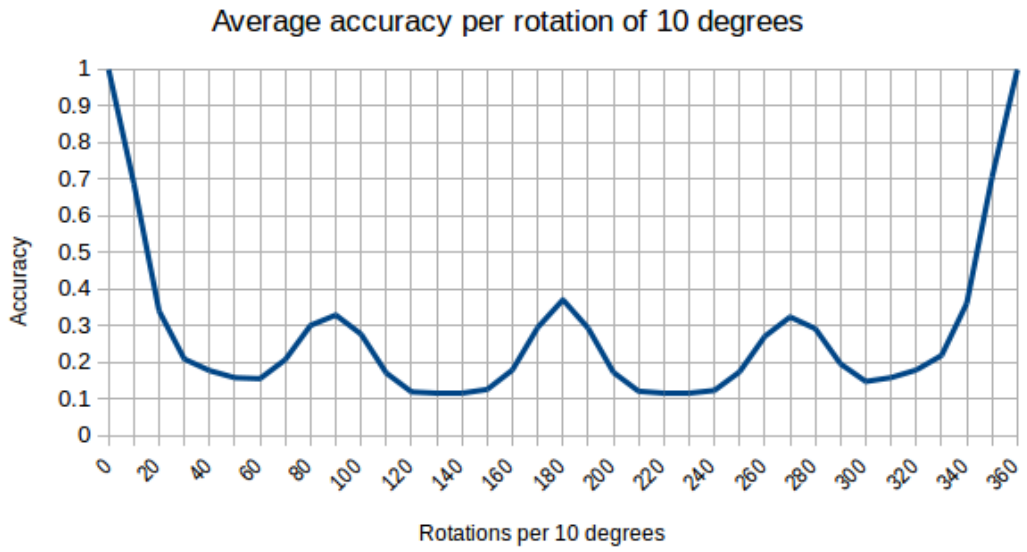


Figure 16: Xception average for rotation.

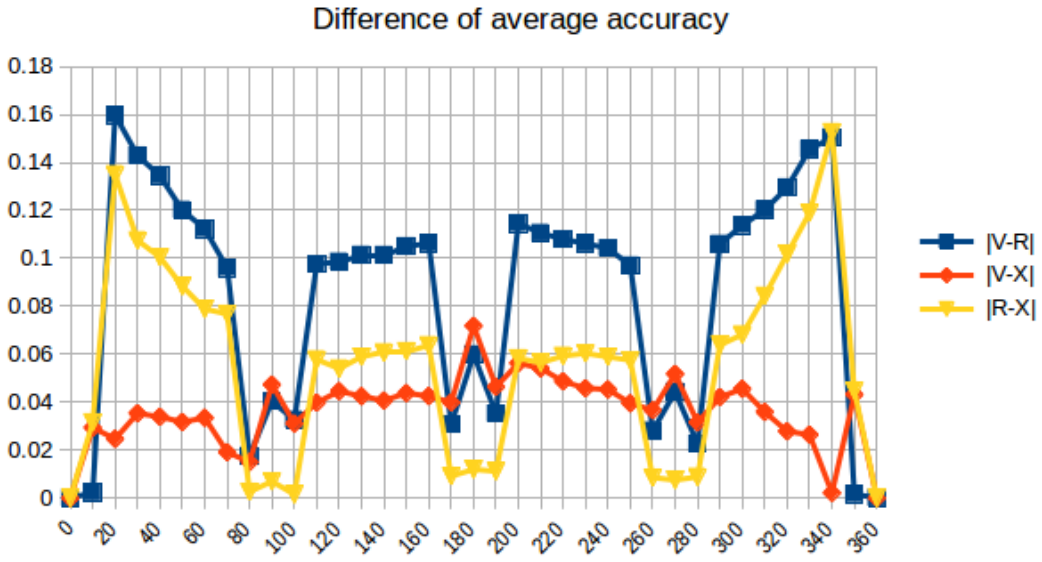
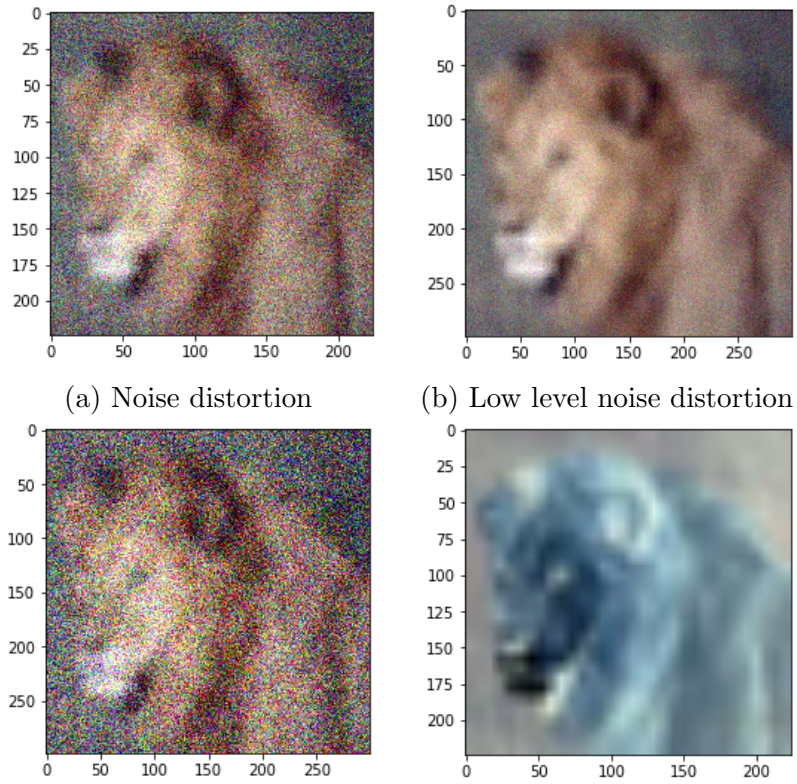


Figure 17: Differences in average for rotation.



(a) Noise distortion

(b) Low level noise distortion

(c) High level noise distortion

(d) Inverted colours

Figure 18: Lion, img[66] of CIFAR 100 dataset.

5.3 Distortions

Adding of noise (Figure 18a, 18b and 18c) and inverting the colours of the images (Figure 18d) were used to test another aspect of the robustness of CNNs: how they deal with distortions instead of rotations (or translations). There may be more possible distortions to look into in the future.

5.3.1 Detailed approach

When doing experiments with distortions we had a similar approach as described in the section on rotation.
After the preparations are done we can manipulate the images with noise or by inverting the colours of the whole image. For noise we used the function random_noise(image, var=sigma**2) with sigma = 0.155. For inverted colours we used the function invert(image).
We use the model.predict(image) function to get all predictions for the images in CIFAR100 and their distorted version. From the array of predictions we take the first top prediction of each image as the truth and find this label and its accuracy in the manipulation of this image. We save the results of the label, the model and the accuracy corresponding to the "true" label.
Using this data we can calculate the average accuracy of all images per degree of rotation.
We have also plotted multiple accuracies corresponding to the images we distorted in one image as can be seen for VGG16 in Figures 34 and 35 in the appendix. Mind the different scale of the vertical axis.

5.3.2 Results

The average results of all images with distortion can been seen in Table 1 for VGG16, in Table 2 for ResNet and in Table 3 for Inception.
As can be seen, with added noise the performance of VGG16 is by far the highest, followed by ResNet and Xception. This is however not the same for all distortions. When inverting the colours of all images, ResNet has the highest accuracy, followed by Xception and VGG16. So in order to achieve the best accuracy it would be wise to base one's choice of CNN on the type of distortions that can occur. The problem is, one usually does not know this. Based on the three networks we have researched here, we can conclude that ResNet is overall the best when images have distortions of noise or inverted colours. However, all networks score very poorly when adding noise. There are however large differences in what images have the lowest accuracy, as can be seen in Figure 19. This shows the accuracy for images 0 to 100 of the CIFAR-100 dataset. The image of the lion, which we have taken as an example image, has a score that is below the average accuracy.

Distortion	**Accuracy**
Noise low	0.00192807
Noise medium	0.00192072
Noise high	0.00191925
Inverted	0.03647352

Table 1: Distortions accuracy for VGG16, see appendix A in Figures 25, 26, 27 and 36 for reference.

Distortion	**Accuracy**
Noise low	0.00138379
Noise medium	0.00137698
Noise high	0.00136104
Inverted	0.05806074

Table 2: Distortions accuracy for ResNet, see appendix A in Figures 28, 29, 30 and 37 for reference.

Distortion	**Accuracy**
Noise low	0.00131603
Noise medium	0.00128962
Noise high	0.00126936
Inverted	0.04803827

Table 3: Distortions accuracy for Xception, see appendix A in Figures 31, 32, 33 and 38 for reference.

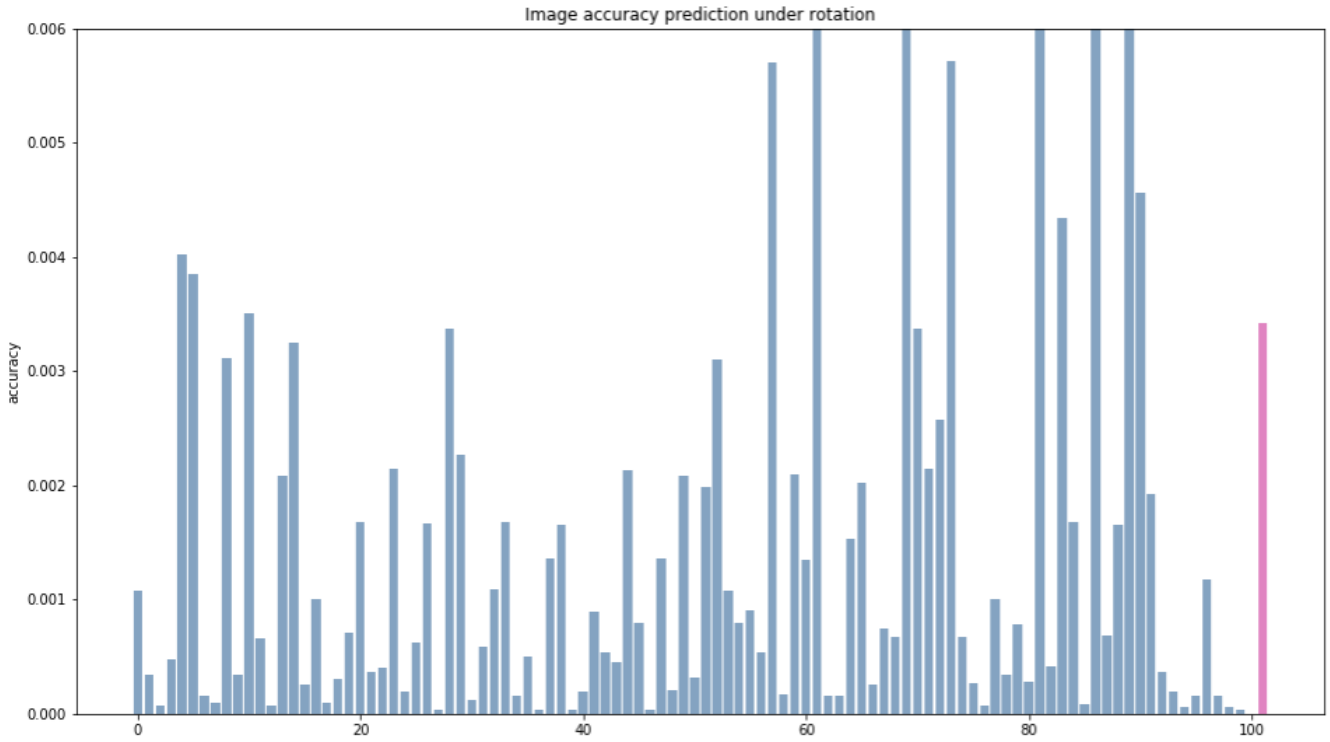


Figure 19: Noise distortion accuracy per image, average in pink on the right.

6 Contributions

In Table 4 we can see the overview of all the CNNs and all the manipulations we have used. Our main findings are:

1. VGG outperforms ResNet and Xception for the average accuracy of *rotated* images.
2. VGG and ResNet outperform Xception for images with added *noise*.
3. ResNet outperforms Xception for images with *inverted colours*.

CNNs	**Rotation Average accuracy**	**Low noise Accuracy**	**Med noise Accuracy**	**High noise Accuracy**	**Inverted Accuracy**
VGG	*0.29073771*	*0.00192807*	*0.00192072*	*0.00191925*	0.03647352
ResNet	0.20473173	0.0013864014	0.00137698	0.00136104	*0.05806074*
Xception	0.25751170	0.00131603	0.00128962	0.00126936	0.04803827

Table 4: All accuracies for all three CNNs and all manipulations.

7 Conclusion

We have compared the accuracy of three state-of-the-art CNNs to each other with different manipulations of the input images; we used rotation, added noise and inverted the colours. Our focus was on three different types of Deep Convolutional Neural Networks: the simple VGG16, the skip connections ResNet-50 and the Inception based (and most complicated of the three) Xception. We have turned all images of the CIFAR-100 dataset for a complete rotation of 360 degrees in steps of 10 degrees. The results for all three CNNs were quite similar at first glance, they all have peaks at 0, 90, 180, 270 and 360 degrees. However, on closer inspection VGG and Xception are most similar, their difference in accuracy has not been higher than 0.08 during the full rotation. However, ResNet and Xception are most similar on the highest peaks (90, 180, 270 degrees). This is due to the fact that VGG scores the highest accuracy on these quarter rotations, leaving the other two close together with their lower scores.
When using distortions there is also not one CNN simply the best. VGG performs best when noise is added, but has by far the lowest accuracy when it comes to images with inverted colours. In general, images with added noise perform dreadful in recognition tasks using CNNs. Inverting colours of images resulted in higher accuracy than expected.

8 Future research ideas

There has been quite some research on how to achieve a better performance when images are (randomly) rotated, as mentioned in the related work section. However, finding solutions for the failure to recognise distorted images might be an interesting next step. For rotation, first trying to find the orientation of an image could make CNNs even more robust. Training on rotated or otherwise manipulated data seems to be a decent base to build on.
When comparing images with distortions, it might be interesting to look into the type or class of images which have a higher or lower accuracy and if there is a common aspect to this. As seen in the appendix, Figures 34 and 35, there are many differences in accuracy for noise and inverted colours. This could also be researched for rotation: what images are almost always recognised? For this, however, one might be able to depend on common sense. Does background matter in recognising images? Is this the same for all CNNs?
It might also be interesting to compare rotation and distortion and whether there are certain areas of overlap or images that are always recognised better.

References

[Cho] F. Cholet. Keras library. https://keras.io/.

[Cho17] F. Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1800–1807.

[CW16] T. Cohen and M. Welling. Group equivariant convolutional networks, 2016. International Conference on Machine Learning (ICML), pages 2990-2999.

[DMM18] B. Dumont, S. Maggio, and P. Montalvo. Robustness of rotation-equivariant networks to adverserial perturbation. *ArXiv*, abs/1802.06627, May 2018.

[DV18] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv*, abs/1603.07285, January 2018.

[ETT+18] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Mádry. A rotation and a translation suffice: Fooling CNNs with simple transformations. *ArXiv*, abs/1712.02779, February 2018.

[HMPK] N. Hirschkind, S. Mollick, J. Pari, and J. Khim. Convolutional neural network. Retrieved July 22, 2019, from https://brilliant.org/wiki/convolutional-neural-network/.

[HZRS15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *ArXiv*, abs/1512.03385, December 2015.

[Jia] Y. Jia. Caffe. https://caffe.berkeleyvision.org/.

[Kri] A. Krizhevsky. Learning multiple layers of features from tiny images. retrieved July 22, 2019 from https://www.cs.toronto.edu/~kriz/cifar.html.

[SI] Scikit-Image. Scikit-image rotation function. https://scikit-image.org/docs/dev/api/skimage.transform.html. Last accessed on 2019-21-07.

[SLJ+14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *ArXiv*, abs/1409.4842, September 2014.

[SZ15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ArXiv*, abs/1409.1556, April 2015.

[Tea] Google Brain Team. Tensorflow. https://www.tensorflow.org/.

A Additional Figures

This appendix contains additional Figures that can be interesting as a reference.

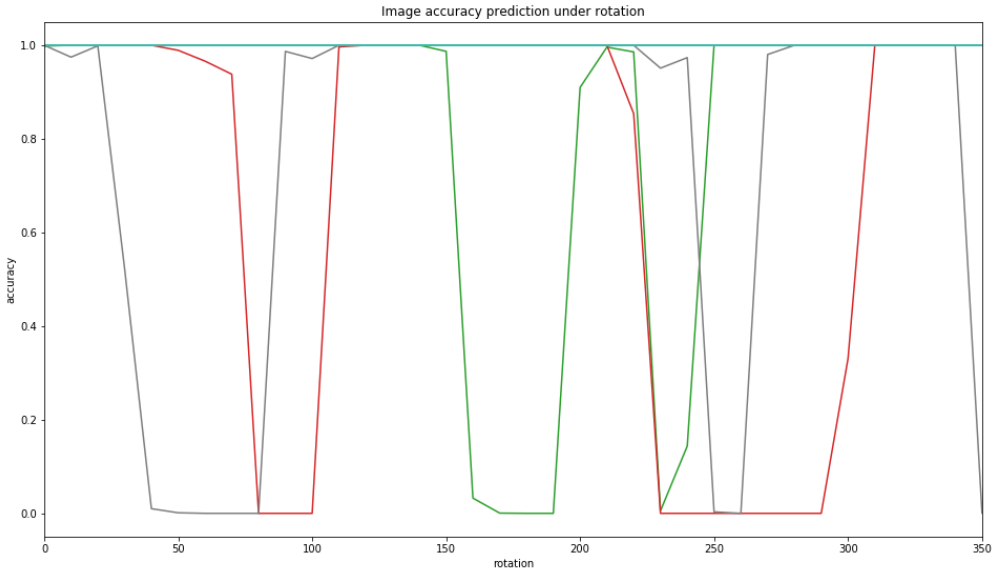


Figure 20: Inception results of images 3000–3010 plot together.

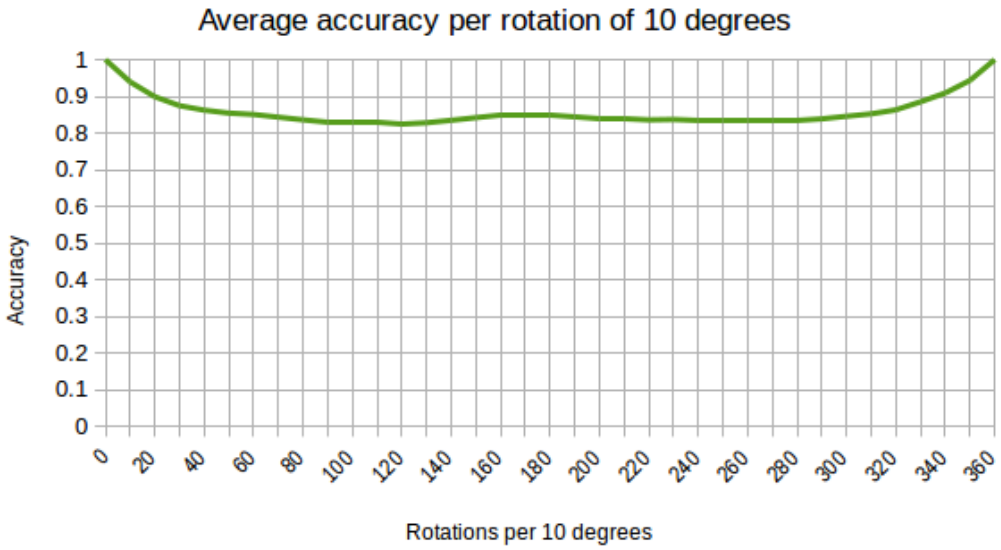


Figure 21: Inception results overall average.

actual_label_con	model	rotation	true_label
1	inception_v3	0	web_site
1	inception_v3	10	web_site
1	inception_v3	20	web_site
1	inception_v3	30	web_site
1	inception_v3	40	web_site
1	inception_v3	50	web_site
1	inception_v3	60	web_site
1	inception_v3	70	web_site
1	inception_v3	80	web_site
1	inception_v3	90	web_site

Figure 22: Inception results all classified as web_site.

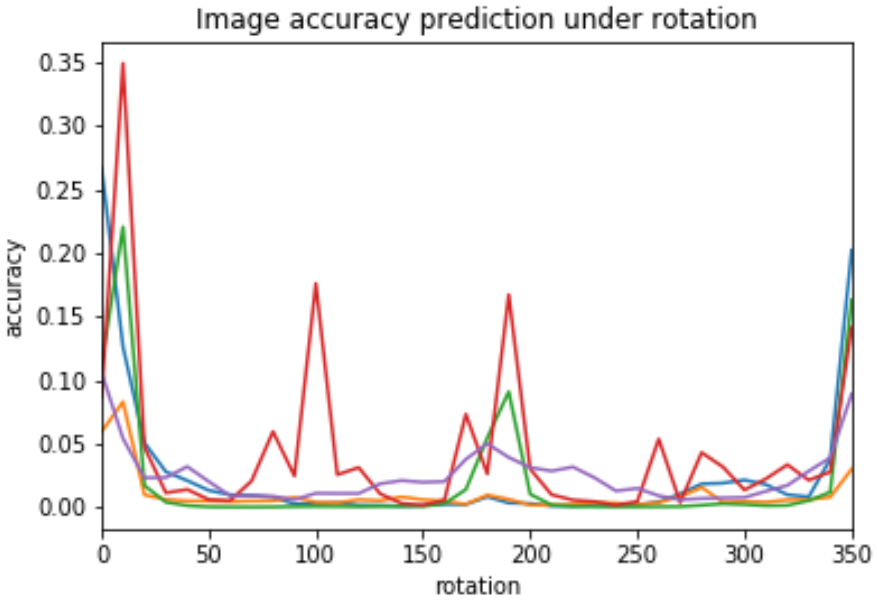


Figure 23: VGG16 results images 0–5 plot together.

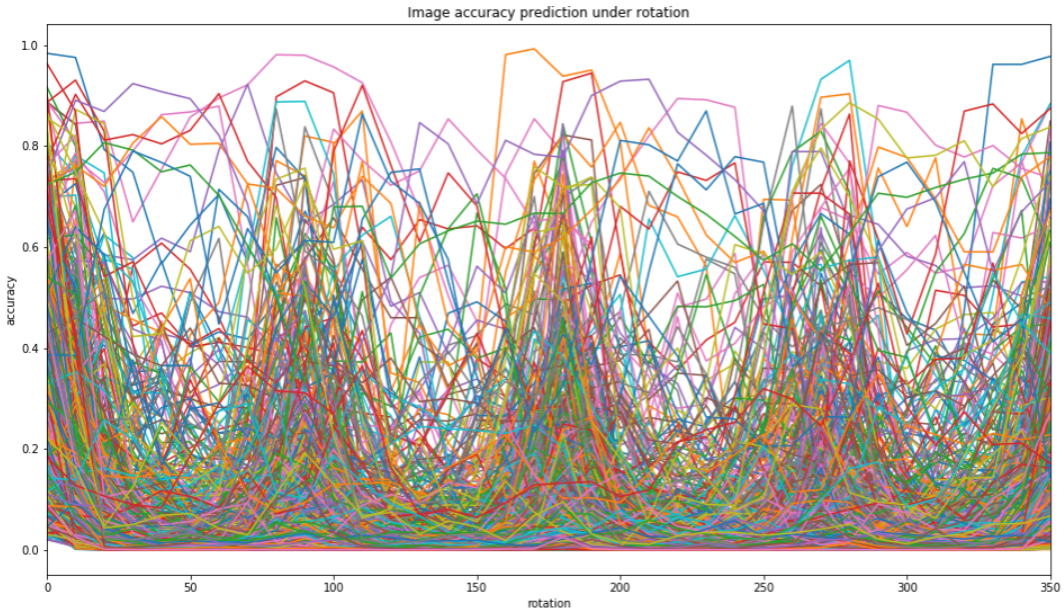


Figure 24: VGG16 results images 2001–3000 plot together.

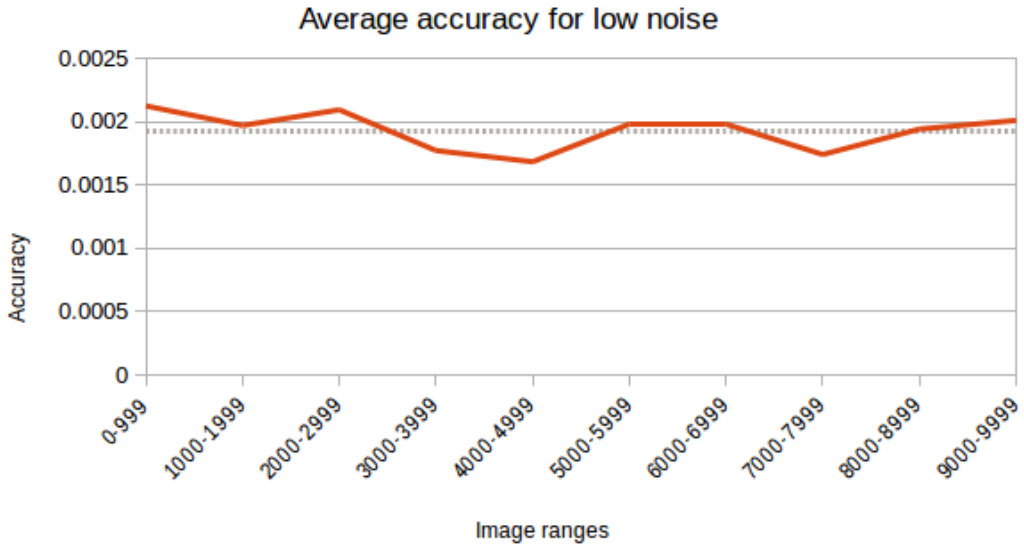


Figure 25: VGG average for low noise (0.055 sigma) per image range.

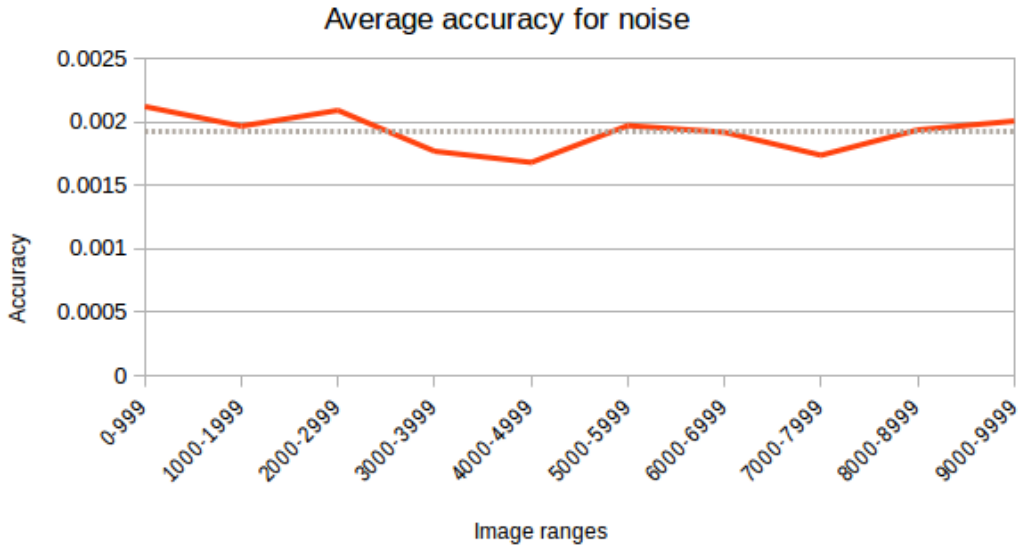


Figure 26: VGG16 average for medium noise (0.155 sigma) per image range.

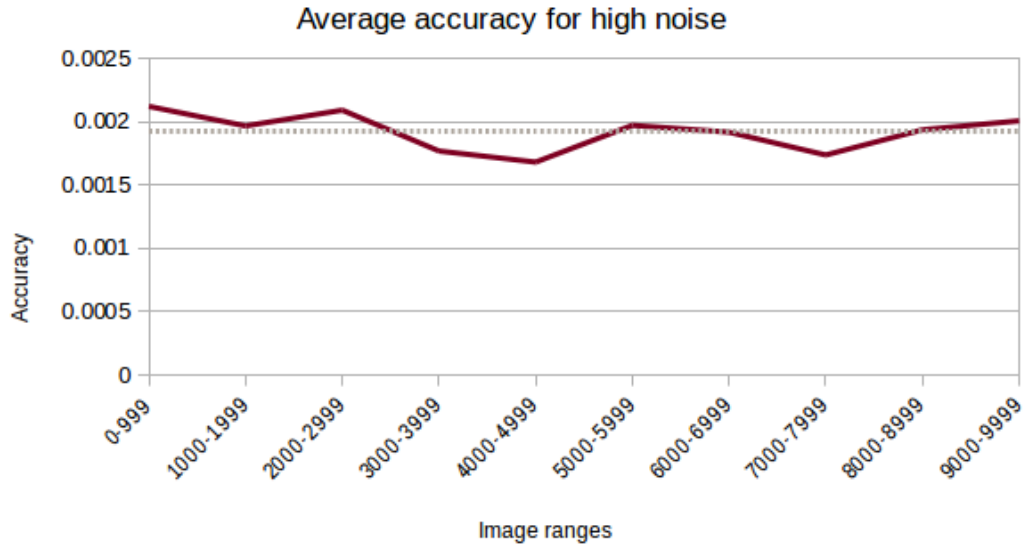


Figure 27: VGG average for high noise (0.255 sigma) per image range.

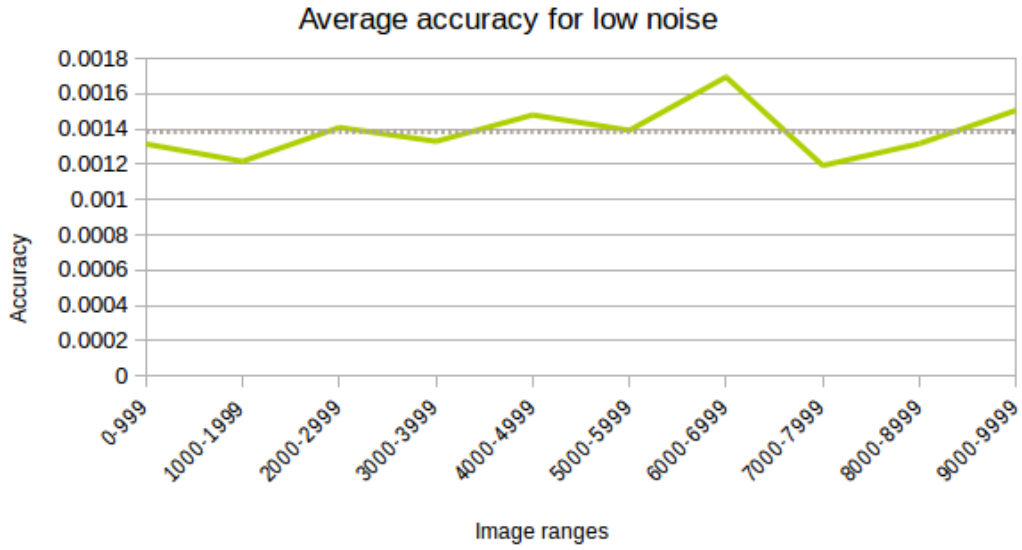


Figure 28: ResNet average for low noise (0.055 sigma) per image range.

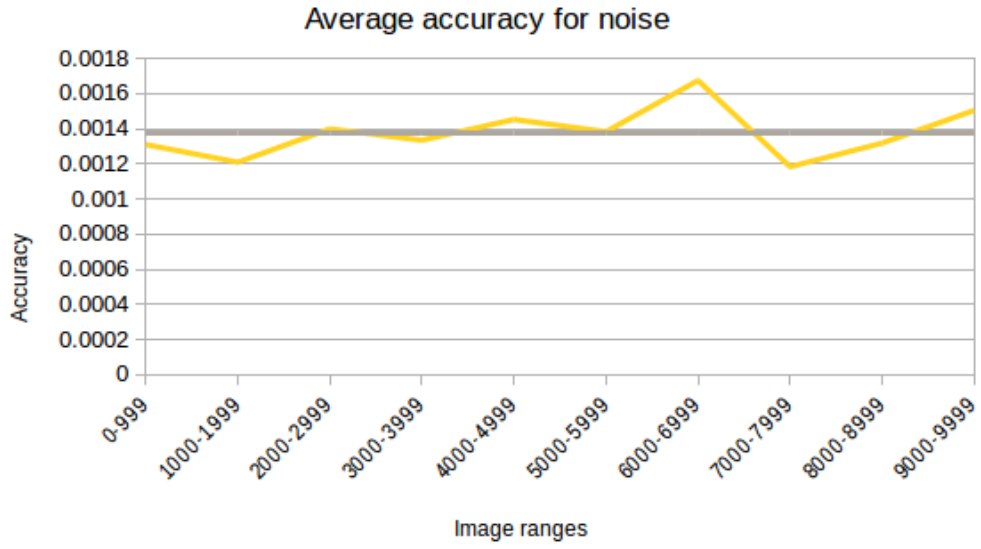


Figure 29: ResNet average for medium noise (0.155 sigma) per image range.

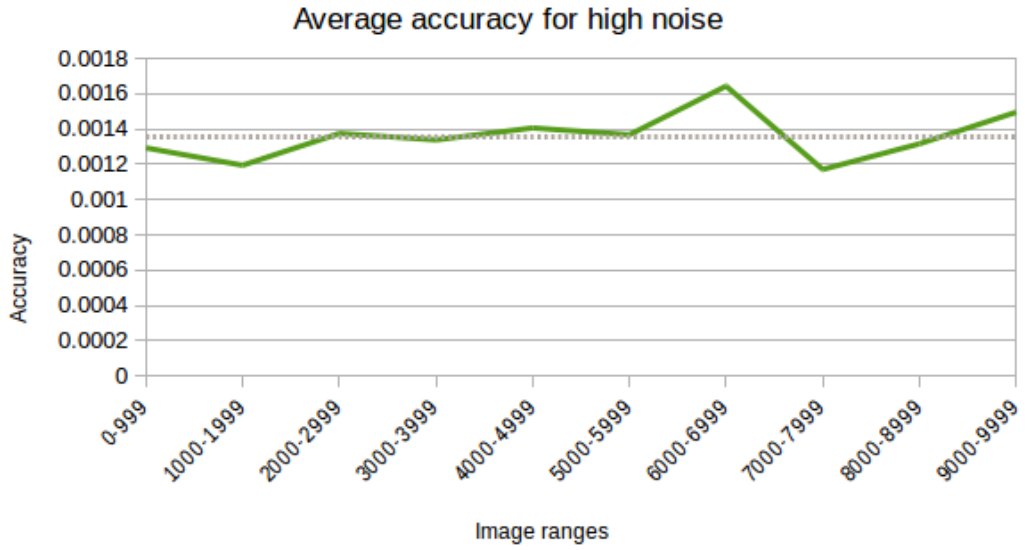


Figure 30: ResNet average for high noise (0.255 sigma) per image range.

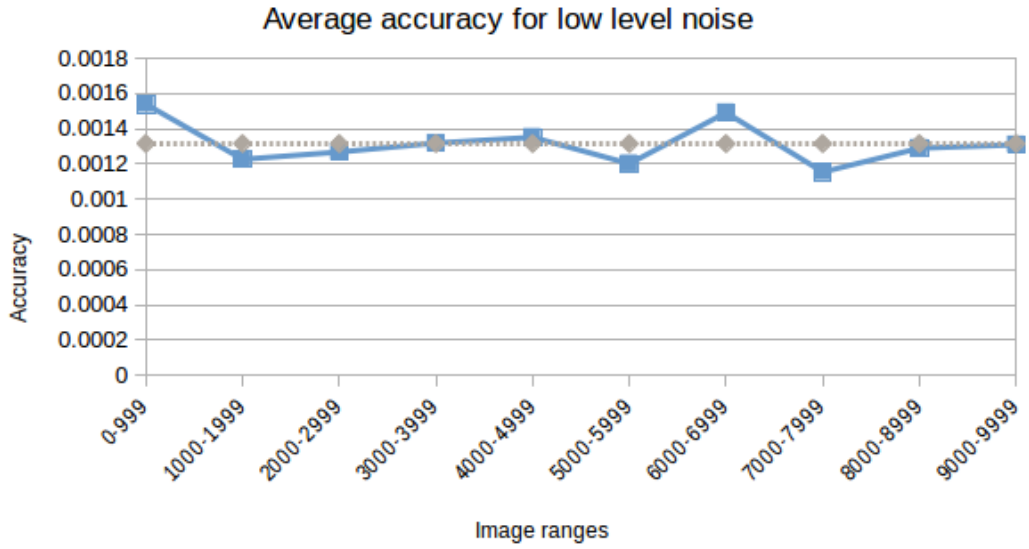


Figure 31: Xception average for low noise (0.055 sigma) per image range.

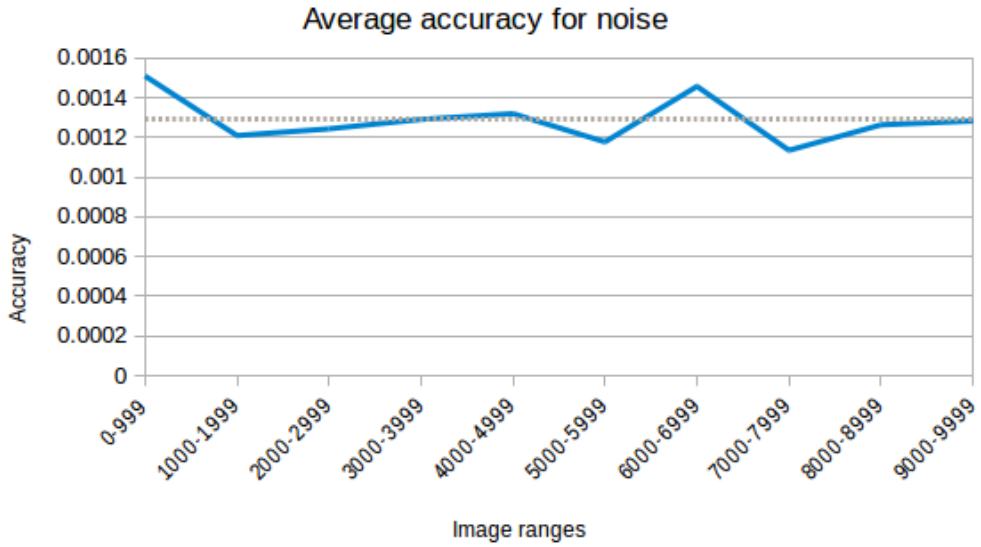


Figure 32: Xception average for medium noise (0.155 sigma) per image range.

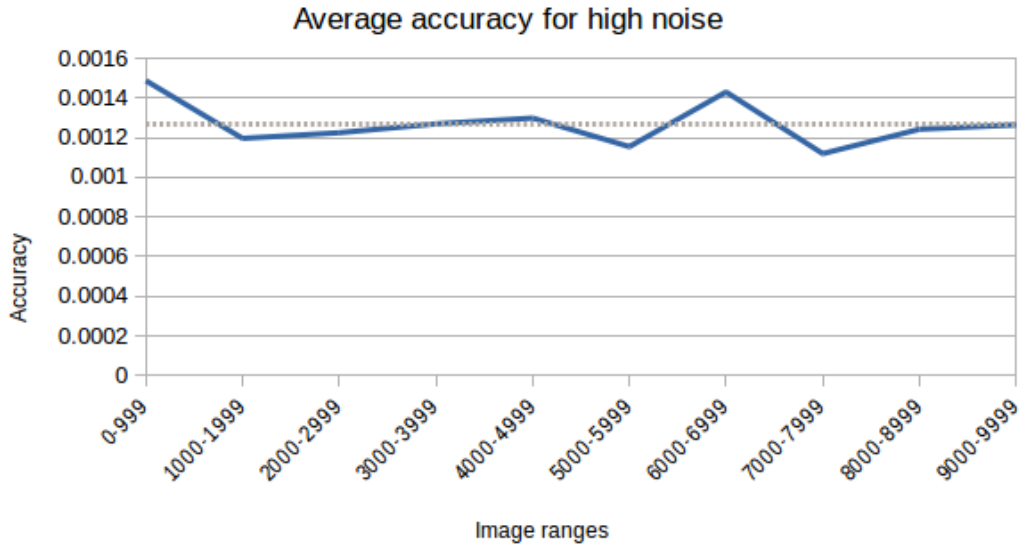


Figure 33: Xception average for high noise (0.255 sigma) per image range.

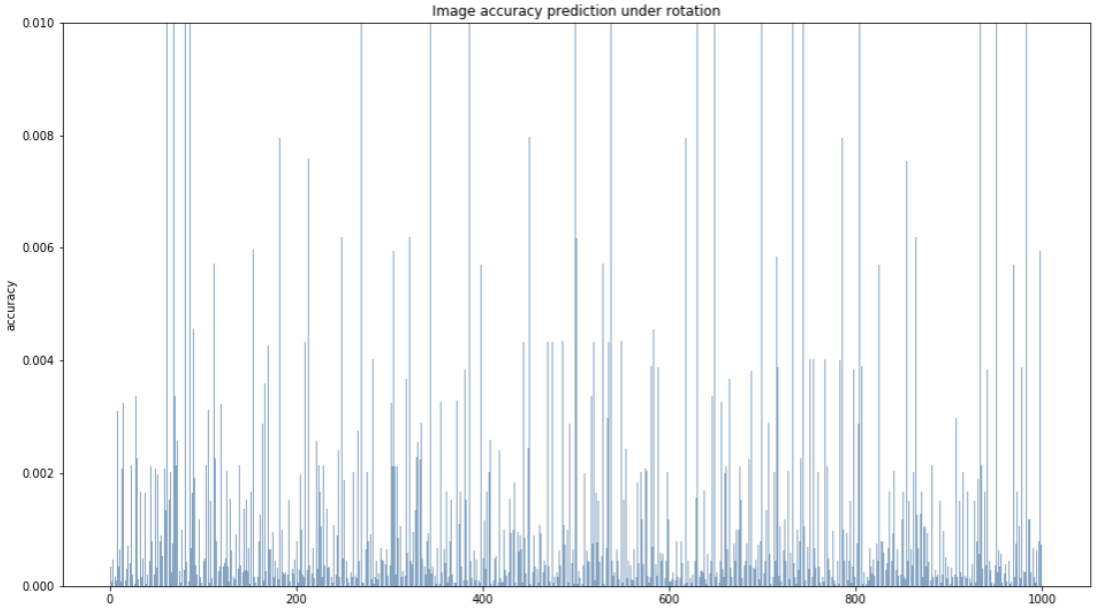


Figure 34: VGG16 results medium noise (0.155 sigma) per image for images 0–1000.

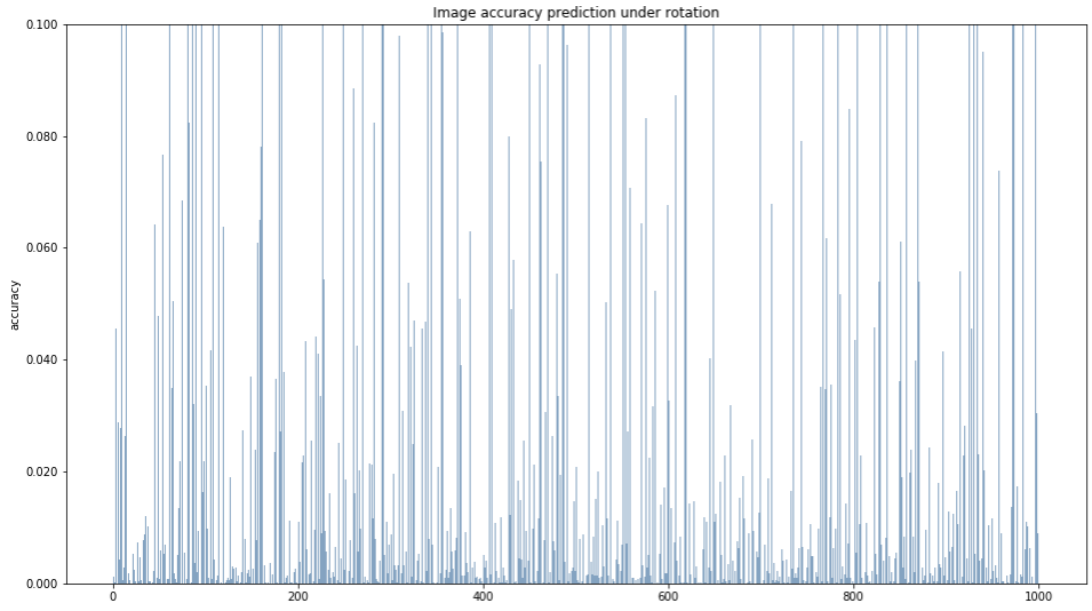


Figure 35: VGG16 results inverted colours per image for images 0–1000.

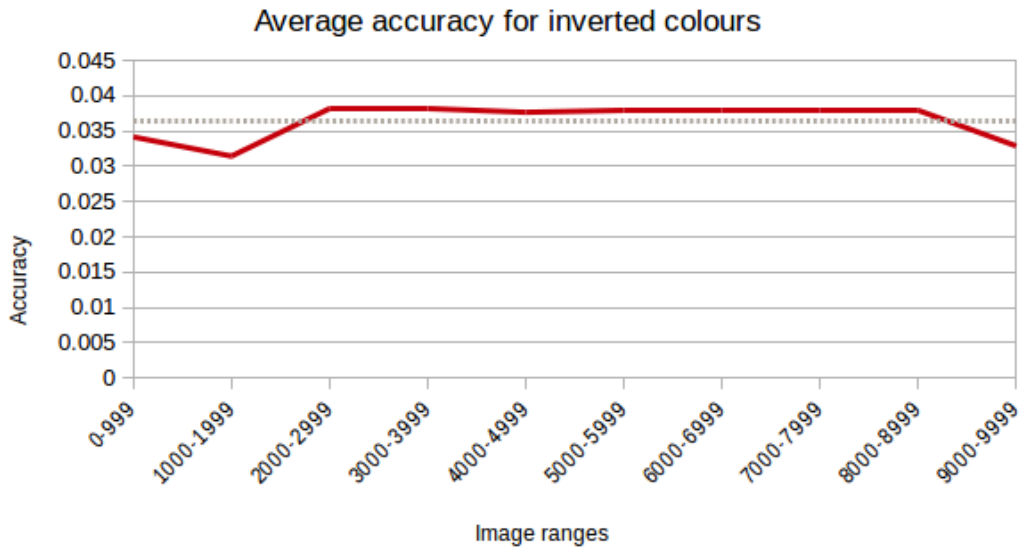


Figure 36: VGG16 average for inverted colours per image range.

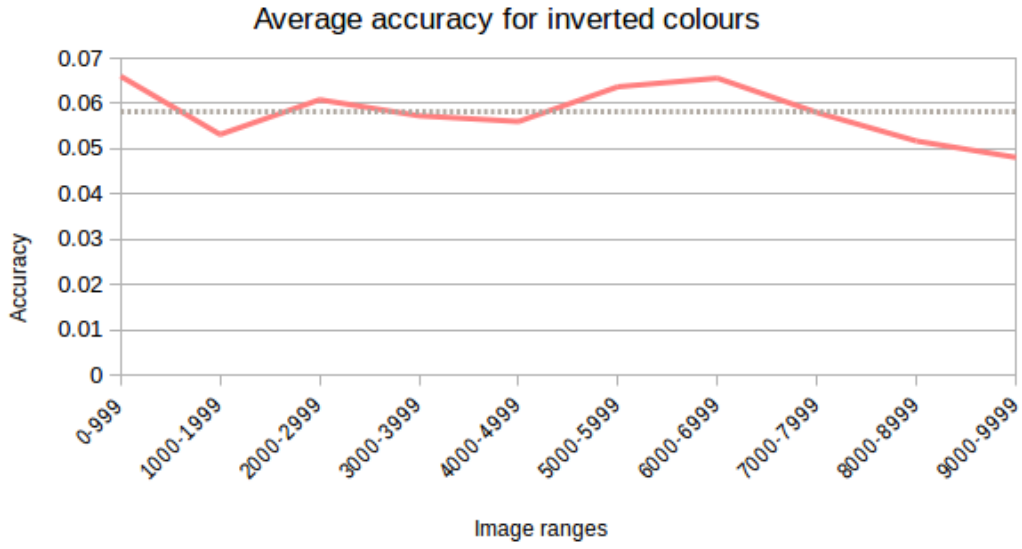


Figure 37: ResNet average for inverted colours per image range.

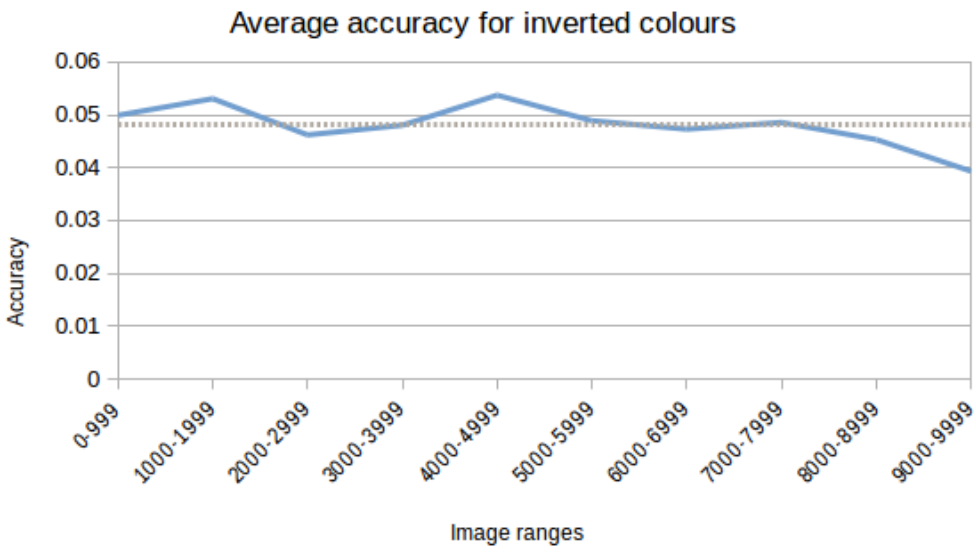


Figure 38: Xception average for inverted colours per image range.