Incorporating Orthogonal Moments in CNNs

André Le Blanc
Abstract

Incorporating Orthogonal Moments in CNNs

André Le Blanc

Convolutional neural networks (CNNs) can accurately classify objects in images using convolutional layers that extract features from images. Features from images can also be extracted using image moments, such as Gabor and Zernike moments. The aim with this project was to evaluate the impact on the accuracy of a CNN when the initial layer of the CNN is substituted with a layer of Zernike or Gabor filters. Three CNNs were trained on the dataset Dogs vs cats; a CNN with one hidden layer, GaborNet and Alexnet. Additionally, GaborNet was trained on the KidneyECCV dataset. After each epoch, the accuracy of the CNN was measured using a validation set. A Zernike layer increased the accuracy of the CNN with one hidden layer by 7.84% after the first epoch and by 2.83% after 50 epochs when training on the Dogs vs Cats dataset. When training GaborNet and AlexNet on Dogs vs Cats, an initial Gabor layer improved accuracy, while a Zernike layer decreased accuracy. An initial Zernike layer yielded the highest initial accuracy for GaborNet on the KidneyECCV; and an initial Zernike or Gabor layer yielded higher accuracy than the reference value but not as high accuracy as the GaborNet with a Gabor layer after 50 epochs.

In conclusion, substituting a CNN's initial layer with an image moment layer can improve the accuracy of the network, with the effect, however, varying between datasets and networks.
Contents

1 Abstract 2

2 Introduction 6

3 Related work 7

4 Background 8
  4.1 Convolutional Neural Networks 8
  4.2 Gabor filter banks 10
  4.3 Zernike Moments 11

5 Materials and Method 14
  5.1 PyTorch 14
  5.2 Datasets 15
    5.2.1 Dogs vs. Cats 15
    5.2.2 KidneyECCV 15
  5.3 Amazon Web Service 16
  5.4 Docker 17
  5.5 Neural Networks 17
    5.5.1 One Hidden Convolutional Layer 17
    5.5.2 GaborNet 18
    5.5.3 Cnn3C and Zern3C 19
5.5.4 AlexNet ........................................................................................................... 20

6 Results .................................................................................................................. 21

   6.1 One Hidden Convolutional Layer, Dogs vs Cats ................................. 21
   6.2 GaborNet, Dogs vs Cats, 5x5 Kernel Size ............................................. 22
   6.3 GaborNet, Dogs vs Cats, 11x11 Kernel Size ......................................... 24
   6.4 GaborNet, KidneyECCV ........................................................................... 25
   6.5 AlexNet, Dogs vs Cats ............................................................................. 26
   6.6 AlexNet, KidneyECCV ............................................................................. 27

7 Discussion ............................................................................................................. 29

8 Future Work .......................................................................................................... 30

9 References ............................................................................................................ 31
2 Introduction

Texture can provide predominant features for feature extraction, image segmentation, and classification. By analyzing texture, relevant patterns and features can be identified in images. Orthogonal moments are a family of polynomials that have strong descriptive power. The human visual system uses methods similar to orthogonal moments to identify patterns and classify objects. They have a strong and long history in image analysis and processing for texture recognition. However, orthogonal moments have fallen out of favour to the benefit of convolutional neural networks (CNNs)[11]. These consist of multiple layers of trainable filters and have proven superior for many learning-based image analysis tasks.

Orthogonal moments can be expressed as filters by applying the orthogonal moment algorithm to subsets of the image and have been shown to have strong texture recognition capabilities even under noisy conditions[1].

The aim of this thesis is to investigate whether a synthesis between CNNs and orthogonal moments, by substituting one of the convolutional layers in a CNN with an orthogonal moment filter layer, improves accuracy.
3 Related work

The inspiration for this thesis came from three papers.

The first paper was "GaborNet: Gabor filters with learnable parameters in deep convolutional neural networks" by Andrey Alekseev and Anatoly Bobe[1]. In their paper, they describe and evaluate GaborNet, comparing it to a CNN with an initial Conv2d-layer instead of a Gabor layer. Their paper showed that using an initial layer that implements a Gabor layer can increase the performance of a neural network.

The second paper was, "Towards automated multiscale imaging and analysis in TEM: Glomerulus detection by fusion of CNN and LBP maps" by Elisabeth Wetzer and Ida-Maria Sintorn[16]. In this paper, the performance of the VGG16 and ResNet50 neural networks on the KidneyECCV dataset was evaluated and compared to its performance on a modified version of the KidneyECCV dataset in which the images had been preprocessed using local binary patterns. The results showed that using image descriptors to preprocess images can improve the performance of neural networks and help classify images.

In the third paper, "Regional Zernike moments for texture recognition"[14] by Gustaf Kylberg and Ida-Maria Sintorn, Zernike moments were applied to regions in images, creating a measure with which texture could be more accurately analyzed. The paper concluded that Zernike moments can provide noise-insensitive and accurate measures for texture recognition.
4 Background

4.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are machine learning algorithms commonly used for image processing. They have proven to be highly efficient at image processing while requiring little to no preprocessing of the image. Like all neural networks, CNNs can be trained over time, meaning that the developer does not need to specify all the values contained in the network. This makes it possible for the same network to be reused by retraining the network for a new task. The ability of CNNs to be trained allows for relatively easy development, high efficacy and achievement of complex tasks.

Segmenting an image is a task that CNNs are commonly used for. When segmenting an image, a CNN divides an image into segments, depending on the contents of those segments. Segmenting an image of a dog could entail dividing the image into a segment containing the dog and a segment of the image that doesn’t include a dog. Classifying an image is another task that CNNs are commonly used for. When classifying an image, a CNN labels an image, thereby deciding whether a member of a class is present in the image or a region of the image. Segmentation answers the question of where in the image a class is present, while classification answers the question whether a class is present in an image or in a region of an image.

A CNN consisting of an input layer, one or more hidden layers, and an output layer, is a feed-forward neural network; hence there are no loops in the network and data is never passed backwards.

An image is either fed to the CNN in the form of a tensor[13], a datatype that can be represented as a multidimensional array or as an image. A defining property of CNNs - and the property that is its namesake - are its convolutional layers, of which a CNN must have at least one. The convolutional layers pass a convolutional filter over their input as seen in Fig. 1. Filters cover several pixels at once, which can reduce the amount of information passed on to the next layer by being effective at extracting the useful information. Filters are many-to-one mappings. This means that we can extract the useful information in a neighbourhood of pixels and condense it to one
pixel. The filter in a convolutional layer uses shared weights; in other words, the filter will be the same for every part of the image. Training the layer is the process of finding the best weights for that filter. With different weights, different features are detected, and training can be seen as the process of finding what features to extract from the image.

Figure 1: A visualization of a convolutional layer passing a 3x3 filter over the input.

Convolutional layers are commonly succeeded by a pooling layer in order to reduce the amount of data. The pooling layer downsamples the output of the convolutional layer by replacing several pixels in the image produced by the convolutional layer with a single pixel. Pooling layers apply a function - such as an averaging function - that yields the average value of the pixels in a tile of pixels, or alternatively, a max function that yields the highest value of the pixels on a tile of pixels. Since the pooling layers reduce the amount of information, they usually have small tile sizes such as 2x2.

Fully connected layers are layers in which each neuron is connected with each neuron in the previous network, hence the name fully connected. Fully connected layers are commonly placed towards the end of neural networks and are used to compile information from different parts of the network in order to provide the neural network with its final output. Since each neuron in the fully connected layer is connected with each neuron in the previous layer, the fully connected layer is computationally heavy.

The output of each neuron in the network is passed to an activation function in order to make the network nonlinear. Without activation functions, the neural network would just be a set of matrixes multiplied by each other and, therefore, could be replaced by a single matrix representing the product.
of the layers. Because of the nonlinearity provided by activation functions, it is possible to perform backpropagation on networks with more than one layer. If there was linearity between layers, the filters of the different layers could be multiplied together, forming one layer. A commonly used activation function is the ReLU function, which sums its inputs and then uses the sum to calculate $f(x) = \max(0, x)$.

In Fig. 2 we see the different layers integrate into a CNN, which in this case can classify images of digits. The image first passes through two consecutive convolutional and max pooling layer pairs, where the convolutional layer extracts features which are then downsampled in the max-pooling layer. Finally a fully connected layer is applied to classify which digit is represented in the image.

4.2 Gabor filter banks

Gabor filters are linear filters that determine whether or not a certain frequency is present in the image. Since Gabor filters are not rotation invariant, they may have to be applied in several different orientations in order to approximate all rotations. Such a set of Gabor filters is referred to as a Gabor filterbank, and they are suitable for texture analysis.

Mathematically, Gabor filters are defined by the following function:
\[ g(x, y; \lambda, \theta, \phi, \gamma) = \exp \left( -\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2} \right) \cos \left( 2\pi \frac{x'}{\lambda} + \phi \right) \]

In the function, \( x \) and \( y \) are the coordinate values representing the location of the pixel in the filter. \( \lambda \) is the period or wavelength of the sinusoidal factor. The rotation of the filter is represented by \( \theta \) in regard to the y-axis. If \( \theta \) is 0° or 180°, the filter will be vertical; whereas if \( \theta \) is 90°, the filter will be horizontal. The phase offset is denoted by \( \phi \), and shifting \( \phi \) shifts the phase of the wave. \( \gamma \) represents the aspect ratio or the ellipticity of the wave; a large value will give a circular wave while a low value will give a line-shaped wave.

The effects of applying a Gabor filter are visualized in Fig. 3b. The Gabor filter shown in Fig. 3c was applied to the image of a zebra shown in Fig. 3a, resulting in Fig. 3b.

\[ \text{Fig. 3a An image of a zebra[6]} \quad \text{Fig. 3b Gabor filter applied} \quad \text{Fig. 3c The Gabor filter} \]

### 4.3 Zernike Moments

Zernike moments[15] are disk shaped orthogonal moments that can be used to describe the pixels within the area covered by the disk. Image moments find these quantities by calculating a weighted average of the pixels in the disk. With the use of orthogonal functions, we can ensure that no redundant information is collected since each function encodes different information. Functions are orthogonal if their inner product is 0, meaning that each variable encodes information that is completely separate from the information encoded by other variables. One of the most common examples of orthogonal
functions is sine and cosine since the dot product of sine(x) and cosine(x)
will be 0 for all x. A primary advantage of Zernike moments is rotation in-
variance; since the Zernike moment’s magnitude is rotation invariant, it only
needs to be passed over the image once in order to find features, regardless
of the feature’s rotation. In order to define the Zernike moments we start by
defining moments on unit disks as \( v_{pq} \):

\[
v_{pq} = \int_{0}^{1} \int_{0}^{2\pi} R_{pq}(r)e^{-iq\phi} \ast f(r, \phi) r dr d\phi
\]

The moment kernel consists of two parts: \( R_{pq}(r) \) as the radial function
and \( e^{-iq\phi} \) as the angular function. \( f(r, \phi) \) represents the image. For Zernike
moments the radial function is defined as:

\[
R_{pq}(r) = \sum_{s=0}^{\frac{p+|q|}{2}} \frac{(-1)^s \ast (p-s)!}{s!(\frac{p+|q|}{2} - s)!((\frac{p-|q|}{2} - s))!} \cdot r^{p-2s}
\]

The order is given by \( p \) and the repetition is given by \( q \).
By applying Zernike moments on an image, we can reconstruct the image. Applying Zernike moments with a radius of 8 pixels twenty times on Fig. 4a, an image of an on/off button yielded twenty floating point numbers. These numbers were used to reconstruct the image resulting in Fig. 4b.
5 Materials and Method

In this section, we will define the tools and datasets and their use in this project.

5.1 PyTorch

The PyTorch[12] machine learning library version 1.8.0 was used for doing tensor operations such as training neural networks. While PyTorch is primarily developed by Facebook, it is available under a modified BSD license[5] since 2016. PyTorch was chosen for its easy-to-use interface, extensive documentation and online community. Since a Gabor layer has already been implemented in PyTorch[2], PyTorch was selected instead of TensorFlow. The main advantage of PyTorch over NumPy is its GPU-acceleration, which allows the training of a CNN to be done on a GPU. Most of the workload while training a CNN is matrix multiplication, which GPUs can parallelise to a much higher degree than a CPU. Since PyTorch’s GPU-acceleration is implemented using CUDA, the CNN needs to be trained on CUDA compliant hardware in order to use GPU-acceleration.
5.2 Datasets

5.2.1 Dogs vs. Cats

Kaggle’s dataset Dogs vs Cats[7] was used to train and evaluate the neural networks. Originally compiled for a machine learning competition, the dataset has gained wide popularity and is a commonly used dataset. The dataset contains 12,500 labeled color images of dogs and a corresponding number of images of cats. The dataset was split into a training set consisting of 10,000 cat images and 10,000 dog images as well as a validation set consisting of 2,500 cat images and 2,500 dog images. In Fig. 5a we see an example of a dog image and in Fig. 5b we see an example of a cat image.

5.2.2 KidneyECCV

KidneyECCV[16] is a dataset consisting of 459 kidney images. These images were mirrored and rotated 90, 180 and 270 degrees, giving a total of 2,295 images. These images were divided into two subsets: a training set and a validation set. The training set contained 1,836 images. 504 of these images contained glomerulus specific structures, while the remaining 1,332 images did not contain such structures. The validation set contained 459 images. 126 of
these images contained glomerulus specific structures while 333 images did not.

Glomerulus are small blood vessels in the kidneys. These blood vessels form structures within the kidney. In Fig. 6a, we see an image containing a glomerulus structure, and in Fig. 6b we see an image without glomerulus. The large round structure in the center of Fig. 6b constitutes the glomerulus structure. The photos were taken during different sessions with a MiniTem 5 electron microscope.

Fig. 6a An image of a glomerulus structure. Fig. 6b An image without a glomerulus structure.

5.3 Amazon Web Service

The networks developed during the course of the project were trained on an Amazon web service g4dn.2xlarge instance [4]. G4 instances are optimized for machine learning and include a NVIDIA T4 GPU, enabling GPU-acceleration of tensor operations.
5.4 Docker

In order to facilitate easy deployment of the code and the data required to train the neural networks on the cloud, the code and the data were containerized in Docker containers\cite{docker}. Docker containers are lightweight virtual containers that can run on top of an operating system. Since dependencies and the environment an application needs to run can be included in the Docker container, running a Docker container on a cloud service only requires uploading the container and then running it on the cloud. The cloud instance does not need to be configured, and dependencies do not need to be installed in order to run the container.

5.5 Neural Networks

Several neural networks that classify images were used for this project. These networks were trained for 50 epochs, and the accuracy of the networks were validated after each epoch. The limit of 50 epochs was chosen since the improvement in accuracy of the networks diminished with increased training and, furthermore, training networks is computationally heavy.

5.5.1 One Hidden Convolutional Layer

Three neural networks with one hidden convolutional layer and two fully connected layers were implemented: CnnC, ZernC, and GaborC. The three networks were differentiated by their initial layers: CnnC has a convolutional layer initialized with default values, GaborC has a convolutional layer initialized with Gabor filters, and ZernC has a convolutional layer initialized with Zernike filters. The initial-layer and the hidden-layer are each followed by a 2x2 maxpooling layer. The architecture of the networks is visualized in Section 5.5.1. The adam optimization algorithm was used with a learning rate of 0.001. The learning rate is the size of the step in the adam optimization algorithm. A higher learning rate means that the optimizer makes a larger change to the filter, while a lower learning rate means that the optimizer makes a smaller change to the filter.
The networks were trained with a batch size of 156 when training on Dogs vs Cats and a batch size of 16 when training on KidneyECCV. The size of the initial layer’s kernel is passed as a parameter, and the second layer has a 3x3 kernel size. Passing the size of the initial layer’s kernel as a parameter allows for the size of the initial layer’s kernel to easily be modified. The network has as many input channels as there are channels in the image that the network is trained on. Thus, if the neural network is trained on a set of greyscale images, such as KidneyECCV, the network would have one input channel. Since Dogs vs Cats has three input channels, the network would have three input channels.

5.5.2 GaborNet

GaborNet[2] is a neural network developed by Andrey Alekseev at the Moscow Institute of Physics and Technology. It utilizes a Gabor filter bank as its initial layer. The network was written in python using the PyTorch library. GaborNet has an initial Gabor layer followed by three hidden convolutional layers as visualized in Fig. 8. The kernel size of the Gabor layer is passed as a parameter during the initialization of the network, allowing for networks
with different Gabor filter sizes. However, the hidden layers have a fixed size of 3x3. Each Gabor and convolutional layer is followed by a maximum pooling layer, which chooses the maximum value of four pixels, thereby condensing the output of the preceding layer by a factor of four. The two final layers of the network are fully connected layers that condense the output of the convolutional layers into a classification of dogs and cats or alternatively glomerulus and non glomerulus tissue. The Gabor filter’s weights are calculated using a real valued implementation of the Gabor function. These weights are trained by PyTorch’s conv2d function. GaborNet was trained with a batch size of 156 for Dogs vs Cats and 16 for KidneyECCV. The learning rate was 0.001 for training on Dogs vs Cats and 0.0002 for training on KidneyECCV.

The code for the Gabor layer was taken from GaborNet’s GitHub repository. No code for the rest of GaborNet was found, so an implementation of it was created using a description of GaborNet[1].

Figure 8: The architecture of GaborNet with three convolutional layers. Image is modified from Alekseev and Bobe[1].

5.5.3 Cnn3C and Zern3C

Two corresponding versions of GaborNet were implemented to compare with GaborNet: Cnn3C, and Zern3C. The architectures of Cnn3C and Zern3C are the same as the architecture of GaborNet except for their initial layers, which have been replaced by an additional conv2d-layer, alternatively a Zernike layer. As with GaborNet, the initial layer has a filter size that is set with a parameter during initialization of the network.
5.5.4 AlexNet

AlexNet[8] is a CNN developed in 2012, and was at the time, one of the best CNNs for classifying objects in images. It was one of the first CNNs to use ReLU as its activation function instead of Tanh, which up until then had been the most common activation function. The architecture of AlexNet, which is depicted in Fig. 9, consists of five convolutional layers followed by three fully connected layers. After each layer, the ReLU activation function is applied. AlexNet was modified to classify two categories instead of 1000 categories, as AlexNet was originally designed to do. Two versions of AlexNet were developed for this thesis: one with the first layer exchanged for a Gabor layer and one with the first layer exchanged for a Zernike layer. AlexNet was trained on Dogs vs Cats with a batch size of 156 and a learning rate of 0.0001.

![Figure 9: The architecture of AlexNet. Adapted from Popular Networks[10]](image-url)
6 Results

6.1 One Hidden Convolutional Layer, Dogs vs Cats

![Figure 10: The accuracy of CnnC, GaborC and ZernC while training on Dogs vs Cats.](image)

<table>
<thead>
<tr>
<th>Epochs</th>
<th>CnnC</th>
<th>GaborC</th>
<th>ZernC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>54.59</td>
<td>60.80</td>
<td>62.43</td>
</tr>
<tr>
<td>10</td>
<td>66.39</td>
<td>73.27</td>
<td>71.52</td>
</tr>
<tr>
<td>20</td>
<td>71.31</td>
<td>75.66</td>
<td>72.53</td>
</tr>
<tr>
<td>30</td>
<td>73.46</td>
<td>75.37</td>
<td>76.22</td>
</tr>
<tr>
<td>40</td>
<td>76.02</td>
<td>79.29</td>
<td>75.73</td>
</tr>
<tr>
<td>50</td>
<td>75.90</td>
<td>78.68</td>
<td>78.77</td>
</tr>
</tbody>
</table>

Table 1: The accuracy of CnnC, GaborC and ZernC during training on Dogs vs Cats.

With one hidden convolutional-layer, the networks that used Gabor and Zernike layers provided a clear increase in the performance of the network, as
shown in Fig. 10. The Zernike layer was 7.84% more accurate after the first epoch compared to CnnC, as shown in Table 1. The difference between the networks narrowed with further training, but even after 50 epochs, ZernC and GaborC outperformed CnnC. The initial layer had a kernel size of 5x5. The accuracy was measured using the validation dataset after each epoch. The images in the validation dataset were not included in the training dataset. The data was collected from training the networks once.

6.2 GaborNet, Dogs vs Cats, 5x5 Kernel Size

![Graph showing accuracy of CnnC, GaborC and ZernC](image)

Figure 11: The accuracy of CnnC, GaborC and ZernC while training on Dogs vs Cats.
Table 2: The accuracy of Cnn3C, Gabor3C and Zern3C during training on Dogs vs Cats.

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Cnn3C</th>
<th>Gabor3C</th>
<th>Zern3C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53.32</td>
<td>52.78</td>
<td>50.65</td>
</tr>
<tr>
<td>10</td>
<td>75.08</td>
<td>73.51</td>
<td>55.42</td>
</tr>
<tr>
<td>20</td>
<td>82.50</td>
<td>81.42</td>
<td>60.74</td>
</tr>
<tr>
<td>30</td>
<td>85.46</td>
<td>84.51</td>
<td>68.20</td>
</tr>
<tr>
<td>40</td>
<td>87.15</td>
<td>87.76</td>
<td>67.76</td>
</tr>
<tr>
<td>50</td>
<td>88.29</td>
<td>89.07</td>
<td>70.77</td>
</tr>
</tbody>
</table>

An initial Gabor layer provides a minor improvement in accuracy for Gabor-Net when the initial kernel size is 5x5, as shown in Fig. 11.

Zern3C, which has a Zernike layer, has lower performance than the other networks as seen in Table 2.
6.3 GaborNet, Dogs vs Cats, 11x11 Kernel Size

Figure 12: The accuracy of Cnn3C, Gabor3C and Zern3C while training on Dogs vs Cats.

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Cnn3C</th>
<th>Gabor3C</th>
<th>Zern3C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.16</td>
<td>53.77</td>
<td>54.02</td>
</tr>
<tr>
<td>10</td>
<td>65.18</td>
<td>77.38</td>
<td>54.02</td>
</tr>
<tr>
<td>20</td>
<td>72.18</td>
<td>83.02</td>
<td>59.59</td>
</tr>
<tr>
<td>30</td>
<td>80.05</td>
<td>87.25</td>
<td>53.84</td>
</tr>
<tr>
<td>40</td>
<td>83.11</td>
<td>87.89</td>
<td>64.75</td>
</tr>
<tr>
<td>50</td>
<td>85.33</td>
<td>89.05</td>
<td>65.11</td>
</tr>
</tbody>
</table>

Table 3: The accuracy of Cnn3C, Gabor3C and Zern3C during training on Dogs vs Cats.

Using an 11x11 initial kernel size, the performance of GaborNet is lowered for all three variants, as depicted in Fig. 12. However, the variant with an initial Gabor layer has its best performance relative to the other networks with this kernel size. This can be seen in Table 3, which contains the accuracy
of the networks during training when the networks were trained once. The underperformance of Zern3C is amplified by increasing the kernel size to 11x11.

6.4 GaborNet, KidneyECCV

![Figure 13: The accuracy of Cnn3C, Gabor3C and Zern3C while training on KidneyECCV.](image)

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Cnn3C</th>
<th>Gabor3C</th>
<th>Zern3C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>83.42</td>
<td>74.46</td>
<td>85.33</td>
</tr>
<tr>
<td>10</td>
<td>83.83</td>
<td>81.39</td>
<td>87.09</td>
</tr>
<tr>
<td>20</td>
<td>84.51</td>
<td>82.61</td>
<td>84.78</td>
</tr>
<tr>
<td>30</td>
<td>84.92</td>
<td>88.45</td>
<td>87.77</td>
</tr>
<tr>
<td>40</td>
<td>87.64</td>
<td>90.08</td>
<td>86.68</td>
</tr>
<tr>
<td>50</td>
<td>88.18</td>
<td>89.67</td>
<td>88.86</td>
</tr>
</tbody>
</table>

Table 4: The accuracy of Cnn3C, Gabor3C and Zern3C during training on KidneyECCV.
GaborNet achieves high initial accuracy on KidneyECCV and improves steadily with training, as shown in Fig. 13. The initial Zernike layer does, however, provide the highest initial accuracy, and it maintains its lead for nearly all epochs until epoch 26. In Table 4, Gabor3C attained the highest accuracy after epoch 30. The values are an average of the values generated by training the networks twice.

6.5 AlexNet, Dogs vs Cats

![Figure 14: The accuracy of AlexNet during training on Dogs vs Cats.](image)

<table>
<thead>
<tr>
<th>Epochs</th>
<th>AlexNet</th>
<th>AlexNet + Gabor</th>
<th>AlexNet + Zernike</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62.10</td>
<td>62.45</td>
<td>63.702</td>
</tr>
<tr>
<td>10</td>
<td>85.76</td>
<td>86.16</td>
<td>80.59</td>
</tr>
<tr>
<td>20</td>
<td>86.47</td>
<td>87.69</td>
<td>80.03</td>
</tr>
<tr>
<td>30</td>
<td>88.53</td>
<td>87.96</td>
<td>79.13</td>
</tr>
<tr>
<td>40</td>
<td>88.46</td>
<td>88.74</td>
<td>80.29</td>
</tr>
<tr>
<td>50</td>
<td>87.73</td>
<td>89.98</td>
<td>81.27</td>
</tr>
</tbody>
</table>

Table 5: The accuracy of AlexNet during training on Dogs vs Cats.
AlexNet offers high performance on the Dogs vs Cats dataset, as depicted in Fig. 14. With a Gabor layer as its initial layer, AlexNet achieves an accuracy of 90%, which is 1-2% more accurate than the standard implementation of AlexNet. The Zernike layer offers high initial performance. However, in Table 5 we see that already after three epochs, standard AlexNet has achieved parity with AlexNet with an initial Zernike layer, and thereafter, surpasses the accuracy of AlexNet with an initial Zernike layer. The data was collected from training the networks once.

6.6 AlexNet, KidneyECCV

![Graph showing accuracy of AlexNet during training on KidneyECCV.](image)

Figure 15: The accuracy of AlexNet during training on KidneyECCV.
<table>
<thead>
<tr>
<th>Epochs</th>
<th>AlexNet</th>
<th>AlexNet + Gabor</th>
<th>AlexNet + Zernike</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>77.71</td>
<td>81.65</td>
<td>71.20</td>
</tr>
<tr>
<td>10</td>
<td>83.29</td>
<td>83.15</td>
<td>82.88</td>
</tr>
<tr>
<td>20</td>
<td>83.42</td>
<td>83.15</td>
<td>86.55</td>
</tr>
<tr>
<td>30</td>
<td>86.82</td>
<td>83.15</td>
<td>88.18</td>
</tr>
<tr>
<td>40</td>
<td>86.14</td>
<td>81.93</td>
<td>87.36</td>
</tr>
<tr>
<td>50</td>
<td>88.59</td>
<td>83.70</td>
<td>82.34</td>
</tr>
</tbody>
</table>

Table 6: The accuracy of AlexNet during training on KidneyECCV.

AlexNet was trained twice on the KidneyECCV dataset, and the results were combined and visualized in Fig. 15. AlexNet quickly reached an accuracy above 80% with all versions of AlexNet. AlexNet and AlexNet with an initial Zernike layer improved slowly with training, while AlexNet with a Gabor Layer flatlined for many epochs. In Table 6, we see that AlexNet with a Zernike layer has the lowest accuracy at epoch 50; however, it had an accuracy of 87.5% in epoch 48, which was the highest of the three networks in the epoch.
7 Discussion

The aim of this project was to evaluate the impact of substituting the first convolutional layer in a CNN with a layer that implements a Zernike or Gabor layer. Zernike moments and Gabor filters are capable image descriptors that can extract information from images without having to be trained. The success of these layers varied depending on circumstances. In the smallest neural network, with one hidden convolutional layer, the Zernike and Gabor layers achieved similar results, both outperforming the regular CNN.

However, with larger networks such as GaborNet, the Zernike layer lowered the accuracy of the network on Dogs vs Cats while retaining high performance on KidneyECCV. The larger the kernel and size, the slower the network trained on Dogs vs Cats. Larger networks with larger kernel sizes were hampered by the use of a Zernike layer on Dogs vs Cats. The Gabor layer did not observe this reduction in performance. While training on KidneyECCV, an initial Zernike layer yielded the highest initial performance for GaborNet, suggesting that Zernike moments are capable image-descriptors for the dataset. However, GaborNet with an initial Zernike layer trained slower than the other versions of GaborNet.

The initial layer of ZernC had 96 output channels, while the initial layers of Zern3C and AlexNet had respectively 32 and 64 output channels. Each output channel had different kernel weights due to a different order, repetition and rotation of the Zernike function. It may be possible to improve the performance of both Zern3C and AlexNet with an initial Zernike layer by altering the number of output channels and the order, repetition and rotation of the Zernike moments.

With small networks and with fewer epochs of training, Zernike and Gabor layers can give an improvement in accuracy.
8 Future Work

There are a multitude of ways to expand on this thesis. Using Zernike filters as the initial layer on a wider number of networks and training those networks on a larger set of datasets would improve the understanding of the performance of Zernike layers. Gabor layers worked better on Dogs vs Cats than they did on the KidneyECCV-dataset[1]. Exploring which datasets Zernike layers perform better or worse on would be an interesting expansion of the project.

Zernike layers are computationally heavier than Conv2d-layers. Benchmarking the performance of Zernike, Gabor and Conv2d-layers in order to see whether it is more efficient to use Zernike or Gabor layers than to train a conv2d-layer longer would be another possible expansion of this thesis.

The implementation of the Zernike layer can be modified, and there are many ways the Zernike layer can be implemented. The Zernike moment is round, while PyTorch uses square-shaped image kernels. In this project’s implementation of the Zernike filter, the Zernike filter was made to fit entirely inside the square-kernel. However, the code can easily be modified so that the entire square-shaped kernel is filled by values generated by the Zernike function. Zernike moments produce complex-valued filters. In this project, only the real part of the complex number was used. A further extension of this project would be to experiment with imaginary and complex-valued Zernike layers.
9 References


