ROAD CRACK DETECTION USING MODERN DEEP LEARNING METHODS

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Department of Computer Science and Engineering

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Diploma Thesis

Katsaliros Angelos

Supervisor : Christophoros Nikou

Co-Supervisor : George Sfikas

March 14, 2021

Abstract

Road crack detection has vital importance in driving safety. However, it is very challenging because of the complexity of the background, cracks are easily confused with foreign objects, shadows, background textures and are also inhomogeneous. Many methods have been proposed for this task, but Convolutional Neural Networks(CNN) are promising for crack classification and segmentation with high accuracy and precision. Another method that is being studied is the use of quaternions. Quaternions have the advantage of providing more structural information of the color and as a result offer better learning results and avoid overfitting. In this study we focused on implementing various cnn networks and compare their results with Quaternion Convolutional Neural Networks (QCNN) which are an extension of the cnn in the quaternion domain. Specifically we replaced the convolutional and linear layers of the cnn's with quaternion convolutional layers and linear layers respectively.

Keywords: image classification, image segmentation, convolutional neural network, deep learning, quaternionic neural network, road crack detection

Extended Abstract in Greek

Η ανίχνευση ρωγμών σε δρόμους αποτελεί ένα πολύ σημαντικό παράγοντα τόσο στην ασφάλεια των οδηγών όσο και στην εξασφάλιση της ποιότητας των αγαθών και των υπηρεσιών που προσφέρονται μέσω οδικών δικτύων. Ωστόσο αποτελεί ένα πολύ απαιτητικό πρόβλημα λόγω της πολυπλοκότητας του φόντου της εικόνας, των ξένων αντικειμένων, των σκιών και της ανομοιογένειας των ρωγμών. Τα τελευταία χρόνια έχει επιτευχθεί σημαντική πρόοδος λόγω της χρήσης βαθιών συνελικτικών νευρωνικών δικτύων και μεθόδων αναγνώρισης προτύπων. Αυτά τα νευρωνικά δίκτυα έχουν πετύχει πολύ καλύτερα αποτελέσματα σε σχέση με χαρακτηριστικά που φτιάχτηκαν από ανθρώπους. Τα βαθιά συνελικτικά νευρωνικά δίκτυα αποτελούνται από πολλά επίπεδα συνελικτικών νευρώνων. Τα χαρακτηριστικά που είναι πιο ποιοτικά για την ταξινόμηση και κατάτμηση της εικόνας βρίσκονται στα πιο βαθιά επίπεδα. Υπάρχουν όμως και τεχνικές οι οποίες δεν έχουν μελετηθεί σε μεγάλο βαθμό και μπορούν να προσφέρουν πολύ καλά αποτελέσματα. Μια τέτοια τεχνική είναι η χρήση τετράδων Hamilton(quaternions) που μπορούν να αντικαταστήσουν τους συνελικτικούς και γραμμικούς νευρώνες παράγοντας ίδια ή και καλύτερα αποτελέσματα, μειώνοντας παράλληλα και τον αριθμό των παραμέτρων στο δίκτυο, μειώνοντας έτσι το υπολογιστικό κόστος.
Σκοπός της εργασίας είναι η υλοποίηση βαθιών συνελικτικών νευρωνικών δικτύων τόσο για την ταξινόμηση όσο και για την κατάτμηση εικόνων με ρωγμές και η σύγκριση της απόδοσης τους αντικαθιστώντας τα συνελικτικά και γραμμικά επίπεδα με αντίστοιχα επίπεδα που χρησιμοποιούν τετράδες Hamilton.

Λέξεις κλειδιά : τετράδες Hamilton, ταξινόμηση , κατάτμηση , βαθιά συνελικτικά νευρωνικά δίκτυα

Acknowledgements

Firstly, I would like to thank **George Sfikas** for his support and guidance he offered me during the project. I would also like to express my gratitude to my family and friends for their invaluable support.

Contents

Chapter 1

Introduction

1.1 Overview

Road crack detection which includes classification and segmentation is an important task for driving safety and also ensuring the quality of transport services.
Computer vision and machine learning have provided many techniques for crack detection with very good results. Nowadays, most methods are taking advantage of the effectiveness of deep features in deep neural networks. Specifically, deep convolutional neural networks (CNN) are the most popular for supervised classification and feature learning. In our study, deep features are learned directly from raw images using convolutional neural networks. However, there has been done little research in other methods such as using quaternions. Quaternions are a form of non real numbers with one real part and three independent imaginary parts. Quaternionic neural networks can replace linear and convolutional layers and provide similar or better results with less parameters than the non-quaternionic.
Goal of this study is to compare the results of image classification and segmentation of convolutional neural networks architectures with their respective quaternionic neural networks and that is done by replacing convolutional and linear layers with their quaternionic ones.

1.2 Methods

Computer vision is a field of computer science that enables computers to identify and process objects in images and videos. Thanks to advances in deep learning and neural networks the field has been able to surpass humans in some tasks such as detective and labeling objects. In this study we will implement convolutional neural networks for image classification such as AlexNet ,Vgg16 and a custom CNN, then replace their convolutional and linear layers with quaternionic layers and compare their accuracy, precision and number of parameters used. Then we will use an already implemented deepcrack segmentation model in order to compare it with its quaternionic version.

1.3 Organization

This report is organized as follows. Chapter 2 introduces neural networks and analyses the methods that have been used in the experiments in this project for the part of image classification. Chapter 3 introduces the concept of quaternions and quaternionic neural networks. Chapter 4 introduces us to the concept of image segmentation. Chapter 5 analyses the experiments conducted in the project. Chapter 6 summarises with our conclusions.

Chapter 2

Neural Networks

2.1 Theoretical background

Neural networks is the use of a large number of elementary nonlinear computing elements called neurons which are organized in the form of networks and are inspired by biological systems. Neurons receives a signal, it process it and transmits it to other neurons. The interest in neural networks goes back in the early 40s when McCulloch and Pitts [9] proposed neuron models in the form of binary threshold devices and stochastic algorithms involving sudden 0-1 and 1-0 changes of states in neurons as the bases for modeling neural systems. In the 1950s the field of machine learning came to the spotlight from the work of Rosenblatt [13]. They developed mathematical proofs for machines called perceptrons that when trained with linearly separable training sets, they would converge to a solution in a finite number of iterative steps. However as the perceptron initially seemed promising, it could not be used to recognise many classes and as a result could be used for most pattern recognition tasks. Perceptron was impossible to learn XOR function. This caused the field of neural network research to stagnate for many years, before considering using multiple layers of perceptrons. Multilayer perceptron, also known as feedforward neural network had greater processing power than perceptrons with one layer. Rumelhart, Hinton and Williams in their work [14] provide a method, called the generalized delta rule for learning by backpropagation, which provided an effective training method for multilayer machines. The generalized delta rule has been used successfully in numerous problems of practical interest.

This success has established multilayer perceptron-like machines as one of the principal models of neural networks currently in use. Although the list of practical uses of neural networks is extensive, applications of this technology in image pattern classification have not become as popular as in other fields. The use of neural networks in image processing relies heavily on architectures known as convolutional neural networks. One of the first applications of convolutional neural networks goes back in 1989 by the work of Lecun et al [6] for handwritten zip code recognition, but it was not until 2012 that convolutional neural networks began to be widely used in image processing following the publication of results by ImageNet Challenge in which the convolutional neural network, called AlexNet [5] won. Today convolutional neural networks are the primary methods for dealing with complex processes.

2.2 Perceptron

Perceptron is a binary linear classifier and it helps to classify the given input data. The perceptron works on these simple steps. First of all the inputs $x_1, x_2, ...x_n$ are multiplied with their weights values $w_1, w_2...w_n$, then it calculates the sum of all multiplied values and apply the weighted sum to an appropriate activation function which compares it with a threshold value. If the weighted sum is higher than the threshold value the neuron return 1 otherwise it returns 0. The perceptron uses the Heaviside step function or unit step function, whose value is zero for negative arguments and one for positive arguments. The mathematical figure is shown in figure 2.1 :

Figure 2.1: Perceptron

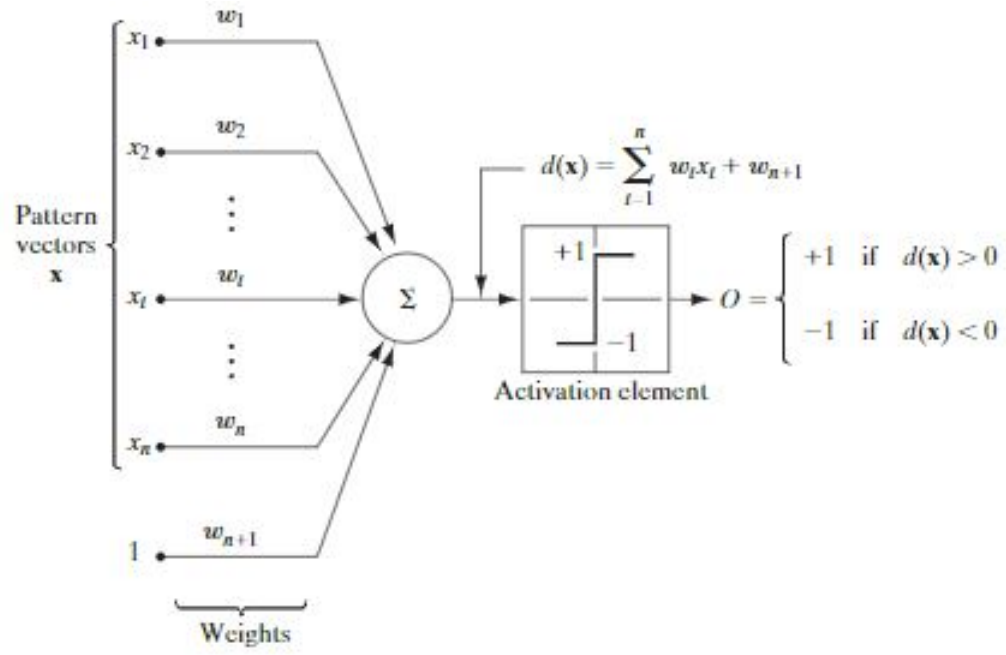


Figure 2.1:The appropriate weights are applied to the inputs, and the resulting weighted sum passed to a function that produces the output O.When $d(x) > 0$, the threshold element causes the output of the perceptron to be $+1$ meaning that x is recognised to belong to class c_1.When $d(x) < 0$ it belongs to c_2.

The training process for a single perceptron is done using the following steps:
1.Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value.
2.For each example j in our training set D, perform the following steps over the input x_j and desired output d_j:
2.1 Calculate the actual output:

$$y_j(t) = f[w(t) \cdot x_j] = f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + w2(t)x_{j,2} + ... + w_n(t)x_{j,n}]$$

2.2 Update the weights:

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t)x_{j,i})$$

, for all features $0 \leq i \leq n$ r is the learning rate.

2.3 MLP

Multilayer perceptrons are capable of approximating an XOR operator as well as many other non-linear functions. A multilayer perceptron (MLP) is a feedforward deep neural network. A feedforward neural network is an artificial neural network where the connections between the nodes do not form a cycle as shown in figure 2.2. They consist of an input layer, a number of hidden layers and an output layer. MLPs with one hidden layer are capable of approximating any continuous function. Training involves adjusting the parameter of the model in order to minimize error. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP uses backpropagation for training. Backpropagation makes adjustments to weights and bias in order to minimize the error which can be measured for example by root mean squared error. It can distinguish data that is not linearly separable.

Figure 2.2: Feed forward network

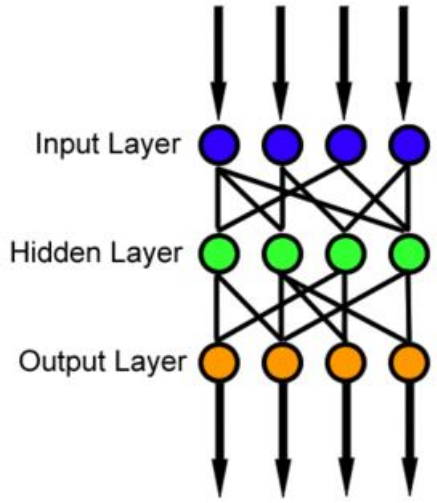


Figure 2.2: In the feed forward network the information flows to only one direction, from the input nodes through the hidden and to the output nodes and as a result it cannot create cycles.

MLP consists of backpropagation algorithm.Backpropagation has a forward and a backward pass. In the forward pass, the information goes from the input layer via the hidden layers to the output layer and then compares the result with the ground truth labels ,which shows us the real class of the input data. In the backward pass, we use partial derivatives to back-propagate the weights through the MLP. The gradient that is produced adjust the parameters in order to get the error minimum. This can be done with any gradient-based optimisation algorithm such as stochastic gradient descent. Training is done by changing weights of the connections between the neurons after each signal is processed, by calculating the error in the output compared to the actual result. The error of the output node can be shown as

$$e_j(n) = t_j(n) - y_j(n))$$

where t is the target value and y is the value produced by the perceptron-neuron and n is the n_{th} data point. The error that the backpropagation minimizes is

$$E(n) = \frac{1}{2}\sum_j e_j^2(n)$$

which is called Mean Square Error for each data point where j is the each neuron. Using gradient descent we can adjust weights to minimize error to the output:

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial u_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the learning rate.

The derivative can be further simplified in output nodes as :

$$-\frac{\partial E(n)}{\partial u_j(n)} = e_j(n)\varphi'(u_j(n))$$

, where φ' is the derivative of the activation function. The derivative of a hidden neuron is :

$$-\frac{\partial E(n)}{\partial u_j(n)} = \varphi'(u_j(n)) \sum_k -\frac{\partial E(n)}{\partial u_k(n)} w_{kj}(n)$$

In order to change the hidden layer weights, we use the the derivative of the activation function to change the output layer weights. The activation function MLP's use are sigmoids and most common are : hyperbolic tangent that ranges from -1 to 1 and is described by $y(u_i) = tanh(u_i)$, logistic function which ranges from 0 to 1 and is described by $y(u_i) = (1 + e^{u_i})^{-1}$ and rectified linear unit described by $f(x) = x^+ = max(0, x)$.

Figure 2.3: Logistic sigmoid function

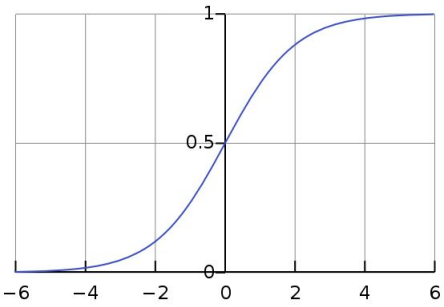


Figure 2.4: Hyperbolic Tangent function

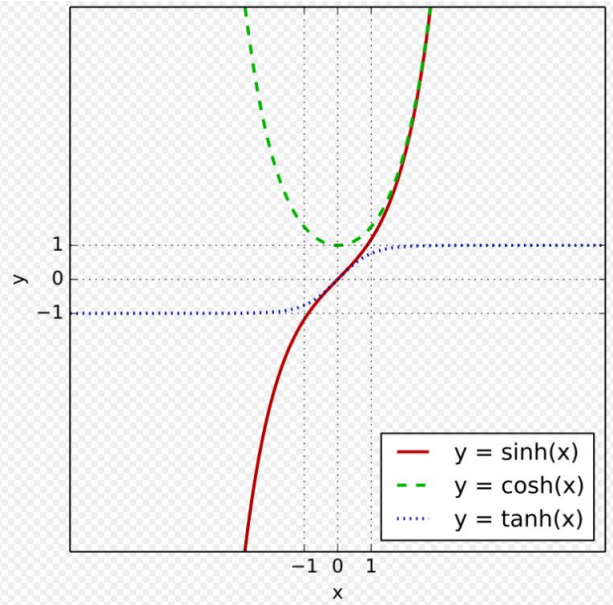
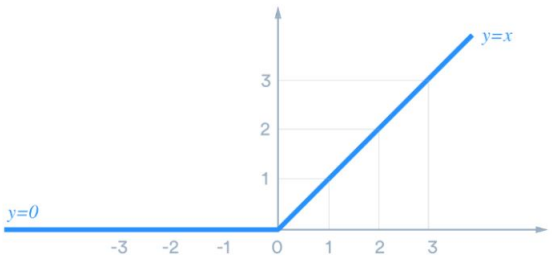


Figure 2.5: Rectified linear unit function



2.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep neural network which can take in an input image and learn characteristics of the image. The pre-processing required in a CNN is much lower as compared to other classification algorithms. They have applications in image and video recognition, image classification, medical image analysis, natural language processing, brain-computer interfaces and financial time series. CNNs are versions of multilayer perceptrons. As its name indicates CNN uses a mathematical operation called convolution. Convolutional networks use convolution in matrix multiplication in their layers. Convolution is a mathematical operation on two functions that produces a third function and is denoted by an asterisk:

$$s(t) = (x * w)(t)$$

In terms of convolutional neural networks, the two functions would be the input to a neuron and the kernel or filter of the convolution. The output is called featured map. In a 2 dimensional array we use a two dimension kernel K and the convolution is described as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

There are also a lot neural networks that implement the cross-correlation function that is same as convolution and is described as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Convolutional networks have spatial connectivity. Using a filter smaller than the input is intentional as it allows the same filter to be multiplied by the input array multiple times at different points on the input. That means that the filter can detect some small specific features of the input data with very small kernels of tens of pixels. As a result we need fewer parameters and memory and computational cost improves because it needs fewer operations. As we said the output from multiplying the filter with the input one time is a single value. Applying the filter multiple times in the input as show in figure 2.6 we create a 2 dimensional array called feature map. When the feature map is created, each value is passed to a nonlinear activation function, for example Relu described by

$$f(x) = x^+ = max(0, x)$$

Another idea that convolutional networks use is parameter sharing. Each member of the kernel is used at every position of the input, except sometimes the boundary pixels. Thus convolution is more efficient than dense matrix multiplication and that means that the network does not have to learn new parameters for every location, but only a single set of parameters. Parameter sharing causes the appearance of equivariance to traslation to the layer. Equivariance means that if the input of a function changes then the output changes as well with the same way.A mathematical approach of this is described as:

$$f(g(x)) = g(f(x))$$

A far as convolution is concerned if a function g is applied to the input data and then perform convolution it would be the same as performing the convolution first to the input data and then apply the function g. However there are some functions that convolution is not equivariant such as rotation. An architecture example of CNN is shown in figure 2.7. A convolutional neural network consists of an input layer, an output layer and multiple hidden layers. The hidden layer consists of series of convolutional layers. The convolutional layer produces a product that runs through a non-linear activation, Relu for example and then is followed by pooling, fully connected and normalization layers.

Figure 2.6: Feature Map

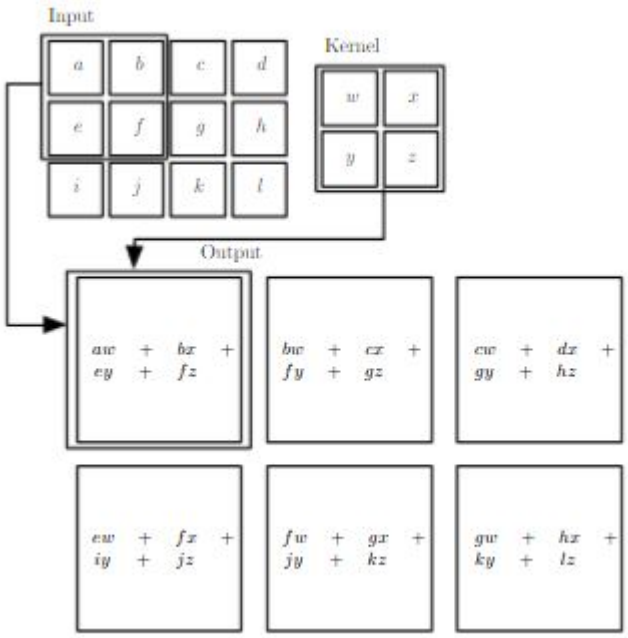


Figure 2.6: This picture is taken from [4]. On the left side is the input image and on the right side is the filter-kernel. The convolution is performed by sliding the kernel over the input.At every location we do element wise matrix multiplication and sum the result, which goes to the feature map

Figure 2.7: Convolutional Neural Network

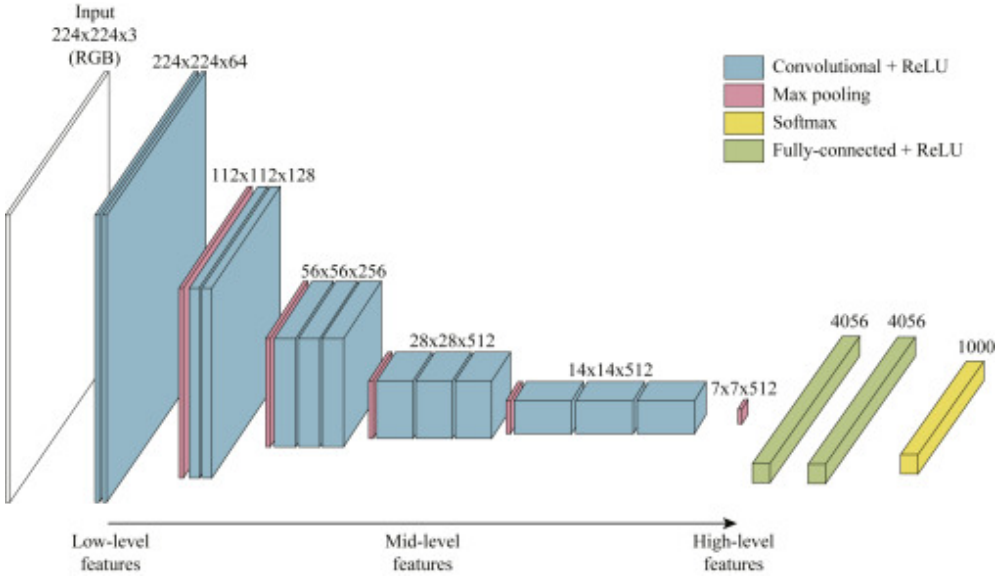


Figure 2.7: An example of a convolutional neural network [8].

2.5 Pooling

Pooling replaces the output of a certain location with a summary statistic of the nearby outputs. The max pooling operation returns the maximum output within a rectangular neighborhood. Other pooling operations includes for example the average of the rectangular neighborhood or the weighted average based on the distance of the central pixel. Pooling has a property which

Figure 2.8: Pooling

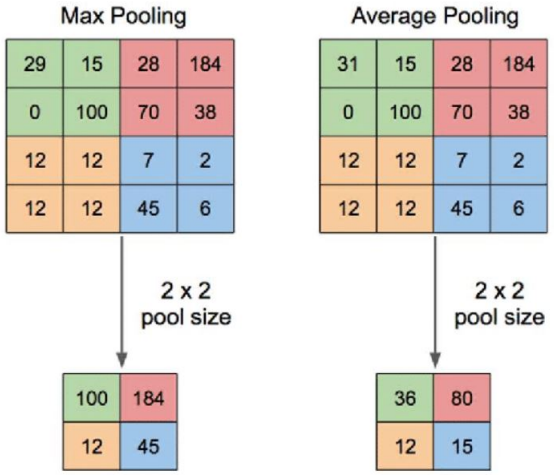


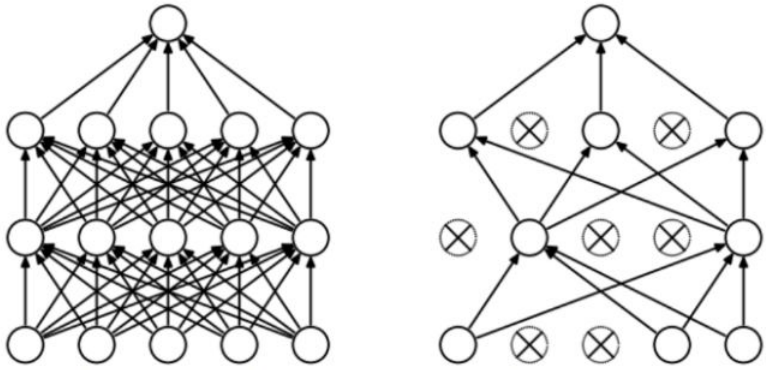
Figure 2.8: In Max Pooling we have a filter which slides over the input array and we take the max of that region and create a new matrix in which every element is the max of each region in the original array. In Average Pooling we take the average in each region [20].

makes it very useful for convolutional networks. Invariance to translation offers us information about whether some feature is present in the input data than where it is exactly. If we want to classify images of variable size, the input to the classification layer must have a fixed size, so pooling is needed to handle varying sizes. This is usually accomplished by varying the size between pooling regions so that the classification layer always receives the features regardless of the input size.

2.6 Dropout

Dropout [17] is a powerful method for regularizing cnns and prevent them from overfitting. This technique improves the performance of cnns in a wide variety of application domains including object classification. Overfitting happens when the model learns all the details and noise of the training data and as a result it has poor performance to new data. The dropout method ignores a set of neurons in visible or hidden layers in forward or backward pass by setting to 0.

Figure 2.9: Dropout Method



(a) Standard Neural Net (b) After applying dropout.

The benefits of using the dropout is that it forces the network to have more robust features because in each epoch different sets of neurons are dropped out. Also it makes sure that in training phase the network does not get overfitted as it has less neurons. One disadvantage of dropout is that it increases the training time. A dropout network may take 2 or 3 times longer to train than a standard network because the parameter updates are noisy as in each training case it trains at a different architecture. Thus the gradients produced at training will not be used at testing. Dropout has a new parameter p, which is the probability of retaining a unit p. At $p = 1$ there is no dropout and lower values of p means that there is a more intense dropout. In hidden layers the common values are in range of 0.5 to 0.8.
However there is a strong connection between the number of neurons n in hidden layers and the hyperparameter p. Smaller p requires big n and it can lead to overfitting and bigger p may not produce enough dropout to prevent

overfitting.

2.7 Data augmentation

A machine learning model performs better if we train it with more data. However, in practice data is limited. In order to face this problem we can create fake data and add it to training set. This technique is extremely useful for classification tasks. In classification a classifier takes a high dimensional input x and summarizes to a category y. Data augmentation can be done with transformations of the array such as rotations, random crops ,horizontal or vertical flips.

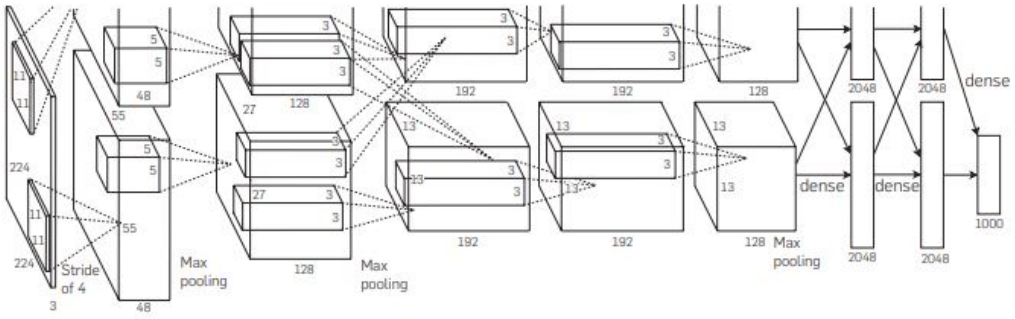
Figure 2.10: Data augmentation

As we see in the figure 2.10 we have various array transformations. Flipping is the inversion of the image so its facing the other side, colour jittering is the random change of the brightness, contrast and saturation of the image. Edge enhancement enhances the contours of the class represented within the image. Fancy PCA performs PCA on sets of RGB pixels throughout the training set by adding multiples of principles components to the training images. With cropping we create a random subset of the original image. Rotating is the rotation of the image in a certain angle [19].

2.8 AlexNet

AlexNet is a convolutional neural network designed by Alex Krizhevsky, in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor [5]. It competed on the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012 and achieved a winning top -5 best error rate of 15.3%, compared to 26.2% achieved by the second- best entry. This network utilizes the graphics processing units (GPUs) during training. For the ImageNet competition the network was trained using 2 GTX 580 GPUs with 3GB of memory. The architecture summarizes in figure 2.11.

Figure 2.11: AlexNet



It contains eight layers, which five of them are convolutional and three are fully connected. The output of the neurons of every layer are going through ReLU activation function because it is faster than using for example hyperbolic tangent(tanh). The last layer produces an output that is fed to 1000-way softmax classifier that produces a distribution over the 1000 class labels. The network maximizes the multinomial logistic regression meaning that it maximizes the log probability of the correct labels under the prediction distribution. The kernels of the second, fourth and fifth layer are connected to only those feature maps of the previous layer that is in the same GPU. The kernel of the third layer is connected to all the feature maps of the second layer. The fully connected layers are connected to all the previous neurons of the previous layer. There are response-normalization layers that are followed after the first and second convolutional respectively.

The response-normalization is applied after ReLU activation function and is given by the expression :

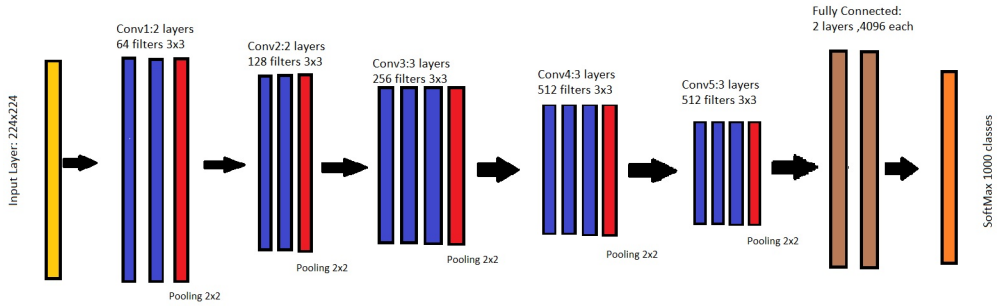
$$b^i_{x,y} = \frac{\alpha^i_{x,y}}{(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (\alpha^j_{x,y})^2)^\beta}$$

where the sum runs over n adjacent feature maps at the spatial position and N is the total number of kernels in the layer. The k, n, α, β are hyper-parameters whose values is determined by running a validation set. Max pooling layers are after the normalization layers and the fifth convolutional layer. As we see in figure the first convolutional layer filters the 224x224x3 image with 96 filters of size 11x11x3 with a stride of 4. The second convolutional layer filters the input with 256 kernels of size 5x5x48. The third layer has 384 filters of size 3x3x256, the fourth 3x3x192 filters and fifth 256 filters of size 3x3x192. The fully connected layers have 4096 each. To prevent overfitting AlexNet utilizes data augmentation and dropout method. In the case of dropout it sets the hyperparameter p to 0.5 meaning that each neuron output has a probability of 0.5 to be 0 and as a result it nearly doubles the number of iterations needed to converge. In case of data augmentation it creates patches of 224x224 from 256x256 images. The number of parameters AlexNet has is 60 million.

2.9 Vgg16

Vgg was introduced at ImageNet competition in 2014 and was designed by Karen Simonyan and Andrew Zisserman [16] and achieved top -5 best error rate of 7.3%.

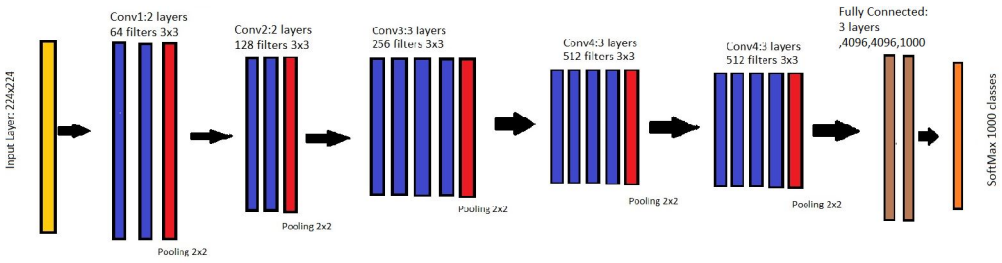
Figure 2.12: Vgg16



On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2–3 weeks depending on the architecture. Vgg16 consists of 13 convolutional layers and 3 linear layers. Vgg16's input is images of size 224x224x3. It consists of 5 blocks of layers and then 3 linear. The first block has two convolutional layers that have 64 filters of 3x3. After a max pool layer is applied. The next block has 2 convolutional layers have 128 filters of size 3x3 and max pool layer. The third block has 3 convolutional layers of 256 filters with size 256x256.Fourth block has 3 convolutional layers with 512 filters of size 512 and a max pool layer. Lastly fifth block contains 3 convolutional layers with 512 filters of size 3x3 and a max pool layer. Then we flatten the output in order to feed the linear layers of size 4096 , 4096, 1000. The output of the third linear layer goes through a Softmax layer. Vgg16 has 138 milion parameters.

2.10 Vgg19

Figure 2.13: Vgg19

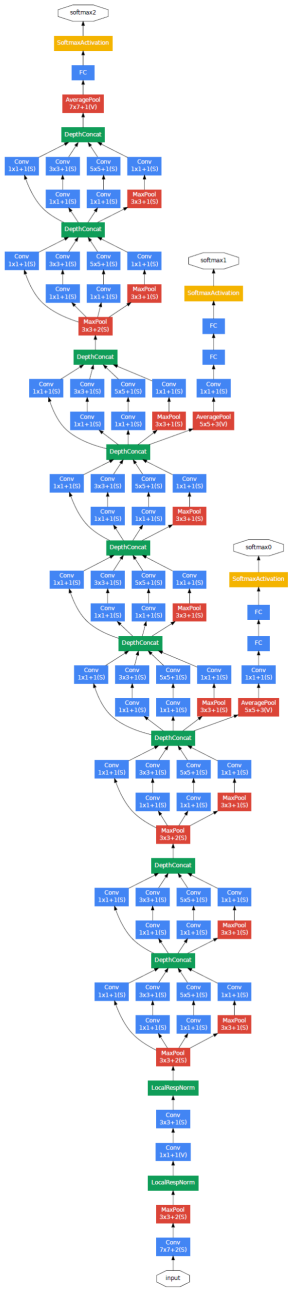


Vgg19's input is images of size 224x224x3. It consists of 5 blocks of layers and then 3 linear. The first block has two convolutional layers that have 64 filters of 3x3. After a max pool layer is applied. The next block has 2 convolutional layers have 128 filters of size 3x3 and max pool layer. The third block has 4 convolutional layers of 256 filters with size 256x256. Fourth block has 4 convolutional layers with 512 filters of size 512 and a max pool layer.Lastly fifth block contains 4 convolutional layers with 512 filters of size 3x3 and a max pool layer. Then we flatten the output in order to feed the linear layers of size 4096, 4096, 1000. The output of the third linear layer goes through a Softmax layer. Vgg19 has 144 million parameters.

2.11 GoogLeNet

GoogLeNet [18] is the winner of 2014 ImageNet Large Scale Visual Recognition Challenge with 6.7% top-5 best error rate. It consists of 22 convolutional layers and uses the idea Inception network architecture. It was designed to have low computational cost. It takes images of 224x224x3 and convolutional layers uses ReLU activation function. GoogLeNet uses filters with size of 1x1 in convolution layers in order to decrease the number of parameters used and also at the end of the network for pooling it uses the global average pooling. Global average pooling takes 7x7 features maps and averages to 1x1. The inception architecture uses blocks of layers. One block consists of convolutional layers with 1x1, 3x3, 5x5 filters sizes and 3x3 max pooling which are working in parallel and the output is stacked in order to get the final output. Inception architecture also uses some intermediate classifiers. These blocks use 5x5 average pooling layer, a convolutional layer with 128 filters of size 1x1 and 2 linear layers of sizes 1024 and 1000 and a softmax layer. The number of parameters GoogLeNet has is 7 million.

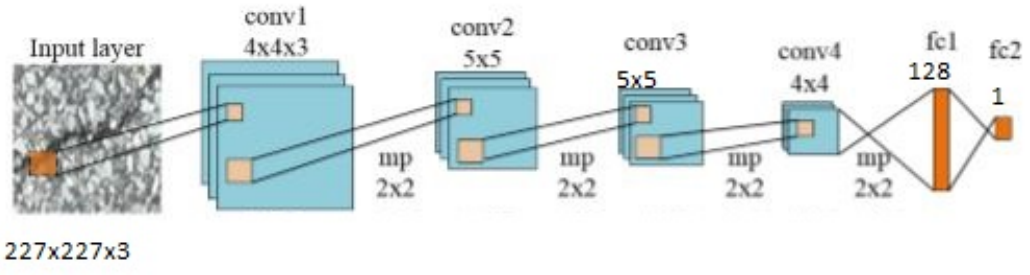
Figure 2.14: GoogLeNet



2.12 Custom CNN

For this study we made a custom convolutional neural network based on the paper of [22] in order to make a low computational cost model for road classification. It consists of 4 convolutional layers, followed by 2 linear layers and a sigmoid activation function. The output of neurons of every layer is going through a ReLU activation function and after that a maxpool 2x2 layer is applied. The first convolution layer has filters of size 4x4, the second 5x5, the third 5x5 and the fourth 5x5. The linear layers have input of 128 and 1. The number of parameters produced are 4.747.489. For the training we use stochastic gradient descent (SGD)

Figure 2.15: Custom CNN



Chapter 3

Quaternion Neural Networks

3.1 Quaternion algebra

Quatenions ([1],[23],[15]) are hyper complex numbers and were introduced by Hamilton in 1843.Quaternions are four-dimensional, where the field can be described as $\mathbb{H}$ or $\mathbb{R}^4$ and each quaternion q can be represented as :

$$q = a + bi + cj + dk$$

where a,b,c,d $\in \mathbb{R}$ and i,j,k are independent imaginary units.Thus quaternions consists of 1 real and 3 imaginary parts.The imaginary units i, j, k obey the quaternion rules

$$i^2 = j^2 = k^2 = ijk = -1$$

and

$$ij = -ji = k, jk = -kj = i, ki = -ik = j$$

In the same way as real numbers we can define some operations for quaternions:
Addition : $p + q = (a_p + a_q) + (b_p + b_q)i + (c_p + c_q)j + (d_p + d_q)k$

Scalar Multiplication: $\lambda q = \lambda a + \lambda bi + \lambda cj + \lambda dk$

Element multiplication: $pq = (a_p a_q - b_p b_q - c_p c_q - d_p d_q) + (a_p b_q + b_p a_q + c_p d_q - d_p c_q)i + (a_p c_q - b_p d_q + c_p a_q + d_p b_q)j + (a_p d_q + b_p c_q - c_p b_q + d_p a_q)k$

where $p = a_p + b_p i + c_p j + d_p k$ and $q = a_q + b_q i + c_q j + d_q k$

Conjugation: $\overline{q} = a - bi - cj - dk$

Quaternion magnitude is described as:

$$|q| = \sqrt{q\overline{q}} = \sqrt{\overline{q}q} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

In addition we can assume the real part of a quaternion as sum of scalar ($S(q)$) and the imaginary part as a three dimensional vector ($V(q)$) and we can write the quaternion as :

$$q = S(q) + V(q) = (a + 0i + 0j + 0k) + (0 + bi + cj + dk)$$

Thus with the above expression the element multiplication which is also called Hamilton product can be written as :

$$pq = S(p)S(q) - V(p) \cdot V(q) + S(p)V(q) + S(q)V(p) + V(p) \times V(q)$$

where the $\cdot$ is the dot product and $\times$ is the cross product.In case the real part of p and q is zero then there is only cross product.
A normalized or unit quaternion can be described as :

$$q^{\triangleleft} = \frac{q}{\sqrt{a^2 + b^2 + c^2 + d^2}}$$

Quaternions can be represented as matrices. One way is to use 2x2 complex matrices and the other is to use 4x4 real matrices. A 2x2 complex matrix of quaternion can be described as :

$$\begin{bmatrix} a + bi & c + di \\ -c + di & a - bi \end{bmatrix}$$

If $c = d = 0$ we have a diagonal complex matrix and if $b = d = 0$ we have real matrix representation of complex numbers and the conjugate of a quaternion is the conjugate transpose (or Hermitian transpose) of the matrix. Using a 4x4 matrix the quaternion can be described as :
the conjugate of a quaternion corresponds to the transpose of the matrix.

$$\begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} = a\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + b\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} +$$

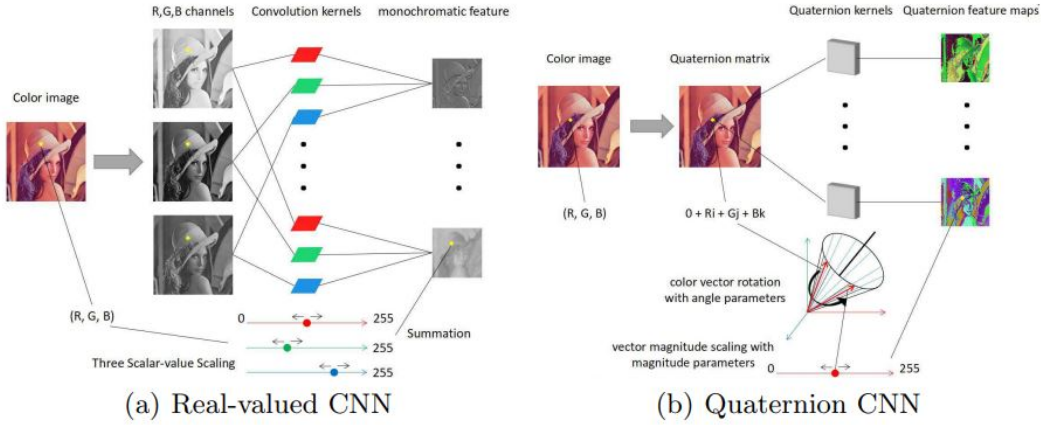
$$c\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} + d\begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

In recent years there have been a lot of studies that propose algorithms based on quaternions that work better in 3-D objects than a real valued. A far as computer vision is concerned, quaternion based methods can extract more representative features for images and have very good results in image classification. In low level tasks quaternion methods keep more information about color and thus can restore images in higher quality. Quaternion algebra gives a boost to performance of CNNs and also allows sharing quaternion weights between channels during the product operation,result in keeping more useful information. Quaternions can be use to create convoltuional and linear layers.

3.2 Quaternion Convolutional Layer

Quaternion convolutional neural networks [11] [12] are introduced to replace the successful convolutional neural networks and are composed of quaternionic parameters,inputs,activation functions and outputs. In general a quaternion filter $g \in \mathbb{H}^{K \times K}$ is fed on an input feature map $f \in \mathbb{H}^{M \times N}$ to produce the output feature map $g \in \mathbb{H}^{(M+K-1)\times(N+K-1)}$

Figure 3.1: CNN-QCNN



(a) Real-valued CNN (b) Quaternion CNN

Picture taken from [23]

Figure 3.1: In the right image we have a real valued CNN which we feed a rgb image. Every channel of the image is convoluted with the channel of the kernel in order to produce an output and then the outputs of the three channels are summed in order to produce the final feature map which is monochromatic. However in the quaternion CNN we turn the rgb image into a quaternion matrix in which every element is a quaternion containing the rgb values of the pixel in its imaginary part. As a result when the convolution is done between the quaternion matrix and the kernel, the Hamilton product will produce an output which is a quaternion and has kept the rgb values in the imaginary part.

If γ_{ab^l} and S^l_{ab} are the quaternion output and the activation quaternion output at layer l and (a, b) indexes of the new feature map, and w the quaternion valued weight filter map of size $K \times K$. Convolution is described as :

$$\gamma^l_{ab} = a(S^l_{ab})$$

with

$$S^l_{ab} = \sum_{c=0}^{K-1} \sum_{d=0}^{K-1} w^l \otimes \gamma^{l-1}_{(a+c)(b+d)}$$

where a is the activation split function. With splitting we use a real-valued activation function to process each component of quaternion value. The split activation function α is defined as:

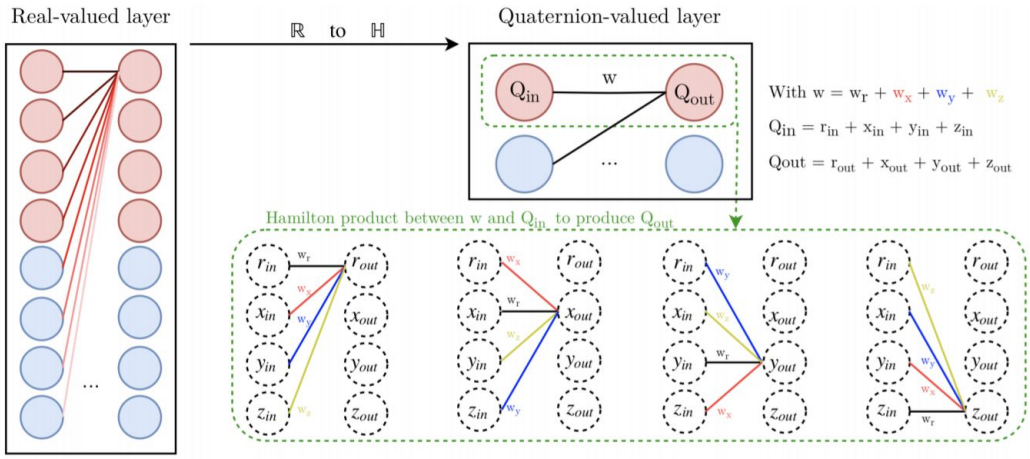
$$\alpha(q) = f(a) + f(b)i + f(c)j + f(d)k$$

where f is any standard activation function.

The output layer of a quaternion neural network quaternion valued or real valued based on the split activation function.
An RGB image can represented in the quaternion domain as:

$$q(p_{x,y}) = 0 + R(p_{x,y})i + G(p_{x,y})j + B(p_{x,y})k$$

Figure 3.2: Hamilton product



Picture taken from [11]

As seen in the figure 3.2 the Hamilton product shares the quaternion weights through multiple quaternion-input parts, creating relations within the elements. In the real valued cnns the weights required to code latent relations for a specific feature are in the same level as the global relations for different features, however in quaternion layers the weight codes these internal relations are within a unique quaternion Q_{out} . The Hamilton product ($\otimes$) replaces the standard real valued dot product and can be defined between two quaternions Q_1 and Q_2 as:

$$Q_1 \otimes Q_2 = (r_1 r_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) + (r_1 x_2 + x_1 r_2 + y_1 z_2 - z_1 y_2)i$$

$$+(r_1 y_2 - x_1 z_2 + y_1 r_2 + z_1 x_2)j + (r_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 r_2)k$$

Chapter 4

Image Segmentation

4.1 Introduction

Image segmentation is the process of creating multiple segments or sets of pixels(or image objects) in an image. It's main goal is to assign labels to every pixel of an image in a way that pixels with the same label share same characteristics. The characteristics could be color, intensity, texture. It has many applications in practical tasks such as face detection, medical imaging, face recognition and there have been developed many algorithms for image segmentation.

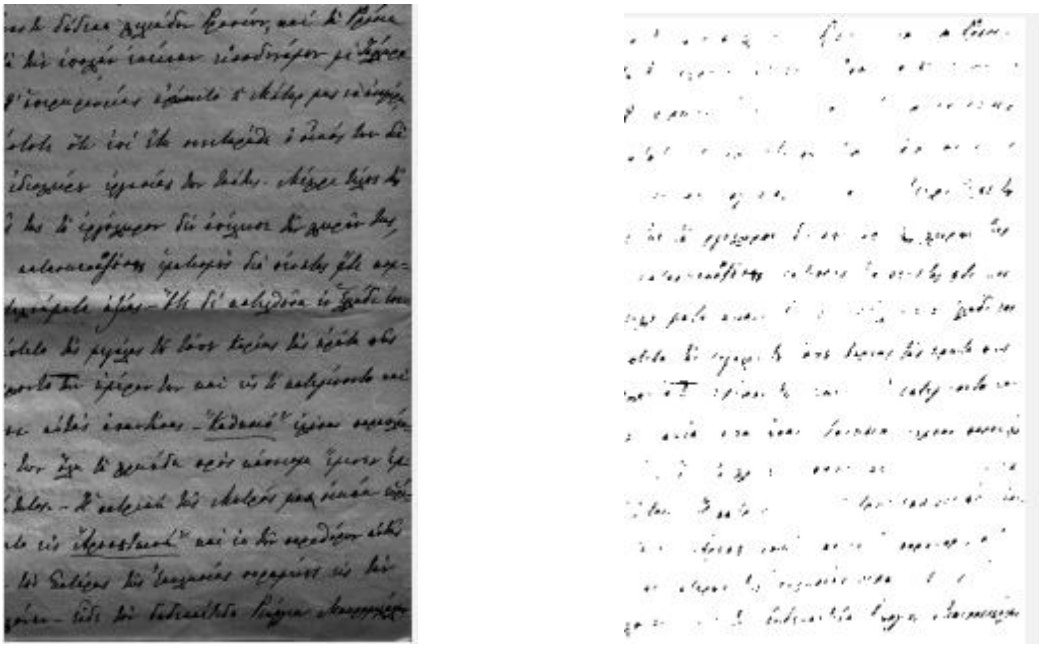
Figure 4.1: Image Segmentation

4.2 Thresholding

The first and simplest method of segmentation is thresholding. Thresholding requires to select a threshold value based on which pixel values take a certain value if they have more or less intensity. As a result, pixels with similar intensity belong to the same segment. Selecting a single value for thresholding creates two segments and is called bilevel thresholding.

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > k \\ 0 & \text{if } f(x,y) \leq k \end{cases}$$

Selecting multiple values creates multiple segments and is called multi-level thresholding. There are multiple thresholding methods such as Otsu's method.



(a) Original (b) Threshold

Figure 4.2: Threshold

4.2.1 Otsu's method

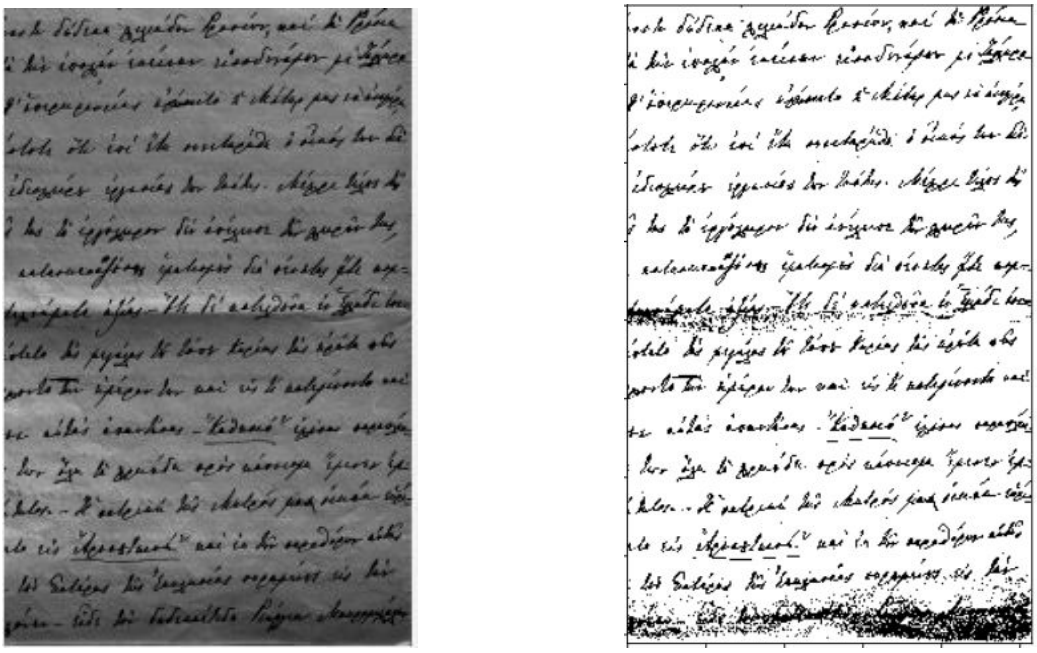
Otsu's method in its simplest form return a single threshold value that separates pixels into two segments,foreground and background. This value is returned by minimizing the variance between the segments. The variance can be thought as how much dispersion the pixels have. The core idea is separating the image histogram into 2 segments with a threshold value defined by the minimization of the weighted variance. This equation is expressed as:

$$\sigma_w^2 = \omega_1(t)[m_1(t) - m]^2 + \omega_2(t)[m_2(t) - m]^2$$

where :

$$\omega_j(t) = n_j(t) / \sum_{j'} n'_j(t)$$

$n_j(t)$ is the number of pixels that belong to class j
$m_j(t)$ is the mean value intensity of pixels that belong to class j
m is the mean value intensity of all pixels



(a) Original

(b) Otsu threshold t=71

Figure 4.3: Otsu

4.2.2 Adaptive thresholding

In adaptive thresholding we find a threshold value by approximating a value for a local set of pixels. Each pixel defines a neighborhood of pixels and we use mean value and standard deviation to find a local threshold. One such method is the **Sauvola** algorithm

$$T_{xy} = m(x,y) + (1 - k(1 - \frac{s(x,y)}{R}))$$

where k=0.5 and R=0.128 and **moving averages**:

$$T_{xy} = km(x,y)$$

In **Bernsen's method** the threshold value is calculated for each pixel from the following expression:

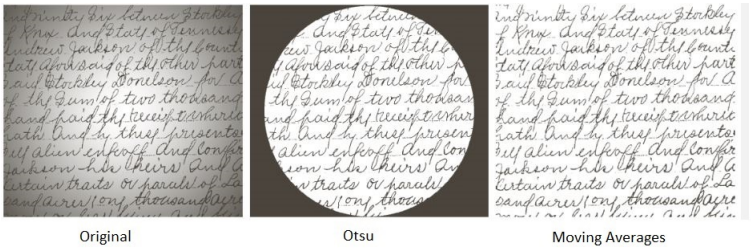
$$T(x,y) = \frac{Z_{low} + Z_{high}}{2}$$

where Z_{low},Z_{high} are the lowest and highest pixel values in the neighborhood whose center is (x,y). If the contrast value $C(x,y) < 1$ then the neighborhood has only one class.
Niblack method calculates a threshold value by sliding a rectangular window over the image:

$$T_{Niblack} = m + k * s$$

where m is mean and s is standard deviation.

Figure 4.4: Moving averages threshold



4.2.3 Isodata Thresholding

Isodata thresholding can automatically detect a threshold for a gray scale image. If we set threshold value as t,then we can express isodata thresholding as:

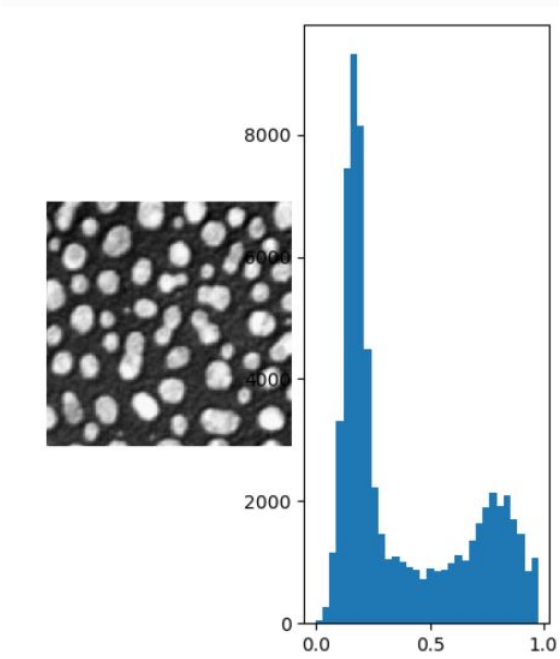
$$t = \frac{m_L(t) + m_H(t)}{2} = m(t)$$

where m_L is the mean of pixels with value less than t and m_H the mean of pixels that have value greater than t.

4.3 Histogram-based Methods

Histogram based methods are very efficient because they require only one pass in the image. They create a histogram from all the pixels and the peaks and valleys of it depicts the clusters in the image. However one drawback of the histogram seeking methods is that sometimes it is difficult to distinguish peaks and valleys in the image.

Figure 4.5: Histogram



4.4 Clustering

Clustering is a technique that can be used in an image that we do not know what objects depict with very good results. The pixels are clustered with similarity criteria

4.4.1 K-Means

K-Means is a data clustering algorithm that tries to put every pixel to one of k clusters. The pixels in the same cluster are as simlar as possible. K-means tries to minimize the following function:

$$(\sum_{j=1}^{k}\sum_{n=1}^{N} z_j^n ||x^n - m_j||^2)$$

where m is the centroid and the norm is usually euclidean distance. The steps of the algorithm are the following:
1.Pick K cluster centers randomly.
2.Assign each pixel in the image to the cluster that hasthe least distance using the previous function.
3.Update the cluster by averaging all of the pixels in the cluster:

$$m_j = \frac{\sum_{x \in S_j(t)} x}{n_j}$$

4.Repeat 2 and 3 until no pixels change the clusters.
A major drawback of K-Means is that the initialization of weights greatly affects the performance of the method.

Figure 4.6: K-means

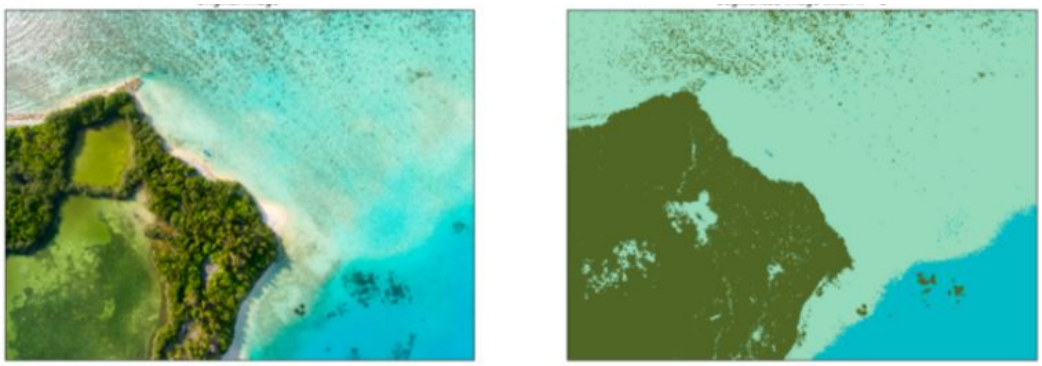


Image Segmentation when K=3

Photo taken from https://www.kdnuggets.com/2019/08/introduction-image-segmentation-k-means-clustering.html. In the figure we see the result of K-Means with 3 clusters.

4.4.2 Fuzzy C-Means Clustering

Fuzzy C-Means Clustering [3] was developedby J.C. Dunn in 1973. The algorithm is follows the steps below:
1.We choose a number of clusters.
2.We randomly assign coefficients to each input.
3.Repeat until the algorithm has converged,meaning that the coefficients have not changed between two iterations over some value ε:
3.1 Compute the centroid of each cluster:

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m}$$

where m is the hyperparmeter of how fuzzy the cluster is and is called fuzzifier . The algorithm tries to minimize the objective function:

$$arg_C min \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^m ||x_i - c_j||^2$$

where

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} (\frac{||x_i - c_j||}{||x_i - c_k||})^{\frac{2}{m-1}}}$$

the w_{ij} is called the contribution matrix and tells the degree the input x_i belongs to cluster c_j.

4.5 Region-growing methods

In region growing we consider initially seed points S(x,y). Similar neighboring points are united as one segment and this keeps happening until different segments do not meet the similarity criteria. Similarity criteria could mean value or standard deviation, color, texture.

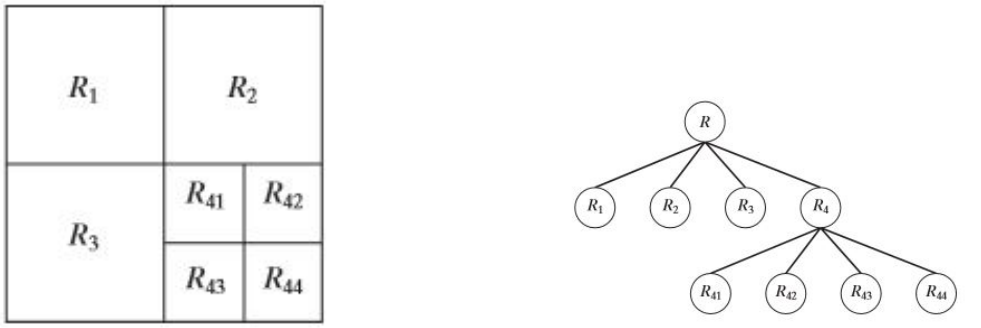
Figure 4.7: Region Growing

Image taken from the book of Ψηφιακή επεξεργασία & ανάλυση εικόνας, Παπαμάρκος Νικόλαος Η.

The advantages of region growing methods is that can separate with good results regions that have same properties or clear edges. Also it can choose multiple criteria at the same time. As far the disadvantages are concerned, it is sensitive to noise, it is a local method and as result cannot view the problem in a global scale.

4.6 Region Splitting and Merging

The steps for the method is the following:
1.Split into four quadrants the regions R_i for which $Q(R_i) = FALSE$
2.When we cannot split anymore, we merge any adjacent region R_j and R_k for which $Q(R_j \cup R_k)$=TRUE
3.Stop when we cannot merge further.



(a) Partitioned image

(b) Quadtree.The root is the whole image

Figure 4.8: Region Splitting and Merging

Figure 4.8: Each level of partitioning can be represented in a tree like structure

Figure 4.9: Region Splitting

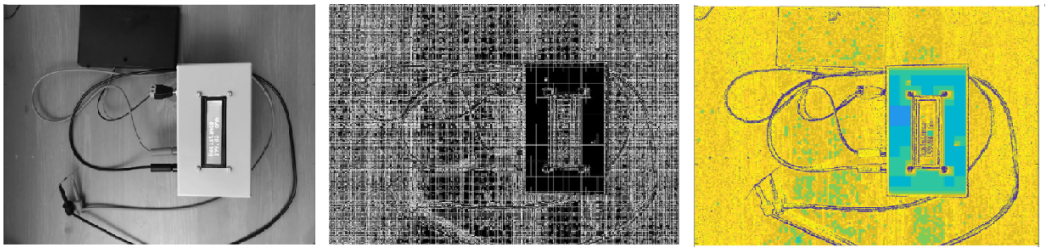


Figure 4.9 : Image taken from wikipedia: Split and merge segmentation. The homogeneity criterion is thresholding, max(region)-min(region) $\dot{1}$0 for a region to be homogeneous. The first image is the original, the middle shows the blocks created during splitting and the last image is the segmented image.

Chapter 5

Experiments

5.1 Library

For this study we used PyTorch, which is an open source machine learning library based on the torch library, developed by Facebook's AI Research lab in 2017. It is free and open sourced under the Modified BSD license. The advantages Pytorch offers is that it has built in deep neural networks and has tensors which can run with graphics processing units (GPU). Tensors are like NumPy Arrays, but they can run on Nvidia GPUs. GPUs makes training of deep learning much faster than CPU and that is because it consists of hundreds of simpler cores and thousand of concurrent hardware threads. As far as this study is concerned, we trained the model in a Nvidia Titan Xp with 12 GB of GDDR5X memory and a CPU AMD Ryzen 5 1600 Six-Core Processor and 32 GB Ram.

5.2 Datasets

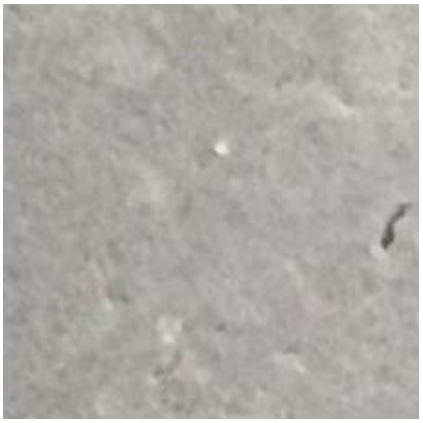
The dataset used for training the various deep learning models contains concrete images with cracks. The data is collected from various METU Campus Buildings [21]. The dataset is divided into two as negative and positive crack images for image classification. Each class consists of 20000 227x227x3 images. The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by [22]. However there has been not applied data augmentation such as random rotation or flipping.

(a) Crack

(b) Crack

(c) Crack

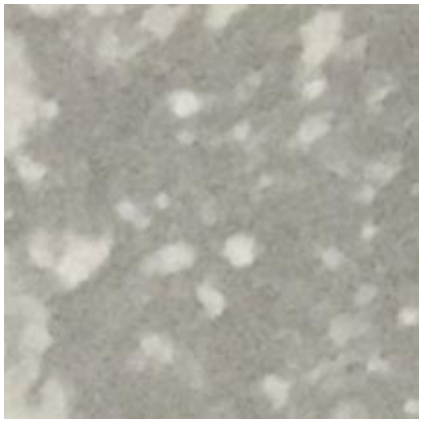
(d) Crack



(a) No Crack

(b) No Crack

(c) No Crack



(d) No Crack

Figure 5.1: Training images

For testing we use 2000 images for each class of this dataset and also one other dataset [2] which contains 400 images of size 448x448x3 which we resize them to 227x227, 200 for each class which contains images taken from asphalt. In this dataset there has been done no data augmentation.

(a) Crack

(b) Crack

(c) Crack

(d) Crack

(a) No Crack

(b) No Crack

(c) No Crack

(d) No Crack

Figure 5.2: Asphalt dataset

5.3 Parameters and training of models

In case of the custom CNN we train the model with a batch size of 10 images, running 1 and 10 epochs with a learning rate of 0.01 and stochastic gradient descent as the optimization algorithm and BCELoss as the loss criterion which measures the Binary Cross Entropy between the target and the output and can be described as :

$$l(x, y) = L = [l_1, ;_2, ...l_N]^T, l_n = -w_n[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)]$$

where N is the batch size.
The model has 4 convolutional layers and their outputs are going through a ReLU activation function and 2x2 maxpooling. Then the output of the last convolutional layer goes through a linear layer, the linear layer goes through a ReLU activation function and then the last linear layer goes through a Sigmoid activation function. In the quaternion version of this model we change the input of the model by creating quaternion matrix version of the image.

$$0 + Ri + Gj + Bk$$

Then we replace all the convolutional layers and linear with the quaternion version. The quaternion version code is taken from [12] which contains PyTorch modules like QuaternionLinear, QuaternionConv, or QuaternionTransposeConv. As far as the AlexNet and Vgg16 is concerned we train these models with the same parameters and in the quaternionic version we replace all the convolutional and linear layers with their quaternionic ones. At the output of the last quaternionic linear layer we sum the 4 channels in order to feed the real valued result to the sigmoid activation function.

5.4 Results

As we can see in the table below the custom CNN has the least number of parameters and the least number of convolutional layers. The results of Vgg19 and GoogleNet were taken from [10] where they used the same dataset for training and pretrained Pytorch Vgg16 and GoogLeNet models.

Model	# of conv layers	# of learnable parameters
AlexNet	5	58.285.441
Vgg16	13	33.609.793
Custom CNN	4	4.747.489
Vgg19	16	144.000.000
GoogLeNet	22	7.000.000

In the next table we compare the number of parameters between the models we implemented and their quaternion versions. As we can see quaternions reduced the number of parameters to about 75%. Also we modified the number of channels in the custom CNN in order to get the same parameters as the quaternionic and see its results. we named it SMCNN(Same Parameter CNN)

Model	Quaternionic models	Convolutional models
AlexNet	14.584.513	58.285.441
Vgg16	8.421.313	33.609.793
Custom CNN	1.187.553	4.747.489
Custom SMCNN		1.188.037

For training we used different sizes as shown in the next table. 70% of images are randomly selected for training, 15% for validation and 15% for testing. For testing we also use the [2] dataset. Thus the biggest training dataset has 28K images. Then we reduce randomly the training dataset 28K to 21K, 14K, 7K, 3.5K, 1.75K, 0.7K and 0.35K to investigate the relation between performance and size of the training dataset. All datasets have equal number of crack and non crack images.

Size	Train Positive	Train Negative	Val Positive	Val Negative
28K	14.000	14.000	3.000	3.000
21K	10.500	10.500	2.250	2.250
14K	7.000	7.000	1500	1500
7K	3.500	3.500	750	750
3.5K	1750	1750	375	375
1.7K	875	875	190	190
0.7K	350	350	75	75
0.35K	175	175	40	40

The computational time in seconds to train a 28K dataset per epoch for each model and its quaternion version is shown below

Model	Convolutional models	Quaternion models
AlexNet	70	90
Vgg16	277	355
Custom CNN	80	103

As we can see the computational time of the quaternion version is more demanding because of the complexity of the operations and is increased by 21%.

In the following tables we show the maximum accuracy and F-score results in 1 epoch and 10 epochs obtained by the convnets and quaternion convnets respectively by using the test dataset of concrete dataset [21]

10 epoch Accuracy	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.989	0.997	0.997	0.994
21K	0.982	0.996	0.989	0.994
14K	0.942	0.996	0.991	0.994
7K	0.935	0.992	0.987	0.989
3.5K	0.931	0.984	0.981	0.983
1.75K	0.741	0.984	0.971	0.971
0.7K	not stable	0.966	0.840	0.968
0.35K	not stable	0.959	not stable	0.943

10 epoch Accuracy	Custom CNN	Q.Custom	SMCNN
28K	0.997	0.996	0.994
21K	0.993	0.992	0.990
14K	0.991	0.995	0.992
7K	0.989	0.991	0.990
3.5K	0.978	0.985	0.976
1.75K	0.968	0.971	0.978
0.7K	0.908	0.932	not stable
0.35K	0.51 not stable	not stable	not stable

10 epoch F-score	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.985	0.994	0.996	0.991
21K	0.979	0.994	0.987	0.993
14K	0.937	0.993	0.990	0.993
7K	0.932	0.990	0.985	0.987
3.5K	0.928	0.982	0.978	0.977
1.75K	0.634	0.977	0.968	0.968
0.7K	not stable	0.956	0.803	0.962
0.35K	not stable	0.959	not stable	0.932

10 epoch F-score	Custom CNN	Q.Custom	SMCNN
28K	0.996	0.995	0.992
21K	0.992	0.992	0.990
14K	0.990	0.993	0.992
7K	0.988	0.988	0.987
3.5K	0.976	0.982	0.973
1.75K	0.963	0.966	0.977
0.7K	0.890	0.921	not stable
0.35K	0.64 not stable	not stable	not stable

1 epoch Accuracy	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.825	0.825	0.984	0.983
21K	0.803	0.817	0.957	0.973
14K	0.772	0.849	0.937	0.981
7K	0.756	0.847	0.891	0.971
3.5K	0.642	0.817	0.911	0.951
1.75K	0.544	0.765	0.713	0.924
0.7K	not stable	0.782	not stable	0.854
0.35K	not stable	not stable	not stable	not stable

1 epoch Accuracy	Custom CNN	Q.Custom	SMCNN
28K	0.983	0.987	0.982
21K	0.985	0.987	0.970
14K	0.964	0.982	0.952
7K	0.883	0.976	0.672
3.5K	0.711	0.878	not stable
1.75K	0.728	0.839	not stable
0.7K	0.512	0.754	not stable
0.35K	0.50 not stable	not stable	not stable

1 epoch F-score	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.837	0.837	0.982	0.977
21K	0.792	0.768	0.953	0.970
14K	0.768	0.769	0.922	0.979
7K	0.762	0.812	0.863	0.964
3.5K	0.631	0.751	0.914	0.943
1.75K	0.598	0.774	0.612	0.909
0.7K	not stable	0.722	not stable	0.812
0.35K	not stable	not stable	not stable	not stable

1 epoch F-score	Custom CNN	Q.Custom	SMCNN
28K	0.981	0.984	0.972
21K	0.978	0.982	0.964
14K	0.962	0.980	0.943
7K	0.878	0.973	0.51
3.5K	0.748	0.873	not stable
1.75K	0.700	0.837	not stable
0.7K	0.654	0.745	not stable
0.35K	not stable	not stable	not stable

From the asphalt dataset [2] we have the following results for 10 epochs:

10 epoch Accuracy	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.795	0.770	0.647	0.682
21K	0.762	0.722	0.575	0.745
14K	0.747	0.787	0.675	0.703
7K	0.820	0.712	0.755	0.817
3.5K	0.835	0.830	0.615	0.772
1.75K	0.782	0.835	0.675	0.812
0.7K	not stable	0.802	0.720	0.767
0.35K	not stable	0.772	not stable	0.850

10 epoch Accuracy	Custom CNN	Q.Custom	SMCNN
28K	0.802	0.655	0.807
21K	0.837	0.660	0.817
14K	0.851	0.740	0.857
7K	0.795	0.810	0.742
3.5K	0.797	0.857	0.712
1.75K	0.832	0.790	0.637
0.7K	0.760	0.830	0.595
0.35K	not stable	0.770	0.545

10 epoch F-score	AlexNet	Vgg16	Q.AlexNet	Q.Vgg16
28K	0.737	0.661	0.405	0.513
21K	0.783	0.520	0.212	0.607
14K	0.728	0.717	0.440	0.530
7K	0.645	0.560	0.652	0.786
3.5K	0.801	0.754	0.330	0.682
1.75K	0.716	0.806	0.465	0.746
0.7K	not stable	0.801	0.585	0.765
0.35K	not stable	0.765	not stable	0.824

10 epoch F-score	Custom CNN	Q.Custom	SMCNN
28K	0.727	0.436	0.738
21K	0.746	0.463	0.758
14K	0.803	0.593	0.797
7K	0.715	0.743	0.629
3.5K	0.736	0.843	0.657
1.75K	0.784	0.719	0.380
0.7K	0.652	0.788	0.277
0.35K	0.659 not stable	0.687	0.130

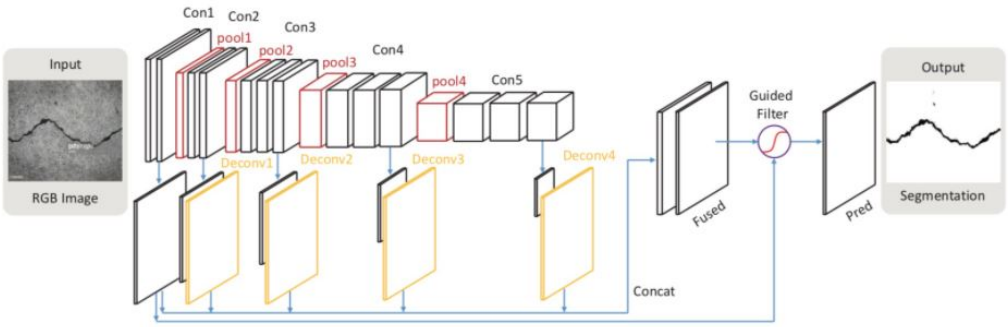
5.5 Analysis of results

Concerning the number of parameters we see that quaternion models have about 75 % less parameters and that is because of quaternionic operations. However the computational time needed is increased to 21% due to the complexity of the operations. Concerning the performance in datasets, first of all in the concrete crack dataset its is observed that higher scores are obtained from higher epochs and larger datasets.
All networks are benefitting from large datasets, achieving best scores of 0.99% from 21k and above. Vgg16 achieves best score even with 7k. At 1 epoch we see that the custom cnn has better results at larger datasets but at smaller dataset it loses accuracy faster than Vgg16. Alexet comes third in terms of perfomance
Comparing the common cnns with the quaternionic version we see that the latter cnns achieve best results even with 1.75k datasets and in general have better accuracy and F-score. The quaternion versions offer better results with 1 epoch than the common increasing the models perfomance with more than 10% at smaller datasets.
In order to see whether the networks have good performance we test the models using another dataset which contains photos with cracks taken from asphalt and has different surface and material.Concerning the AlexNet we see that overall the CNN model has better results than quaternionic and shows best results in datasets with size 3.5k and 7k. That means that in larger datasets it shows signs of overfitting.The same happens also in Vgg16. However in the custom cnn we see that the SMCNN and the quaternionic custom cnn have the same parameters but in smaller datasets the quaternionic shows better results but in bigger datasets the SMCNN shows better results. That means that the QCNN shows overfitting in bigger datasets and SMCNN does not have good accuracy with smaller datasets.
To conclude we see that quaternions offers better performance in smaller datasets and especially Vgg16 which seems good for generic crack detection.

5.6 DeepCrack

DeepCrack [7] is a deep hierarchical feature learning architecture for crack segmentation. The model architecture is depicted in figure 5.3.

Figure 5.3: DeepCrack Architecture

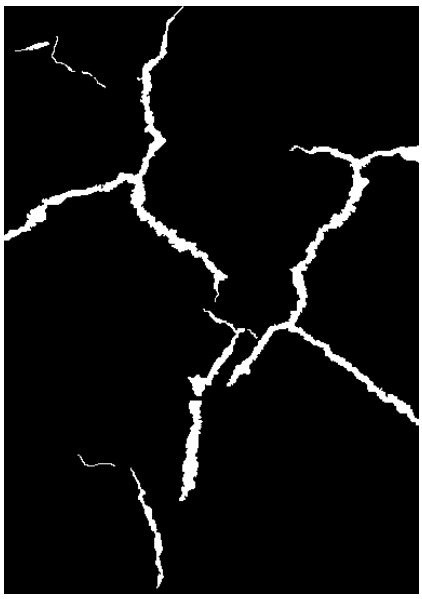


It consists of 13 convolutional layers, where each layer is comprised of a convolution, batch normalization and Rectified Linear Unit(ReLU). The spatial pooling is done with 4 maxpooling layers of filter size 2x2 and stride 2. The side output features are obtained from a convolutional layer with filter size 1 and output N. Except the first side output layer, the other four side output layers are followed by deconvolution layers, which are applied in order to to upsample the feature maps so as to be the same size as the input image. Then the output features are concatenated to form final features followed by a convolutional layer and a Sigmoid layer. The output of the fused layer might have noise because of the dirt and dark spots and to deal with this problem the architecture applies Guided Filtering, which uses the output of the first side output layer as a guidance map to the segmented image generated by the fused prediction.
To recap the architecture consists of three parts: Firstly the 13 convolutional layers, secondly the side output layers and thirdly the Guided Filtering. The number of parameters the architecture has is 14.7 million.

5.7 DeepCrack Dataset

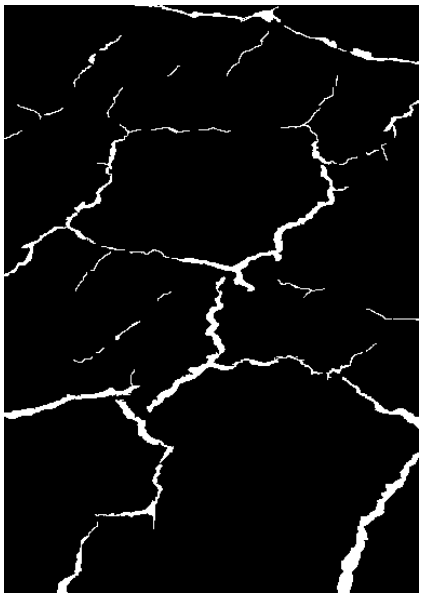
The dataset consists of 537 RGB images of size 544x384, 300 for training and 237 for testing with manually annotated segmentations. Each image has a segmentation map which is a mask covering exactly the crack regions. In training set the percentage of crack pixels is 2.91% and in test set it is 4.33%. For the training phase there has been done data augmentation in the images. They have been rotated to 8 different angles and they have been flipped at each angle horizontally. Also they have been resized to 256x256.

(a) Crack



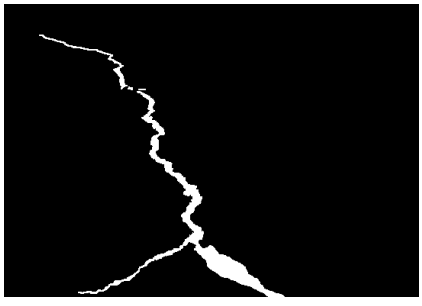
(b) Crack Label

(a) Crack



(b) Crack Label

(a) Crack

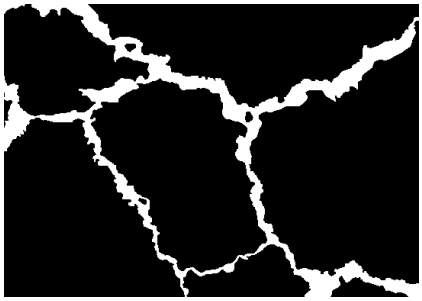


(b) Crack Label

(a) Crack

(b) Crack Label

(a) Crack



(b) Crack Label

Figure 5.4: Deepcrack dataset

5.8 Metrics

Let the n_{ij} be the number of pixels of the class i predicted to be the class j, where n_{cls} is the number of different classes and $t_i = \sum_j n_{ij}$ is the total number of pixels of the class i (both true and false positives). The metrics used for evaluating the network are the following:

Global Accuracy(G),measures the percentage of pixels that have been predicted correctly :

$$\frac{\sum_i n_{ii}}{\sum_i t_i}$$

Precision(P):

$$\frac{TruePositives}{TruePositives + FalsePositives}$$

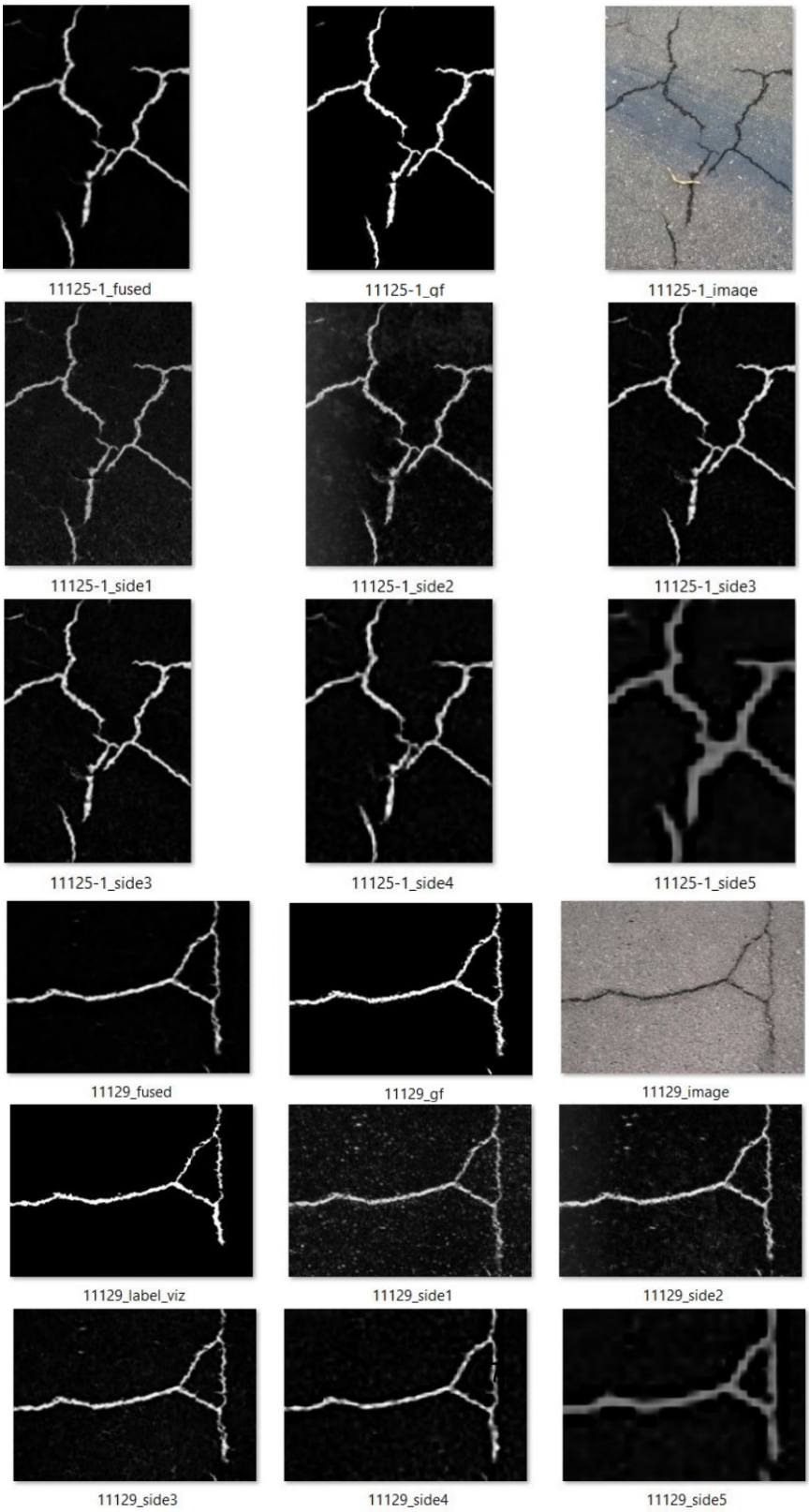
Recall(R):

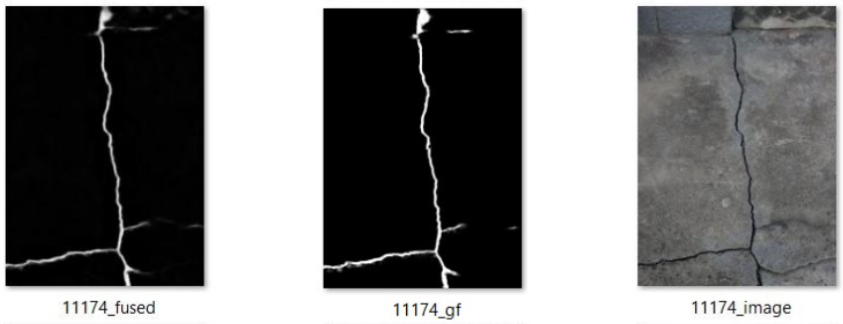
$$\frac{TruePositives}{TruePositives + FalseNegatives}$$

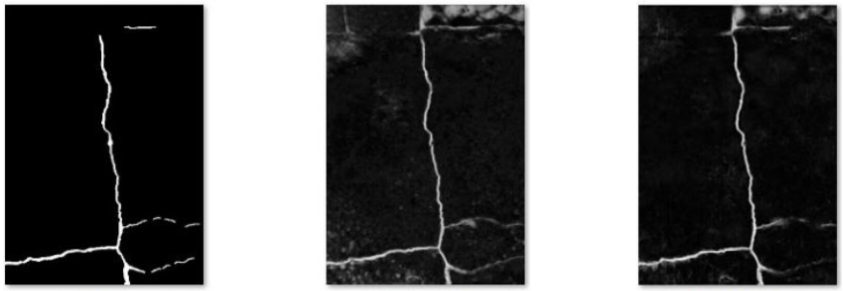
F-Score(F):

$$\frac{2PR}{R + R}$$

Figure 5.5: DeepCrack Output







In the images above we see the results produced from 3 different images. For each image we have 9 results. First is the result produced from fused layer, second is the Guided Filtering, third is the original image, fourth is the ground truth label which shows the true segmentation and the other 5 are the results from the side layers 1-5.

5.9 Quaternion DeepCrack

In quaternion DeepCrack we swap every convolutional layer with a quaternionic convolutional layer. To use a quaternionic convolutional layer we need to create a quaternion matrix from the image.

$$0 + Ri + Gj + Bk$$

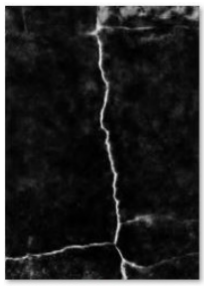
At the outputs of the side output and the fused layers we sum the 4 channels in order to feed the activation function with real valued results. The number of parameters used in common DeepCrack and the quaternion is shown below

Model	Quaternionic model	Convolutional model
DeepCrack	3.628.000	14.720.000

As we see the quaternion model uses 75% less parameters. The computational time needed per epoch for each model is

Model	Quaternion model(s)	Convolutional model(s)
DeepCrack	13	8

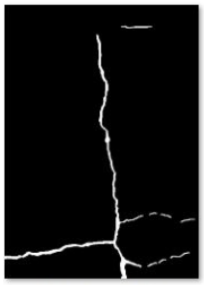
The computational time increased by 62%
The outputs from side layers, the fused and the Guided Filter are shown below



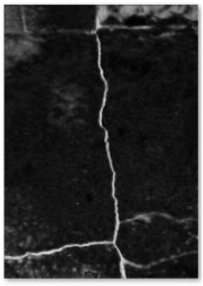
11174_fused

11174_gf

11174_image



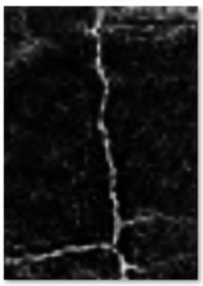
11174_label_viz



11174_side1

11174_side2

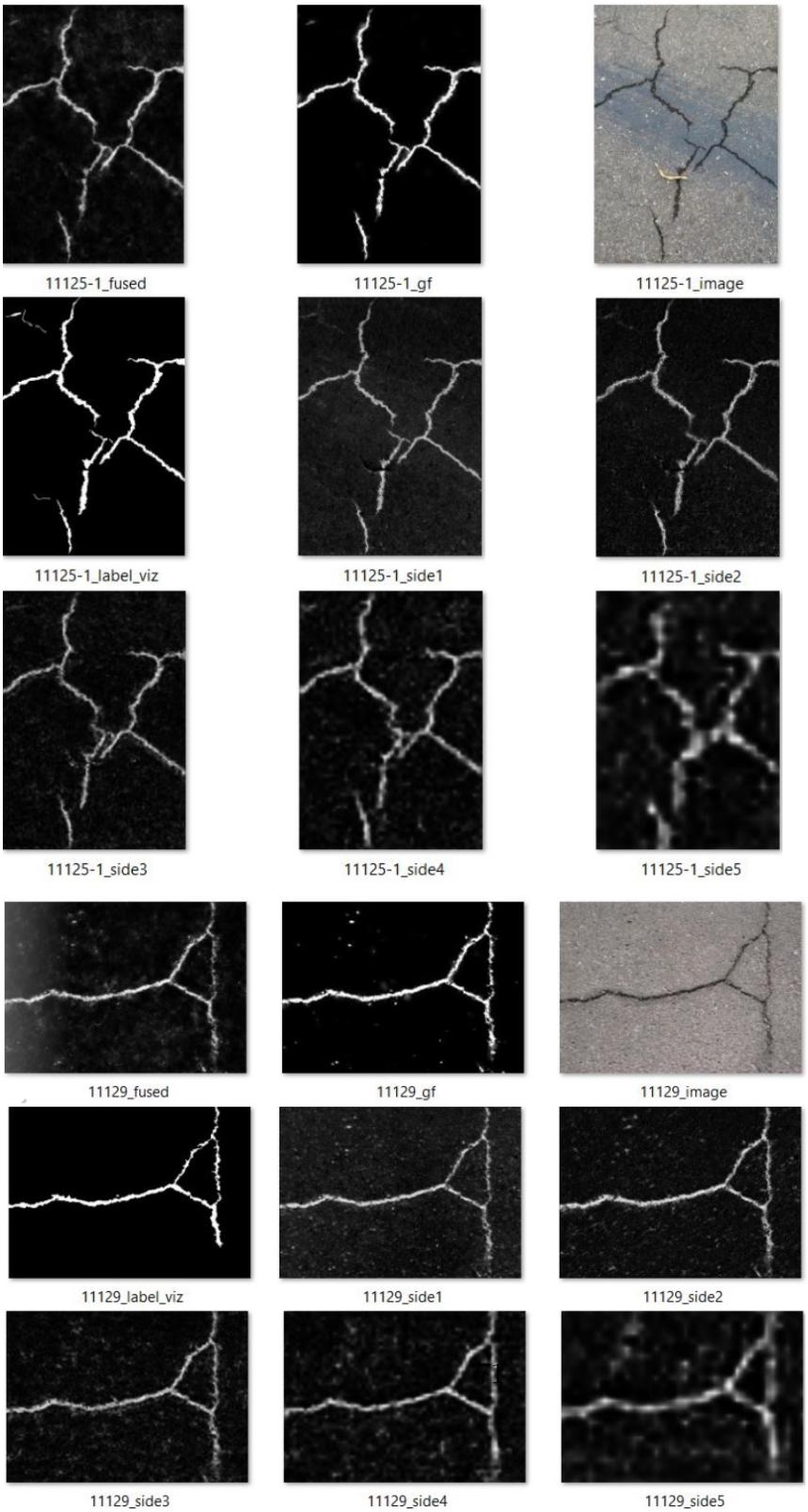
11174_side3



11174_side4

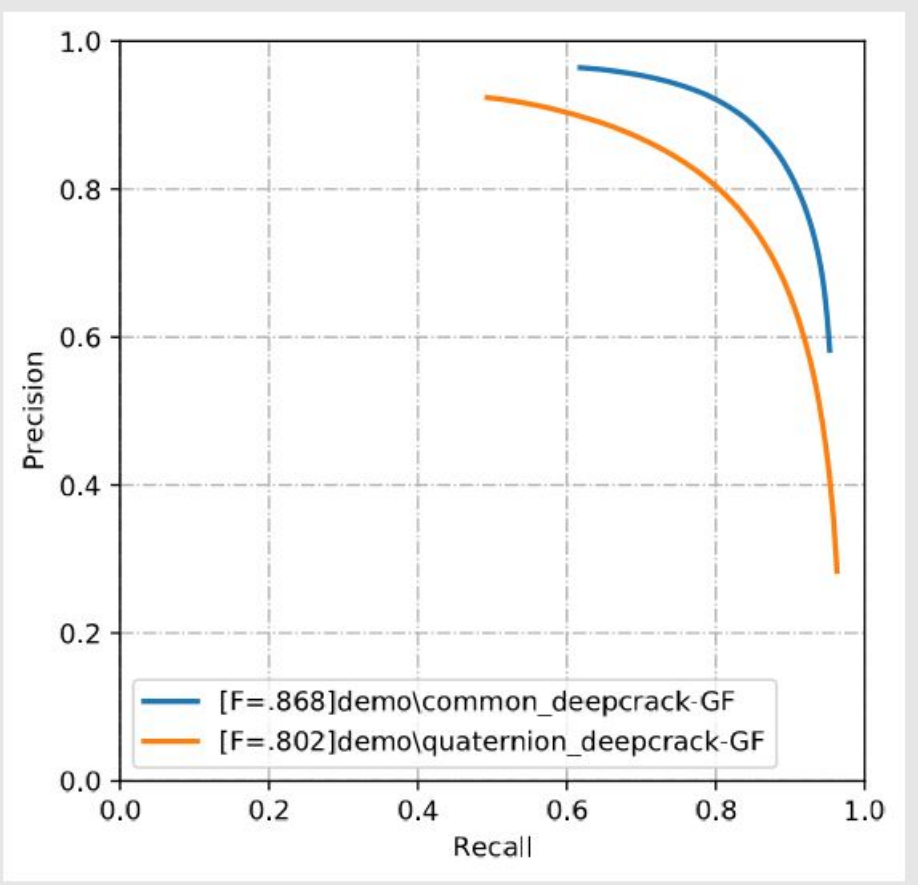
11174_side5

Figure 5.6: Quaternion DeepCrack Output



In the plot below we display the Precision, Recall and F-Score of the common and the quaternion DeepCrack respectively.

Figure 5.7: Precision-Recall Curve of Common-Quaternion DeepCrack



We can summarize the best metrics of the models as follows:

Outputs	bT	G	P	R	F
Quaternion DeepCrack GF	0.45	0.9808	0.7837	0.8212	0.802
Common DeepCrack GF	0.48	0.9888	0.8795	0.8575	0.8684
Common Fused	0.31	0.9873	0.8582	0.8456	0.8518
Common Side Output 1	0.42	0.9834	0.8122	0.8022	0.8071
Common Side Output 2	0.43	0.9864	0.8591	0.8199	0.8390
Common Side Output 3	0.36	0.9854	0.8334	0.8295	0.8315
Common Side Output 4	0.36	0.9823	0.7886	0.8077	0.7980
Common Side Output 5	0.38	0.9735	0.6646	0.7807	0.7180
Quaternion Fused	0.35	0.9804	0.7749	0.7716	0.7733
Quaternion Side Output 1	0.40	0.9791	0.7708	0.7353	0.7526
Quaternion Side Output 2	0.42	0.9818	0.7952	0.7806	0.7878
Quaternion Side Output 3	0.39	0.9807	0.7798	0.7725	0.7761
Quaternion Side Output 4	0.33	0.9724	0.6605	0.7451	0.7003
Quaternion Side Output 5	0.31	0.9541	0.4771	0.6367	0.5455

All metrics in both models are increasing gradually from the low level layers to the middle ones and decreasing again to the higher ones. As a result that the low level features are susceptible to noise. The features from deeper layers are more abstract and that means that it decreases false positives, however it increases false negatives. The fused output show better results by aggregating the features that are found in multiple levels. The guided filtering refinement increases the performance in both networks. We see that that the common model has better overall results, however the less parameter needed for the quaternion model to work makes it ideal for low cost devices.

Chapter 6

Conclusions

In this study we use quaternions in convolutional neural networks and try to compare their performance with the common ones. Quaternions are a good alternative to common convolutional neural networks, having the same and sometimes better performance even if they have significantly less features .
Concerning the models used our custom cnn had very few features but it gave good results in terms of accuracy and F-score.
In general our quaternion models gave better results than the common ones with smaller datasets. However the complexity of the operations in the quaternion domain has a toll in the computational time needed for the model, increasing the cost to about 21%. Also the parameters which the models were initialized were the same for all of them
One major improvement that could be done is to better modify the parameters of the models, so as to get better results and make the quaternion models less computational expensive and also augmentating the data so as to make it more diverse. As far as image segmentation and specifically Deepcrack is concerned the quaternion model reduces the number of parameters and has slightly worse performance. Possible future improvements are parameter tuning, techniques better merging of side output features and other sophisticated algorithm for filtering.

List of Figures

Bibliography

[1] Quaternion. https://en.wikipedia.org/wiki/Quaternion. Accessed: 2020-12-29.

[2] Thiru Balaji; M S Dinesh; Nair Binoy; D. S Harish Ram A, Jayanth Balaji; G. Asphalt crack dataset. 2019.

[3] Joseph C Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 1973.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[7] Yahui Liu, Jian Yao, Xiaohu Lu, Renping Xie, and Li Li. Deepcrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing*, 338:139–153, 2019.

[8] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. Chapter 10 - convolutional neural networks. In Andrea Mechelli and Sandra Vieira, editors, *Machine Learning*, pages 173 – 191. Academic Press, 2020.

[9] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[10] Ç F Özgenel and A Gönenç Sorguç. Performance comparison of pre-trained convolutional neural networks on crack detection in buildings. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 35, pages 1–8. IAARC Publications, 2018.

[11] Titouan Parcollet, Mohamed Morchid, and Georges Linarès. Quaternion convolutional neural networks for heterogeneous image processing. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8514–8518. IEEE, 2019.

[12] Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linarès, Chiheb Trabelsi, Renato De Mori, and Yoshua Bengio. Quaternion recurrent neural networks. In *International Conference on Learning Representations*, 2019.

[13] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[15] Giorgos Sfikas, Angelos P Giotis, George Retsinas, and Christophoros Nikou. Quaternion generative adversarial networks for inscription detection in byzantine monuments.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[19] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. 08 2017.

[20] Muhamad Yani, S Irawan, and M.T. S.T. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019.

[21] Çağlar Fırat Özgenel. Concrete crack images for classification. 2018.

[22] Lei Zhang, Fan Yang, Yimin Daniel Zhang, and Ying Julie Zhu. Road crack detection using deep convolutional neural network. In *2016 IEEE international conference on image processing (ICIP)*, pages 3708–3712. IEEE, 2016.

[23] Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. Quaternion convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–647, 2018.