Deep Quaternion Networks

Prospectus

by

Chase Gaudet

April 8, 2018

Supervisor:
Dr. Anthony Maida

University of Louisiana at Lafayette
Faculty of Computer Science

Abstract

The field of deep learning has seen significant advancement in recent years. However, much of the existing work has been focused on real-valued numbers. Recent work has shown that a deep learning system using the complex numbers can be deeper for a fixed parameter budget compared to its real-valued counterpart. In this work, we explore the benefits of generalizing one step further into the hyper-complex numbers, quaternions specifically, and provide the architecture components needed to build deep quaternion networks. We go over quaternion convolutions, present a quaternion weight initialization scheme, and present algorithms for quaternion batch-normalization. These pieces are tested in a classification model by end-to-end training on the CIFAR-10 and CIFAR-100 data sets and a segmentation model by end-to-end training on the KITTI Road Segmentation data set. The quaternion networks show improved convergence compared to real-valued and complex-valued networks, especially on the segmentation task.

Contents

Chapter 1

Introduction

1.1 Deep Learning

In recent years the area of artificial intelligence has seen massive growth. The majority of work has been around a fairly new area sub-field called Deep Learning (DL). Deep Learning is broadly described as Neural Networks composed of multiple layers. One type of DL model called Convolutional Neural Networks (CNNs) has achieved state-of-the-art results in almost all Computer Vision (CV) problems, as well as some outside of CV. The power of CNNs comes from automatically finding a set of highly expressive, hierarchical features. As we will discuss more in the CNN section, the way they work is closely related to the way the biological vision system works.

1.2 Beyond Real Numbers

Historically the majority of work done in DL, and even Neural Networks, has been using real-valued numbers. Recently there has been increased attention on using complex numbers. Complex numbers have several advantages from a computational, biological, and signal processing perspective [TBS+17]. One big motivation is that a complex number holds more information than a real number, complex-valued formulation captures amplitude and phase by definition. Biologically this can be thought of as the neurons firing rate and the relative timing of its activity.

Quaternions are an extension of the complex numbers. While the complex plane is 2D, the quaternion space is 4D. They offer some important benefits that over the complex numbers, but note that since a quaternion can be expressed as a 2×2 matrix of complex numbers that we are not giving anything up. We will show that the main advantages come from a quaternion being able to very compactly represent rotations in 3D space and, more importantly, from the algebraic structure inducing promising properties for CNNs.

1.3 Outline

In the following chapters we will fully cover CNNs and their latest developments as well as give the reader a primer on quaternions. This sets up necessary background knowledge for us to describe our main contributions, which include a novel quaternion weight initialization scheme and novel algorithms for quaternion batch-normalization. These pieces, along with quaternion convolutions, allow for the construction of a quaternion CNN (QCNN). We perform sanity checks on the QCNN by comparing its performance against real-valued and complex CNNs on common benchmark data sets in both classification and segmentation and find the QCNN outperforms both. With these promising results are goal is to explore QCNN performance on more demanding tasks, particularly ones we believe the algebraic structure of quaternions lends itself too.

Chapter 2

Introduction To Convolutional Neural Networks

The Convolutional Neural Network (CNN) is a well-known deep learning architecture that was loosely inspired by the natural visual perception mechanism of some living organisms. This comes from a paper by Hubel and Wiesel [HW68] in 1959 where they found that their were specific cells in the visual cortex of animals that were responsible for detecting light in the receptive field, which eventually won them a Nobel Prize. It was not until 1990 that LeCun et al. [LBD$^+$90] released the crucial paper that would establish modern framework for CNNs. Their model was a multiple layer neural network called LeNet-5 that could classify 28x28 greyscale hand written digit images with very high accuracy. As seen in Fig. 2.1 LeNet-5 has multiple layers and can be trained with the back-propagation algorithm [HN$^+$88], but used a low number of parameter values for its size. To achieve this LeNet-5 used two core components, which we will discuss in the next section.

2.1 Basic CNN Components

While there are many different layer types in current literature, here we will focus on the two introduced in LeNet-5, which are still used today along with

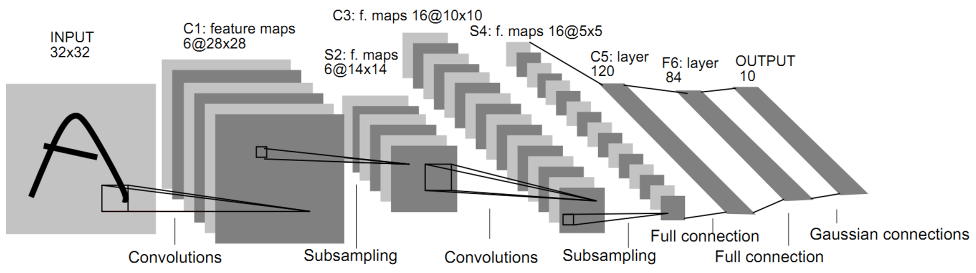


Figure 2.1: The architecture for LeNet-5, which is composed of repeating convolution and pooling layers.

some variants of them.

2.1.1 Convolutional Layer

The first is the convolution layer. Its purpose is to learn feature representations of the inputs. Each convolution layer has a specified number of kernels where each kernel is unique as to generate a distinct feature map. Each neuron of a feature map is connected to a region of neighboring neurons in the previous layer. The size of the connection is known as the kernel size or the receptive field size. The feature map is obtained by first convolving the input with the kernel and then applying an element-wise nonlinear activation function on the result. Because it is a convolution operation each kernel is shared with all spatial locations of the input. This spatial weight sharing is what creates the low parameter number compared to a regular neural network which fully connects all inputs and outputs. It also allows the CNN to take advantage of the underlying structure in images. Topological information, i.e., spatial information about the structure in an image, such as adjacency and rotations are also taken into account. The equation for finding the feature value at location (i, j) in the k^{th} feature map of the l^{th} layer, $z^l_{i,j,k}$ is

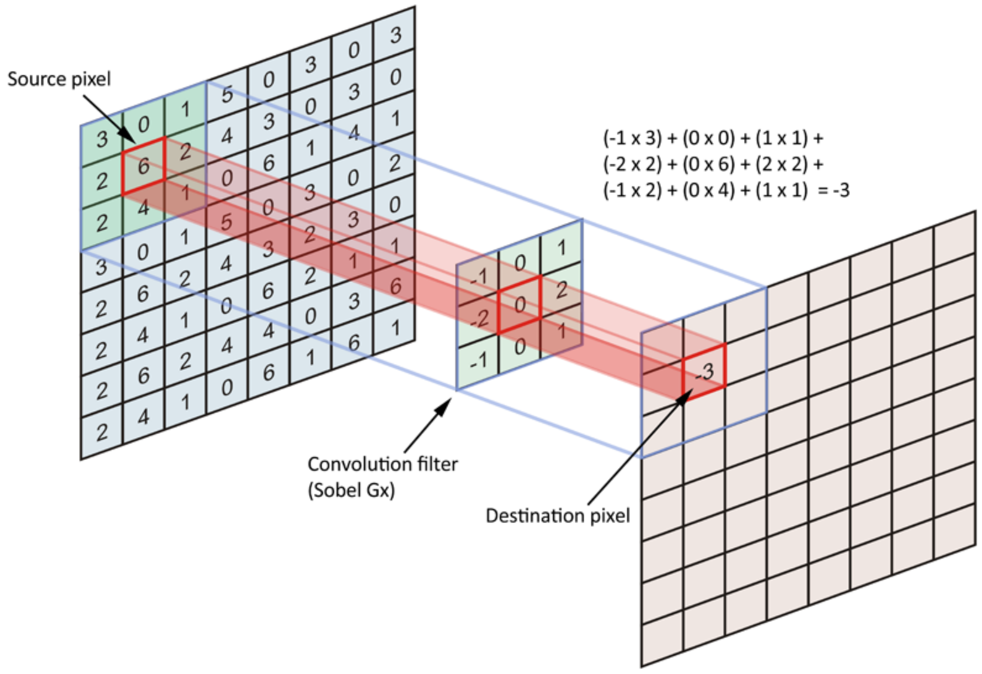
$$z^l_{i,j,k} = \mathbf{w}^{l\,T}_k \mathbf{x}^l_{i,j} + b^l_k \tag{2.1}$$

where $\mathbf{w}^l_k$ and b^l_k are the weight vector, or convolution kernel, and the bias term of the l^{th} layer respectively, and $\mathbf{x}^l_{i,j}$ is the input patch centered at location (i, j) of the l^{th} layer. An example of this operation can be seen in

Fig. 2.2. After the convolution is applied the result is then put through the nonlinear activation function:

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{2.2}$$

where $a(\cdot)$ is the nonlinear activation function. We will discuss different activation functions in great detail in Chapter 3.



Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.2: Convolution operation performed on a single feature location and one convolution kernel [App00].

2.1.2 Pooling Layer

The other key layer is the pooling layer shown in Fig.2.1 as subsampling. The pooling layer comes after a convolution layer and it performs a reduction in the resolution of the feature map. This layer works typically with two arguments: the spatial size of the pooling window F and the stride, or shift per operation, S. It is accomplished in a similar manner to convolution, starting at the top right a window of size $F \times F$ is grabbed. From this window the pooling function is used, which is commonly a **MAX** function. The window then moves over by S pixels and the above operation happens again. This is repeated until the whole image is covered. For example, if one chooses $F = 2$ and $S = 2$ they will be selecting the maximum pixel in a 2×2 window and moving 2 pixels over exactly as shown in Fig. 2.3. Note that this reduces the original image by a factor of $F = 2$. This serves the purpose of reducing the number of multiplications the architecture must perform and also aims to achieve shift invariance. We will discuss different pooling techniques in Chapter 3.

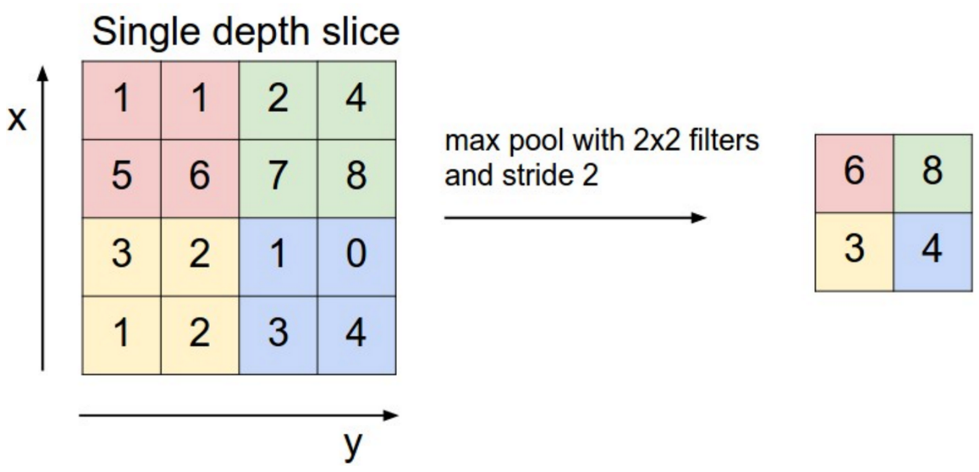


Figure 2.3: Example max pooling operation with a size of 2x2. A 2x2 window is moved over the input with a stride of 2 and the maximum value of the window is taken [Kar13].

There is also the inverse of pooling, often called upsampling, that will increase the resolution of the feature maps. There are several cases where this is a desired effect which we will discuss later.

2.1.3 CNN Common Tasks

The last layers of LeNet-5 are fully connected layers like that of a regular Feed-Forward Neural Network (FNN), sometimes called Multilayer Perceptrons [RHW85]. The second to last layer is sometimes called a decision layer as it takes the outputs from the feature mapping and combines them together. The last layer is a fully connected layer of size 10, which corresponds to the number of classes in that particular data set. The classes, which are digits numbered 0 through 9, are represented as a 'one hot' vector, meaning that if the class is the digit '0' the first element of the output vector should be 1 and the rest should be 0. This can be thought of as a probability distribution on the classes. This is an important representation because it allows one to use loss functions that minimizes the differences of distributions opening up many potential loss functions. This was the common use case for CNNs for a while, given an image predict a distribution over a set of classes. This task is known as *classification*.

Recently CNNs have seen use in another area known as *semantic segmentation*. Semantic segmentation, also sometimes just called segmentation, is the process of partitioning an image into multiple segments, or sets of pixels. More precisely, segmentation is the process of assigning a class from our classification set to each pixel, or potentially area of pixels, in an image. Unlike classification, the output of this CNN would typically be in the form of an image. Typically the network will output a distribution over a set of classes *per pixel* and then the pixel is labeled with the class with the highest probability. There are many applications where image classification alone does not giving enough information and one needs pixel-level labels. Examples include: detecting road signs [MBLAGJ+07], detecting tumors in medical imaging [LJH15, LPAL15, KPU15, HDWF+17], detecting objects of interest from satellite photos [CXLP13], finding pedestrians [DEKLD16], and many more. Having pixel level labels allows multiple objects of different classes to be detected in one image compared to image level labels where there is a single output per image.

In recent years, CNNs have obtained state of the art results on almost all classification and segmentation data sets. The improvements in CNNs have come from a few different areas including better computers, improvements to initialization of network weights, and specific architectures to combat problem areas of deep networks. In this next chapter we will go through a list of improvements and key papers.

Chapter 3

State of the Art

In this chapter we will go over the latest improvements for CNNs. They range from the activation function used within each neuron to major architecture changes to combat problems that arise from training deeper and deeper networks.

3.1 Activation Functions

The current trend has been using activation functions that are non-saturated, meaning they do not have very small derivatives at large input values. Also, these new functions are chosen to help combat the 'vanishing gradient problem' [HBF$^+$01]. This problem was due to many of the originally used activation functions (e.g sigmoid or tanh) 'squash' their input into a very small output range in a very non-linear fashion. For example, sigmoid maps the real number line onto a 'small' range of $[0, 1]$. As a result, there are large regions of the input space which are mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output - hence the gradient is small. This becomes much worse when we stack multiple layers of such non-linearities on top of each other, leading to difficulty in training deeper networks. We will go through a few new activation functions, whose plots are shown in Fig. 3.1.

3.1.1 ReLU:

The most notable non-saturated activation function is the rectified linear unit (ReLU) [NH10]. This is defined as:

$$f(x) = \max(0, x). \tag{3.1}$$

The benefits of this unit include: faster calculation since the $\max(\cdot)$ operation is faster than sigmoid or tanh, it creates sparsity in the hidden units by giving true zero values often to help sparse representations form, and does not suffer from vanishing gradients in deep models. ReLU has been shown to work better than sigmoid or tanh in several tasks and shows fast convergence even without pretraining [GBB11, KSH12, ZRM+13, MHN13].

3.1.2 LReLU:

The ReLU has zero gradient whenever its inputs add to a negative value or when a large gradient changes the weights to large negatives value making future input into the unit always have a negative value. These units will never be updated and learning will not occur, which can be a problem. Mass *et al.* [MHN13] found a solution to this by introducing a non zero component to the ReLU when the input is negative. They called this unit the Leaky ReLU as it 'leaks' a positive gradient on the negative side of its graph by having a very small slope. This leaky ReLU or LReLU is defined as:

$$f(x) = \max(0, x) + \lambda \min(0, x) \tag{3.2}$$

where $\lambda \in (0, 1)$ and is predefined per model. This unit allows for small gradients when the unit is not active, which helps the problems of the ReLU unit.

3.1.3 PReLU:

Rather than trying to find an optimal value for λ in the LReLU, He et al. [HZRS15b] introduced the Parametric ReLU which adapts the λ during training. It is defined as:

$$f(x) = \max(0, x) + \lambda_k \min(0, x) \tag{3.3}$$

where λ_k is the k-th channel. Since the number of parameters in networks is often very large compared to the number of total channels, the extra computational cost to learn the values of λ_k's is not much.

3.1.4 ELU:

Next is the Exponential Linear Unit (ELU) [CUH15]. ELUs are similar to the above mentioned units, but they have a saturating function on their negative side. The saturation function decreases the variation of the units if they are not activated, which gives those units a chance to update while making them more robust to noise. The ELU function is defined as:

$$f(x) = \max(0, x) + \min(0, \lambda(e^x - 1)) \tag{3.4}$$

where λ is predefined as in the LReLU.

3.1.5 PELU:

A natural extension of the ELU is the Parametric ELU [TGCd16]. Unlike the PReLU which learns one parameter to modify the LReLU, the PELU learns two parameters during training to modify the ELU. This gives the network more control over the vanishing gradients. The PELU is defined as:

$$f(x) = \max(0, \frac{a}{b}x) + \min(0, a(e^{x/b} - 1)) \tag{3.5}$$

where $a, b > 0$. Both a and b change the slope of the linear function together, b effects the scale of the exponential decay, and a is the saturation point in the negative side of the function. In [TGCd16] it is shown that PELU does a better job than all the above functions in several different models and data sets.

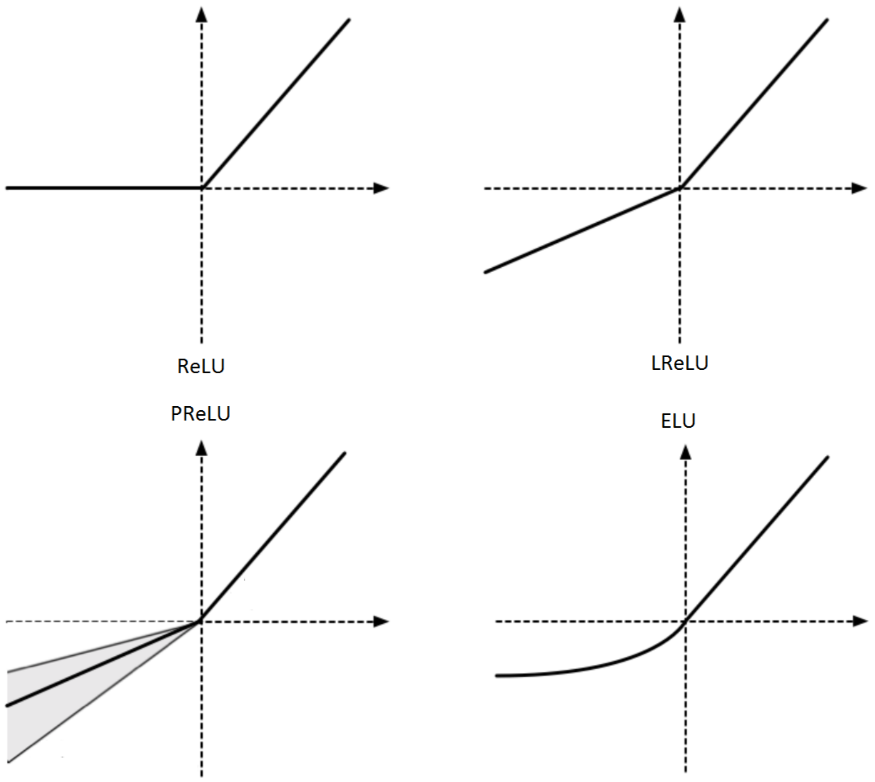


Figure 3.1: Examples of some of the discussed activation functions. From top to bottom and left to right: ReLU, LReLU, PReLU, and ELU.

3.2 Weight Initialization

The parameters of a network are randomly initialized because if they were not, there would exist symmetry among the parameters and nothing useful would be learned. The deeper and more complex the network, the more parameters one is attempting to optimize. Each layer of the network will scale its input by some constant k, meaning the final layer will scale the original input by L^k where L is the number of layers. It is clear that for deep network that values $k > 1$ will have very large outputs, while values $k < 1$ lead to outputs to diminish to zero. Because of this determining an optimal way to initialize these parameters was vital to the network converging to a good set of weights. There are two commonly used schemes we will mention in this chapter, Glorot and Bengio [GB10] and a modification to that by He et al. [HZRS15b].

3.2.1 Glorot Initialization

A big assumption in the derivation of this initialization scheme is that the neurons are linear functions. Let X be the input to n linear neurons with weights W that produce output Y

$$Y = W_1X_1 + W_2X_2 + \ ... \ + W_nX_n.$$

To find the variance of Y first observe that

$$\mathrm{Var}(W_iX_i) = \mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

and if assumed that X_i and W_i are all independently and identically distributed one can write Y as

$$\mathrm{Var}(Y) = \mathrm{Var}(W_1X_1 + W_2X_2 + \ ... \ + W_n + X_n) = n\mathrm{Var}(W_i)\mathrm{Var}(X_i).$$

This tells us under these assumptions that the variance of the input is the variance of the input scaled by $n\mathrm{Var}(W_i)$, so if we want the variance input and output to be the same the variance of the weights should be

$$\mathrm{Var}(W_i) = \frac{1}{n_{in}}$$

where we introduce the subscript on n to denote that this is the count of neurons going into the layer. The reason for that is because the above must also be done for the backpropagated pass where the same formula is found except n is the number of neurons going out of the layer

$$\mathrm{Var}(W_i) = \frac{1}{n_{out}}.$$

These two constrains can only be satisfied if $n_in = n_out$, so Glorot and Bengio take the average

$$\mathrm{Var}(W_i) = \frac{2}{n_{in} + n_{out}}. \tag{3.6}$$

In practice it turns out that the backpropagation pass is less important to preserve compared to the forward pass. Because of this and the fact that finding n_{out} is expensive to find in most software implementations the Glorot initialization is often shortened to

$$\mathrm{Var}(W_i) = \frac{1}{n_{in}}. \tag{3.7}$$

3.2.2 He Initialization

He et al. set out to define a good initialization solely for the ReLU activation function. They build on Glorot and Bengio's work by noting that the ReLU is almost a linear activation function, but is zero for half of its input, so you need to double the size of weight variance to keep the signals variance constant. This leads to

$$\mathrm{Var}(W_i) = \frac{2}{n_{in}}. \tag{3.8}$$

It is of note that using this initialization scheme they were the first to surpass human level performance on the ImageNet 2012 classification challenge.

3.3 Regularization

Some of the biggest improvements to NNs have been through regularizing the network in different ways to prevent overfitting. Overfitting means that the network learns the training set well, but performs poorly on any new or unseen data.

3.3.1 Dropout

Introduced by Hinton *et al.* [HSK+12], Dropout has shown to be very effective at improving network results and reducing overfitting. In their paper Dropout is applied to fully-connected layers where the output of the Dropout layer is

$$\mathbf{y} = \mathbf{r} * a(\mathbf{W}^T\mathbf{x}), \tag{3.9}$$

where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ is the input to the fully-connected layer, $a(\cdot)$ is an activation function, $\mathbf{W}$ is the weight matrix, $*$ is an element-wise multiplication, and $\mathbf{r}$ is a binary vector whose elements are independently drawn from a Bernoulli distribution with parameter p. This means that at every update to the network, each neuron has a p chance of getting multiplied by zero. Dropout usually prevents the network from becoming too reliant on a few neurons and helps the network generalize by spreading feature information. Some extensions to Dropout have been proposed like in [WM13] where a method to improve the speed of training while using Dropout are introduced. Similar to the learned parameters of the activation functions, [BF13] learn the p in the Dropout distribution adaptively. Finally, in [TGJ$^+$15] they modify Dropout with CNNs specifically in mind by extending the Dropout value across the entire feature map, meaning that during training each feature map has a p chance of being completely ignored for each update step. They call this method Spatial Dropout and it seems to help the network learn more general features, which improves results on limited size data sets.

3.3.2 Batch Normalization

It is standard practice to normalize data to have zero-mean and unit variance before feeding it into a NN, but as the data goes through the network, especially deep networks, the data will lose this property. This change to the input distribution is known as internal covariance shift. To keep data normed through the network [IS15] introduced an efficient method called Batch Normalization (BN). BN works by normalizing the mean and variance of each layers input using each batch rather than the entire training set. To demonstrate BN, let $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ be a n dimensional input to a layer. The k-th dimensions is normalized by:

$$\hat{x}_k = (x_k - \mu_B)/\sqrt{\sigma_B^2 + \epsilon} \tag{3.10}$$

where μ_B and σ_B^2 are the mean and variance of the batch respectively and ϵ is some small constant value that will guarantee the root term is always defined. Finally the input $\hat{x}_k$ is further transformed:

$$y_k = \mathrm{BN}_{\alpha,\beta}(x_k) = \alpha\hat{x}_k + \beta \tag{3.11}$$

where α and β are parameters learned during training. There are many benefits to using BN. By reducing internal covariance shift the network will converge faster. By making sure no inputs get too large or small BN makes it possible to use saturating activation functions without fear of getting stuck in a saturating section and having vanishing gradients.

3.4 Architecture Improvements

There are a couple of recent architecture improvements to NNs that both help with speed (by lowering computational cost for similar results) and immensely with vanishing gradients.

3.4.1 Network-In-Network

The first was introduced in [LCY13] where they use fully connected layers on the outputs of each convolution operation called Network-in-network. The idea was an alternative to stacking multiple convolutional layers (each with a number of their own filters) to get deepness in a neural network model, by replacing each filter with a multi-layer perceptron, which is essentially a small neural network that slides across the image like a convolutional filter. The math works out so that a $1 \times 1 \times U$ convolution filter convolved across a V-channel image emulates a $U \times V$ matrix multiplied by each V-channel pixel, which is the same as running a single-layer neural network across every pixel of your input as if each pixel were an example vector in a training set. Chaining together such $1 \times 1 \times U$ convolutions, you get the same result as if running a many-layered neural network at each input pixel of V channels. The idea from the paper was to turn a convolution filter which is a generalized linear model, into a non-linear model. It allows the network to combine channels from previous layer in a non-linear fashion, which can lead to more advanced features.

3.4.2 Inception Blocks

Google's team was inspired by the Network-in-network paper and saw the power of using the 1x1 convolutions not only for non-linearity, but to reduce computational burden of deep models by using the cross-channel pooling aspect to reduce their feature maps before convolution layers. In [SLJ+15] Christian Szegedy and his team introduced the Inception module as seen in Fig.3.2.

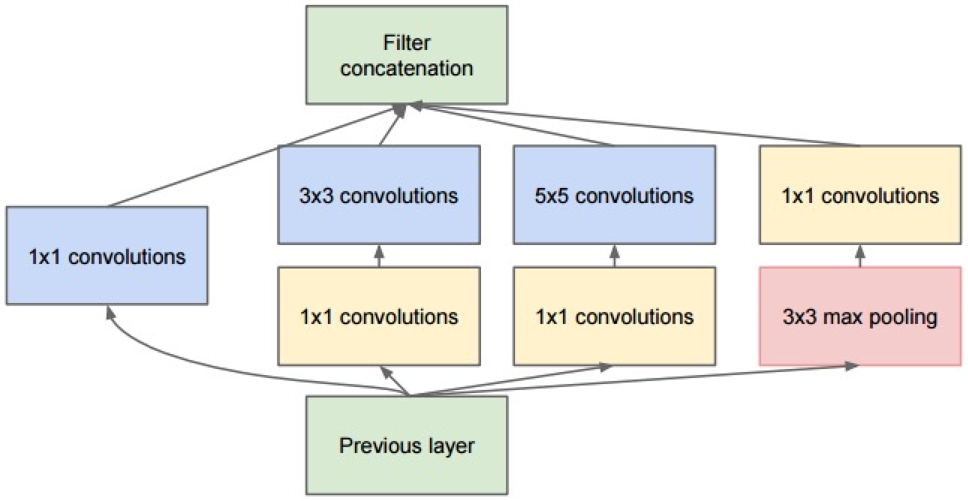


Figure 3.2: An inception block from [SLJ+15]. Notice the 1x1 convolutions before the 3x3 and 5x5 convolutions are reducing the number of feature channels. This reduces the number of multiplications that must be performed.

3.4.3 Residual Connections

Another work that focused on fixing the vanishing gradient problem of deep networks was Residual Nets (ResNets) [HZRS15a] where they used what is now called shortcut connections. These connections were inspired by Long Short Term Memory (LSTM) units, a type of recurrent neural network unit that feeds information back into itself with weighted gates. Instead of using weighted gates shortcut connections pass information with the identify function so it is untransformed. This means that the activation of deep units can be written as the sum of the activation of some shallower unit and a residual function, which is a series of network layers. Or to put simply, the input to a layer group is added to the output of that layer group. This is called a Residual Block and can be seen in Fig.3.3. By stacking these residual blocks together one gets a fully residual model. This design allows the gradients to be directly propagated to shallower units allowing for networks of 100s of layers to be trained.

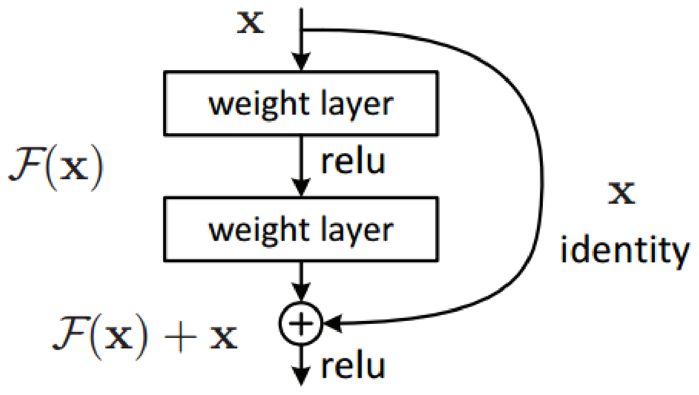

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.3: A residual block from [HZRS15a].

3.5 Conclusion

In this chapter we covered many advanced concepts of CNNs that have led to large improvements in their performance. The big areas that have seen improvement are the activation functions, weight initializations, regularization, and overall architecture improvements. There are other areas to possibly find improvements however. At present, the vast majority of all work done in neural networks is based on real-valued operations and representations. There have been some recent work exploring the use of complex numbers and very few the use of hyper-complex number (quaternions) showing promising results due to these numbers having greater representational power to the reals. In [TBS+17] they present the first complex forms of complex batch-normalization and complex weight initialization strategies to use along with complex convolution to create a fully complex deep convolutional network. They achieve some state-of-the-art results over the real-valued counter part models, which leads to the question of creating forms for quaternion batch-normalzation and weight initialization for potentially better results. In the next chapter we will give the reader a basic foundation of the quaternion algebra to build up to the theory of these quaternion network layers.

Chapter 4

Quaternions

This chapter presents a primer on quaternions so the reader can understand the theory developed in the research section. The quaternions are a number system that extends the complex numbers. They were first described by Irish mathematician William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space. In modern mathematical language, quaternions form a four-dimensional associative normed division algebra over the real numbers, and therefore also a domain. In this chapter we will define operations on the quaternion algebra, draw connection to the complex numbers and $\mathbb{R}^3$, and show some real world use cases of quaternions.

4.1 Quaternion Algebra

In 1833 Hamilton proposed that complex numbers $\mathbb{C}$ be defined as the set $\mathbb{R}^2$ of ordered pairs (a, b) of real numbers. He then began working to see if triplets (a, b, c) could extend multiplication of complex numbers. In 1843 he discovered a way to multiply in four dimensions instead of three, but the multiplication lost commutativity. This construction is now known as quaternions. Quaternions are composed of four components, one real part, and three imaginary parts. Typically denoted as

$$\mathbb{H} = \{a + bi + cj + dk \ : \ a, b, c, d \in \mathbb{R}\} \tag{4.1}$$

where a is the real part, (i, j, k) denotes the three imaginary axis, and (b, c, d) denotes the three imaginary components. Sometimes a is referred to as the scalar part and (b, c, d) as the vector part. Note that we shall use the forms $bi + cj + dk$ and (b, c, d) interchangeably. Quaternions are governed by the following arithmetic:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{4.2}$$

which leads to the non-commutative multiplication rules

$$ij = k,\ jk = i,\ ki = j,\ ji = -k,\ kj = -i,\ ik = -j \tag{4.3}$$

4.1.1 Addition and Multiplication

The addition of two quaternions acts component wise, exactly the same as two complex numbers. Consider the quaternion q

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{4.4}$$

and the quaternion p

$$p = p_0 + p_1 i + p_2 j + p_3 k. \tag{4.5}$$

Their addition is given by

$$p + q = (p_0 + q_0) + (p_1 + q_1)i + (p_2 + q_2)j + (p_3 + q_3)k. \tag{4.6}$$

The product of two quaternions will produce another quaternion and is given by

$$\begin{aligned} pq &= (p_0 + p_1 i + p_2 j + p_3 k)(q_0 + q_1 i + q_2 j + q_3 k) \\ &= p_0 q_0 - (p_1 q_1 + p_2 q_2 + p_3 q_3) \\ &\quad + p_0(q_1 i + q_2 j + q_3 k) + q_0(p_1 i + p_2 j + p_3 k) \\ &\quad + (p_2 q_3 - p_3 q_2)i + (p_3 q_1 - p_1 q_3)j + (p_1 q_2 p - p_2 q_1)k. \end{aligned}$$

It can be seen that the result is a quaternion by refactoring it, which we do by utilizing the inner and cross products of two vectors in $\mathbb{R}^3$ [Hun80]:

$$pq = p_0 q_0 - \mathbf{p} \cdot \mathbf{q} + p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q} \tag{4.7}$$

where $\mathbf{p} = (p_1, p_2, p_3)$ and $\mathbf{q} = (q_1, q_2, q_3)$ are the vector parts of p and q respectively. The first two terms form the scalar part and the last three form the vector part.

4.1.2 Conjugate, Norm, and Inverse

The *conjugate* of q, denoted as q^*, is defined as a negation of the vector part of q

$$q^* = q_0 - \mathbf{q} \tag{4.8}$$

and has the properties

$$\begin{aligned}
(q^*)^* &= q_0 - (-\mathbf{q}) = q, \\
q + q^* &= 2q_0, \\
q^* q &= (q_0 - \mathbf{q})(q_0 + \mathbf{q}) \\
&= q_0 q_0 - (-\mathbf{q}) \cdot \mathbf{q} + q_0 \mathbf{q} + (-\mathbf{q}) q_0 + (-\mathbf{q}) \times \mathbf{q} \\
&= q_0^2 + \mathbf{q} \cdot \mathbf{q} \\
&= q_0^2 + q_1^2 + q_2^2 + q_3^2 \\
&= q q^*
\end{aligned}$$

The *norm* of a quaternion q, denoted as $|q|$, is the scalar

$$|q| = \sqrt{q^* q}$$

and a quaternion is said to be a *unit quaternion* if its norm is 1. For instance let $\hat{\mathbf{u}} = u_1 i + u_2 j + u_3 k$ be any unit vector, then a unit quaternion is given by

$$q_{unit} = \cos\theta + \hat{\mathbf{u}} \sin\theta.$$

One can check by plugging this into (4.1.2)

$$\begin{aligned}
|q_{unit}| &= \sqrt{q_{unit}^* q_{unit}} \\
&= \sqrt{\cos^2\theta + u_1 \sin^2\theta + u_2 \sin^2\theta + u_3 \sin^2\theta} \\
&= \sqrt{\cos^2\theta + \sin^2\theta} \\
&= 1.
\end{aligned}$$

The norm of the product of two quaternions p and q is the product of the individual norms,

$$\begin{aligned}|pq|^2 &= (pq)(pq)^* \\ &= pqq^*p^* \\ &= p|q|^2p^* \\ &= pp^*|q|^* \\ &= |p|^*|q|^*.\end{aligned}$$

The *inverse* of a quaternion q is defined as

$$q^{-1} = \frac{q^*}{|q|^2},$$

which gives $q^{-1}q = qq^{-1} = 1$. For all unit quaternions, the inverse is equal to the conjugate.

4.2 Geometric Representation of Quaternions

Here we will try to show a more intuitive geometric interpretation of quaternions by showing their link to complex numbers and how they can be used to represent rotations. To do this we will first give a brief reminder of complex numbers and their geometric interpretation of rotating a 2D plane.

4.2.1 Complex Algebra

Complex numbers are composed of two components, one real part, and one imaginary part. This is usually denoted as

$$\mathbb{C} = \{a + bi \ : \ a, b \in \mathbb{R}, \quad i^2 = -1\} \tag{4.9}$$

where a is the real part, i denotes the single imaginary axis, and b denotes the single imaginary component.

Let c and d be two complex numbers, their addition is given by

$$c + d = (c_0 + c_1 i) + (d_0 + d_1 i) = (c_0 + d_0) + (c_1 + d_1)i \tag{4.10}$$

and their multiplication by

$$cd = (c_0d_0 - c_1d_1) + (c_0d_1 + c_1d_0)i. \tag{4.11}$$

Any complex number has a length given by

$$|c| = |c_0 + c_1i| = \sqrt{c_0^2 + c_1^2} \tag{4.12}$$

and like quaternions, any complex number with a length of 1 is called a *unit complex number*.

4.2.2 Complex Rotation Operation

The set of unit complex numbers lies on the unit circle in $\mathbb{C}$ and Leonhard Euler showed that

$$e^{i\theta} = \cos\theta + i\sin\theta. \tag{4.13}$$

If we multiply this by any positive number r, we get a complex number of length r. Therefore, by adjusting the length r and the angle θ, we can write any complex number. This form goes by the name *polar coordinates*.

They are a great way to multiply complex numbers. Instead of (4.11) let us write each complex in polar coordinates

$$c = (c_0 + c_1i) = re^{i\theta}, \quad d = (d_0 + d_1i) = se^{i\phi}$$

and then multiply

$$cd = re^{i\theta}se^{i\phi} = rse^{i(\theta+\phi)}.$$

This tells us that to multiply two complex numbers, multiply their lengths and add their angles. In particular, if we multiply a given complex number by a unit complex number the resulting length is the same, but we have rotated it by θ degrees. This gives us some hints that quaternions may also be able to perform rotation operation in higher dimensional space.

4.2.3 Quaternion Rotation Operation

We will stick to interpretations of quaternions in $\mathbb{R}^3$ as it is easier to visualize, but how can a quaternion, which exists in $\mathbb{R}^4$, operate on a 3D vector? Recall that the imaginary components of a quaternion are called the vector part and a vector $\mathbf{v} \in \mathbb{R}^3$ is a *pure quaternion* whose real part is zero.

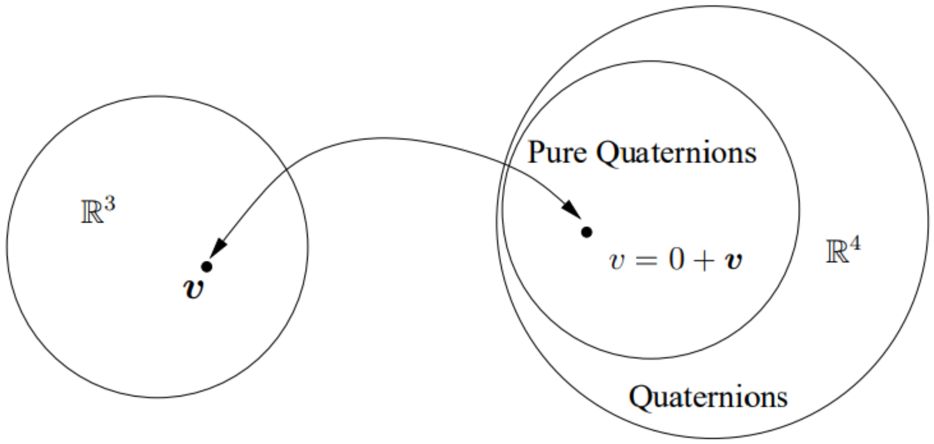


Figure 4.1: $\mathbb{R}^3$ can be viewed as a subspace of quaternions called pure quaternions which have a real part of zero. [Jia08]

Using the unit quaternion q let us define a function $L_q : \mathbb{R}^3 \to \mathbb{R}^3$ on vectors $\mathbf{v} \in \mathbb{R}^3$ [Hun80]:

$$\begin{aligned} L_q(\mathbf{v}) &= q\mathbf{v}q^* \\ &= (q_0^2 - ||\mathbf{q}||^2)\mathbf{v} + 2(\mathbf{q}\cdot\mathbf{v})\mathbf{q} + 2q_0(\mathbf{q}\times\mathbf{v}). \end{aligned} \tag{4.14}$$

Two observations to note are that first, the operation (4.14) does not modify the length of the vector $\mathbf{v}$:

$$\begin{aligned} ||L_q(\mathbf{v})|| &= ||q\mathbf{v}q^*|| \\ &= |q| \cdot ||\mathbf{v}|| \cdot |q^*| \\ &= ||\mathbf{v}||. \end{aligned}$$

And second, the direction of $\mathbf{v}$, if along $\mathbf{q}$, is left unchanged by the function. To verify let $\mathbf{v} = k\mathbf{q}$

$$\begin{aligned} L_q(\mathbf{v}) &= q(k\mathbf{q})q^* \\ &= (q_0^2 - ||\mathbf{q}||^2)k\mathbf{q} + 2(\mathbf{q}\cdot k\mathbf{q})\mathbf{q} + 2q_0(\mathbf{q}\times k\mathbf{q}) \\ &= k(q_0^2 + ||\mathbf{q}||^2)\mathbf{q} \\ &= k\mathbf{q}. \end{aligned}$$

Using our insights from complex numbers this lets us guess that the function (4.14) acts like a rotation about $\mathbf{q}$.

Theorem 1. *For any unit quaternion*

$$q = q_0 + \boldsymbol{q} = cos\frac{\theta}{2} + \hat{\mathbf{u}}sin\frac{\theta}{2}, \tag{4.15}$$

where $\hat{\mathbf{u}} = u_1 i + u_2 j + u_3 k$ *is any unit vector and for any vector* $\boldsymbol{v} \in \mathbb{R}^3$ *the result of the function*

$$L_q(\boldsymbol{v}) = q\boldsymbol{v}q^*$$

on $\boldsymbol{v}$ *is equivalent to a rotation of the vector through an angle* θ *about* $\hat{\mathbf{u}}$ *as the axis of rotation.*

Proof. Given a vector $\mathbf{v} \in \mathbb{R}^3$, we decompose it as $\mathbf{v} = \mathbf{a} + \mathbf{n}$, where $\mathbf{a}$ is the component along the vector $\mathbf{q}$ and $\mathbf{n}$ is the component normal to $\mathbf{q}$. Then we show that under the function L_q, $\mathbf{a}$ is invariant, while $\mathbf{n}$ is rotated about $\mathbf{q}$ through an angle θ.

Earlier we showed that $\mathbf{a}$ is invariant under L_q so let us see how L_q transforms $\mathbf{n}$.

$$\begin{aligned} L_q(\mathbf{n}) &= (q_0^2 - ||\mathbf{q}||^2)\mathbf{n} + 2(\mathbf{q}\cdot\mathbf{n})\mathbf{q} + 2q_0(\mathbf{q}\times\mathbf{n}) \\ &= (q_0^2 - ||\mathbf{q}||^2)\mathbf{n} + 2q_0(\mathbf{q}\times\mathbf{n}) \\ &= (q_0^2 - ||\mathbf{q}||^2)\mathbf{n} + 2q_0||\mathbf{q}||(\hat{\mathbf{u}}\times\mathbf{n}), \end{aligned}$$

where in the last step we introduced $\hat{\mathbf{u}} = \mathbf{q}/||\mathbf{q}||$. Let $\mathbf{n}_\perp = \hat{\mathbf{u}} \times \mathbf{n}$ to get

$$L_q(\mathbf{n}) = (q_0^2 - ||\mathbf{q}||^2)\mathbf{n} + 2q_0||\mathbf{q}||\mathbf{n}_\perp. \tag{4.16}$$

Also note that $\mathbf{n}_\perp$ and $\mathbf{n}$ have the same length:

$$||\mathbf{n}_\perp|| = ||\mathbf{n}\times\hat{\mathbf{u}}|| = ||\mathbf{n}||\cdot||\hat{\mathbf{u}}||\text{sin}\frac{\pi}{2} = ||\mathbf{n}||.$$

Then rewriting (4.16) we arrive at

$$\begin{aligned} L_q(\mathbf{n}) &= \left(\text{cos}^2\frac{\theta}{2} - \text{sin}^2\frac{\theta}{2}\right)\mathbf{n} + \left(2\text{cos}\frac{\theta}{2}\text{sin}\frac{\theta}{2}\right)\mathbf{n}_\perp \\ &= \cos\,\theta\mathbf{n} + \sin\,\theta\mathbf{n}_\perp. \end{aligned}$$

The resulting vector is a rotation of $\mathbf{n}$ through an angle θ in the plane defined by $\mathbf{n}$ and $\mathbf{n}_\perp$ [Jia08]. □

4.3 Conclusion

In this chapter we gave the reader the needed information of the quaternion algebra for theory sections to follow. We have discussed some motivation for the representational power of quaternions by showing how they can encode rotations in a compact form. It was also hinted that the structure of quaternion algebra benefits CNNs specifically. This comes from the multiplication rule for quaternions, which makes the convolution operation in the CNN treat the channels of the image as a single entity. We will show this in detail in the next chapter within the quaternion convolution Section 5.2.3.

Chapter 5

Quaternion Networks

At this point we have introduced the CNNs and their more recent advances as well as the basic concepts of quaternions. In this chapter we will go through the motivation for constructing a neural network with quaternion values. With the motivation in place we will then go through the known quaternion convolution operation before moving to our novel contributions of quaternion weight initialization and quaternion batch-normalization. These pieces are then used to run some benchmark experiments and compared against real and complex valued networks. The quaternion network outperforms both on two classification tasks and a segmentation task.

5.1 Motivation and Related Work

The ability of quaternions to effectively represent spatial transformations and analyze multi-dimensional signals makes them promising for applications in artificial intelligence.

One common use of quaternions is for representing rotation into a more compact form. PoseNet [KGC15] used a quaternion as the target output in their model where the goal was to recover the $6-$DOF camera pose from a single RGB image. The ability to encode rotations may make a quaternion network more robust to rotational variance.

Quaternion representation has also been used in signal processing. The amount of information in the phase of an image has been shown to be sufficient to recover the majority of information encoded in its magnitude by Oppenheim and Lin [OL81]. The phase also encodes information such as shapes, edges, and orientations. Quaternions can be represented as a 2 x 2 matrix of complex numbers, which gives them a group of phases potentially holding more information compared to a single phase.

Bulow and Sommer [BS01] used the higher complexity representation of quaternions by extending Gabor's complex signal to a quaternion one which was then used for texture segmentation. Another use of quaternion filters is shown in [SE00] where they introduce a new class of filter based on convolution with hyper-complex masks, and present three color edge detecting filters. These filters rely on a three-space rotation about the grey line of RGB space and when applied to a color image produce an almost greyscale image with color edges where the original image had a sharp change of color. More quaternion filter use is shown in [SF07] where they show that it is effective in the context of segmenting color images into regions of similar color texture. They state the advantage of using quaternion arithmetic is that a color can be represented and analyzed as a single entity (by assigning each color channel to an imaginary axis). This comes from the structure of quaternion multiplication that was mentioned in Section 4.3, which we will show holds for quaternion convolution in a convolutional neural network architecture as well in Section 5.2.3.

A quaternionic extension of a feed forward neural network, for processing multi-dimensional signals, is shown in [MINM17]. They expect that quaternion neurons operate on multi-dimensional signals as single entities, rather than real-valued neurons that deal with each element of signals independently. A convolutional neural network (CNN) should be able to learn a powerful set of quaternion filters for more impressive tasks.

Another large motivation is discussed in [TBS$^+$17], which is that complex numbers are more efficient and provide more robust memory mechanisms compared to the reals [Bül99, SE00, BS01]. They continue that residual networks have a similar architecture to associative memories since the residual shortcut paths compute their residual and then sum it into the memory provided by the identity connection. Again, given that quaternions can be represented as a complex group, they may provide an even more efficient and robust memory mechanisms.

5.2 Quaternion Network Components

This section will include the work done to obtain a working deep quaternion network. Some of the longer derivations are given in the Appendix.

5.2.1 Quaternion Representation

Recall the quaternion algebra presented in 4.1. Since we will be performing quaternion arithmetic using reals it is useful to embed $\mathbb{H}$ into a real-valued representation. There exists an injective homomorphism from $\mathbb{H}$ to the matrix ring $M(4,\mathbb{R})$ where $M(4,\mathbb{R})$ is a 4x4 real matrix. The 4 x 4 matrix can be written as

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

This representation of quaternions is not unique, but we will stick to the above in this paper. It is also possible to represent $\mathbb{H}$ as $M(2,\mathbb{C})$ where $M(2,\mathbb{C})$ is a 2 x 2 complex matrix.

With our real-valued representation a quaternion real-valued $2D$ convolution layer can be expressed as follows. Say that the layer has N feature maps such that N is divisible by 4. We let the first $N/4$ feature maps represent the real components, the second $N/4$ represent the i imaginary components, the third $N/4$ represent the j imaginary components, and the last $N/4$ represent the k imaginary components.

5.2.2 Quaternion Differentiability

In order for the network to perform backpropagation the cost function and activation functions used must be differentiable with respect to the real, i, j, and k components of each quaternion parameter of the network. As the complex chain rule is shown in [TBS+17], we provide the quaternion chain rule which is given in the Appendix section 6.1.

5.2.3 Quaternion Convolution

Convolution in the quaternion domain is done by convolving a quaternion filter matrix $\mathbf{W} = \mathbf{A} + i\,\mathbf{B} + j\,\mathbf{C} + k\,\mathbf{D}$ by a quaternion vector $\mathbf{h} = \mathbf{w} + i\,\mathbf{x} + j\,\mathbf{y} + k\,\mathbf{z}$. Performing the convolution by using the distributive property and grouping terms one gets

$$\begin{aligned}\mathbf{W} * \mathbf{h} = {} & (\mathbf{A} * \mathbf{w} - \mathbf{B} * \mathbf{x} - \mathbf{C} * \mathbf{y} - \mathbf{D} * \mathbf{z}) + \\ & i(\mathbf{A} * \mathbf{x} + \mathbf{B} * \mathbf{w} + \mathbf{C} * \mathbf{z} - \mathbf{D} * \mathbf{y}) + \\ & j(\mathbf{A} * \mathbf{y} - \mathbf{B} * \mathbf{z} + \mathbf{C} * \mathbf{w} + \mathbf{D} * \mathbf{x}) + \\ & k(\mathbf{A} * \mathbf{z} + \mathbf{B} * \mathbf{y} - \mathbf{C} * \mathbf{x} + \mathbf{D} * \mathbf{w}). \end{aligned} \tag{5.1}$$

Using a matrix to represent the components of the convolution we have:

$$\begin{bmatrix} \mathscr{R}(\mathbf{W} * \mathbf{h}) \\ \mathscr{I}(\mathbf{W} * \mathbf{h}) \\ \mathscr{J}(\mathbf{W} * \mathbf{h}) \\ \mathscr{K}(\mathbf{W} * \mathbf{h}) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & -\mathbf{B} & -\mathbf{C} & -\mathbf{D} \\ \mathbf{B} & \mathbf{A} & -\mathbf{D} & \mathbf{C} \\ \mathbf{C} & \mathbf{D} & \mathbf{A} & -\mathbf{B} \\ \mathbf{D} & -\mathbf{C} & \mathbf{B} & \mathbf{A} \end{bmatrix} * \begin{bmatrix} \mathbf{w} \\ \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} \tag{5.2}$$

An example is shown in Fig. 5.1, which is useful to visualize one of the main motivational factors of quaternions for CNNs. Notice that the result of the quaternion convolution produces a unique linear combination of each axis per the result of a single axis. This comes from the structure of quaternion multiplication and is forcing each axis of the kernel to interact with each axis of the image. Real-valued convolution simply multiplies each channel of the kernel with the corresponding channel of the image. The quaternion convolution is similar to a mixture of standard convolution and depthwise separable convolution from [Cho16]. This reuse of filters on every axis and combination may help extract texture information across channels as seen in [SF07]. One can think in terms of a RGB where the greyscale can be the real axis and the RGB channels can be the i, j, k axes. Then a quaternion kernel convolved against this quaternion image will view the colors as a single entity, unlike standard real-valued convolution. Since a quaternion can be thought of as a vector, the quaternion kernels and feature maps can be thought of vector as well.

5.2.4 Quaternion Batch-Normalization

Batch-normalization [IS15] is used by the vast majority of all deep networks to stabilize and speed up training. It works by keeping the activations of the network at zero mean and unit variance. The original formulation of batch-normalization only works for real-values. Applying batch normalization to complex or hyper-complex numbers is more difficult, one can not simply translate and scale them such that their mean is 0 and their variance is 1. This would not give equal variance in the multiple components of a complex or hyper-complex number. To overcome this for complex numbers a whitening approach is used [TBS+17], which scales the data by the square root of their variances along each of the two principle components. We use the same approach, but must whiten 4D vectors.

However, an issue arises in that there is no nice way to calculate the inverse square root of a 4 x 4 matrix. It turns out that the square root is not necessary and we can instead use the Cholesky decomposition on our covariance matrix. The details of why this works for whitening given in the Appendix section 6.2. Now our whitening is accomplished by multiplying the $\mathbf{0}$-centered data $(\mathbf{x} - \mathbb{E}[\mathbf{x}])$ by $\mathbf{W}$:

$$\tilde{x} = \mathbf{W}(\mathbf{x} - \mathbb{E}[\mathbf{x}]) \tag{5.3}$$

where $\mathbf{W}$ is one of the matrices from the Cholesky decomposition of $\mathbf{V}^{-1}$

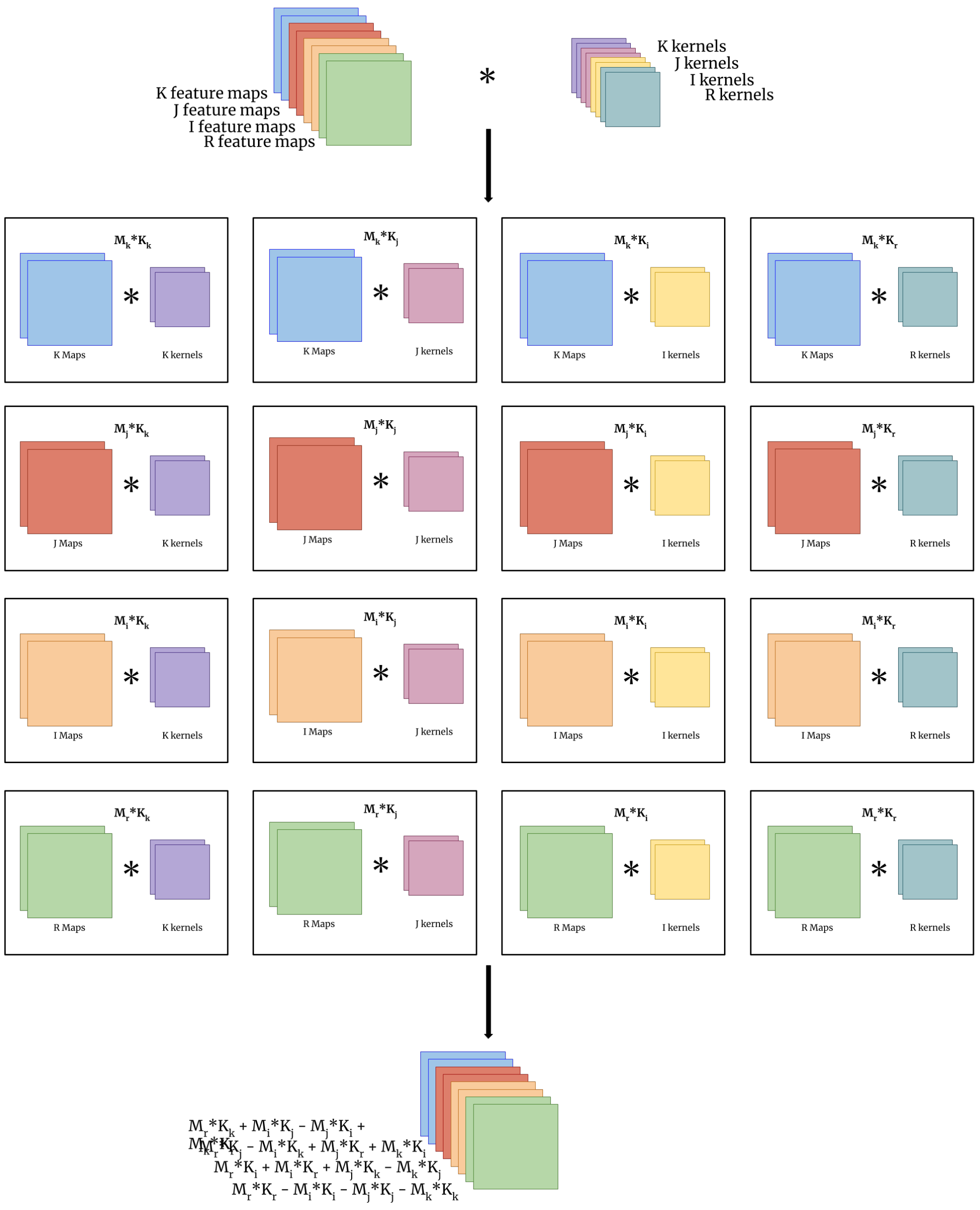

K feature maps
J feature maps
I feature maps
R feature maps
*
K kernels
J kernels
I kernels
R kernels
$M_k * K_k$ K Maps * K kernels
$M_k * K_j$ K Maps * J kernels
$M_k * K_i$ K Maps * I kernels
$M_k * K_r$ K Maps * R kernels
$M_j * K_k$ J Maps * K kernels
$M_j * K_j$ J Maps * J kernels
$M_j * K_i$ J Maps * I kernels
$M_j * K_r$ J Maps * R kernels
$M_i * K_k$ I Maps * K kernels
$M_i * K_j$ I Maps * J kernels
$M_i * K_i$ I Maps * I kernels
$M_i * K_r$ I Maps * R kernels
$M_r * K_k$ R Maps * K kernels
$M_r * K_j$ R Maps * J kernels
$M_r * K_i$ R Maps * I kernels
$M_r * K_r$ R Maps * R kernels
$M_r * K_k + M_i * K_j - M_j * K_i +$
$M_k * K_r$
$M_r * K_j - M_i * K_k + M_j * K_r + M_k * K_i$
$M_r * K_i + M_i * K_r + M_j * K_k - M_k * K_j$
$M_r * K_r - M_i * K_i - M_j * K_j - M_k * K_k$

Figure 5.1: An illustration of quaternion convolution.

where $\mathbf{V}$ is the covariance matrix given by:

$$\begin{aligned}\mathbf{V} &= \begin{bmatrix} V_{rr} & V_{ri} & V_{rj} & V_{rk} \\ V_{ir} & V_{ii} & V_{ij} & V_{ik} \\ V_{jr} & V_{ji} & V_{jj} & V_{jk} \\ V_{kr} & V_{ki} & V_{kj} & V_{kk} \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{C}(\mathscr{R}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{R}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{R}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{R}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{C}(\mathscr{I}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{I}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{I}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{I}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{C}(\mathscr{J}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{J}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{J}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{J}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \\ \mathrm{C}(\mathscr{K}\{\mathbf{x}\},\mathscr{R}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{K}\{\mathbf{x}\},\mathscr{I}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{K}\{\mathbf{x}\},\mathscr{J}\{\mathbf{x}\}) & \mathrm{C}(\mathscr{K}\{\mathbf{x}\},\mathscr{K}\{\mathbf{x}\}) \end{bmatrix}\end{aligned}$$

where $\mathrm{C}(\cdot)$ is the covariance and $\mathscr{R}\{x\}$, $\mathscr{I}\{x\}$, $\mathscr{J}\{x\}$, and $\mathscr{K}\{x\}$ are the real, i, j, and k components of $\mathbf{x}$ respectively.

Real-valued batch normalization also uses two learned parameters, β and γ. Our shift parameter $\boldsymbol{\beta}$ must shift a quaternion value so it is a quaternion value itself with real, i, j, and k as learnable components. The scaling parameter $\boldsymbol{\gamma}$ is a symmetric matrix of size matching $\mathbf{V}$ given by:

$$\gamma = \begin{pmatrix} V_{rr} & V_{ri} & V_{rj} & V_{rk} \\ V_{ri} & V_{ii} & V_{ij} & V_{ik} \\ V_{rj} & V_{ij} & V_{jj} & V_{jk} \\ V_{rk} & V_{ik} & V_{jk} & V_{kk} \end{pmatrix} \tag{5.4}$$

Because of its symmetry it has only ten learnable parameters. The variance of the components of input $\tilde{\mathbf{x}}$ are variance 1 so the diagonal of $\boldsymbol{\gamma}$ is initialized to $1/\sqrt{16}$ in order to obtain a modulus of 1 for the variance of the normalized value. The off diagonal terms of $\boldsymbol{\gamma}$ and all components of $\boldsymbol{\beta}$ are initialized to 0. The quaternion batch normalization is defined as:

$$\mathrm{BN}(\tilde{\mathbf{x}}) = \gamma\tilde{\mathbf{x}} + \beta \tag{5.5}$$

5.2.5 Quaternion Weight Initialization

The proper initialization of weights is vital to convergence of deep networks. In this work we derive our quaternion weight initialization using the same procedure as Glorot and Bengio [GB10] and He et al. [HZRS15b].

To begin we find the variance of a quaternion weight:

$$\begin{aligned} W &= |W| e^{(\cos\phi_1 i + \cos\phi_2 j + \cos\phi_3 k)\theta} \\ &= \mathscr{R}\{W\} + \mathscr{I}\{W\} + \mathscr{J}\{W\} + \mathscr{K}\{W\}. \end{aligned} \tag{5.6}$$

where $|W|$ is the magnitude, θ and ϕ are angle arguments, and $\cos^2\phi_1 + \cos^2\phi_2 + \cos^2\phi_3 = 1$ [PT02].

Variance is defined as

$$\mathrm{Var}(W) = \mathbb{E}[|W|^2] - (\mathbb{E}[W])^2, \tag{5.7}$$

but since W is symmetric around 0 the term $(\mathbb{E}[W])^2$ is 0. We do not have a way to calculate $\mathrm{Var}(W) = \mathbb{E}[|W|^2]$ so we make use of the magnitude of quaternion normal values $|W|$, which follows an independent normal distribution with four degrees of freedom (DOFs). We can then calculate the expected value of $|W|^2$ to find our variance

$$\mathbb{E}[|W|^2] = \int_{\infty}^{\infty} x^2 f(x)\ dx = 4\sigma^2 \tag{5.8}$$

where $f(x)$ is the four DOF distribution given in the Appendix.

And since $\mathrm{Var}(W) = \mathbb{E}[|W|^2]$, we now have the variance of W expressed in terms of a single parameter σ:

$$\mathrm{Var}(W) = 4\sigma^2. \tag{5.9}$$

To follow the Glorot and Bengio [GB10] initialization we have $\mathrm{Var}(W) = 2/(n_{in} + n_{out})$, where n_{in} and n_{out} are the number of input and output units respectivly. Setting this equal to (5.9) and solving for σ gives $\sigma = 1/\sqrt{2(n_{in} + n_{out})}$. To follow He et al. [HZRS15b] initialization that is specialized for rectified linear units (ReLUs) [NH10], then we have $\mathrm{Var}(W) = 2/n_{in}$, which again setting equal to (5.9) and solving for σ gives $\sigma = 1/\sqrt{2n_{in}}$.

As shown in (5.6) the weight has components $|W|$, θ, and ϕ. We can initialize the magnitude $|W|$ using our four DOF distribution defined with the appropriate σ based on which initialization scheme we are following. The angle components are initialized using the uniform distribution between $-\pi$ and π where we ensure the constraint on ϕ.

5.3 Experimental Results

Our experiments covered image classification using both the CIFAR-10 and CIFAR-100 benchmarks and image segmentation using the KITTI Road Estimation benchmark.

5.3.1 Classification

We use the same architecture as the large model in [TBS+17], which is a 110 layer Residual model similar to the one in [HZRS16]. There is one difference between the real-valued network and the ones used for both the complex and hyper-complex valued networks. Because the datasets are all real-valued the network must learn the imaginary or quaternion components. We use the same technique as [TBS+17] where there is an additional block immediately after the input which will learn the hyper-complex components

$$BN \rightarrow ReLU \rightarrow Conv \rightarrow BN \rightarrow ReLU \rightarrow Conv.$$

Since all datasets that we work with are real-valued, we present a way to learn their imaginary components to let the rest of the network operate in the complex plane. We learn the initial imaginary component of our input by performing the operations present within a single real-valued residual block

This means that to maintain the same approximate parameter count the number of convolution kernels for the complex network was increased. We however did not increase the number of convolution kernels for the quaternion trials so any increase in model performance comes from the quaternion filters and at a lower hardware budget.

The architecture consists of 3 stages of repeating residual blocks where at the end of each stage the images are downsized by strided convolutions. Each stage also doubles the previous stage's number of convolution kernels. The last layers are a global average pooling layer followed by a single fully connected layer with a softmax function used to classify the input as either one of the 10 classes in CIFAR-10 or one of the 100 classes in CIFAR-100.

We also followed their training procedure of using the backpropagation algorithm with Stochastic Gradient Descent with Nesterov momentum [Nes] set at 0.9. The norm of the gradients are clipped to 1 and a custom learning rate scheduler is used. The learning scheduler is the same used in [TBS+17] for a direct comparison in performance. The learning rate is initially set to 0.01 for the first 10 epochs and then set it to 0.1 from epoch 10-100 and then cut by a factor of 10 at epochs 120 and 150. Table 5.1 presents our results along side the real and complex valued networks. Our quaternion model outperforms the real and complex networks on both datasets on a smaller parameter budget.

Architecture	CIFAR-10	CIFAR-100
[HZRS16] Real	6.37	-
[TBS+17] Complex	5.60	27.09
Quaternion	**5.44**	**26.01**

Table 5.1: Classification error on CIFAR-10 and CIFAR-100. Note that [HZRS16] is a 110 layer residual network, [TBS+17] is 118 layer complex network with the same design as the prior except with additional initial layers to extract complex mappings.

5.3.2 Segmentation

For this experiment we used the same model as the above, but cut the number of residual blocks out of the model for memory reasons given that the KITTI data is large color images. The 110 layer model has 10, 9, and 9 residual blocks in the 3 stages talked about above, while this model has 2, 1, and 1 and does not perform any strided convolutions. This gives a total of 38 layers.

The last layer is a 1×1 convolution with a sigmoid output so we are getting a heatmap prediction the same size as the input. The training procedure is also as above, but the learning rate is scheduled differently. Here we begin at 0.01 for the first 10 epochs and then set it to 0.1 from epoch 10-50 and then cut by a factor of 10 at 100 and 150. Table 5.2 presents our results along side the real and complex valued networks where we used Intersection over

Union (IOU) for performance measure. Quaternion outperformed the other two by a larger margin compared to the classification tasks.

Architecture	KITTI
Real	0.747
Complex	0.769
Quaternion	**0.827**

Table 5.2: IOU on KITTI Road Estimation benchmark.

5.4 Conclusions and Proposed Research

We have extended upon work looking into complex valued networks by exploring quaternion values. We presented the building blocks required to build and train deep quaternion networks and used them to test residual architectures on two common image classification benchmarks. We show that they have competitive performance by beating both the real and complex valued networks with less parameters.

Since the entirety of the work so far has been done in supervised learning, it seems prudent to explore results in unsupervised learning compared to real and complex networks. The proposed idea is to test quaternion values in a Deep Adversarial Gaussian Mixture Autoencoder [HMB17]. We will compare the current results of with results of both complex and quaternion networks. Another area to explore would be in networks designed to handle 3D data given quaternions natural expressive capapilities in this area. PointNet [QSMG17] and its newer variants take in a group of 3D points and use a deep network to classify the entire cloud as a class and/or classify each point in the cloud as a class. We will compare the latest PointNet variant with complex and quaternion value models.

The majority of work was the upfront work of creating and coding the algorithms needed for deep quaternion networks. With that done the work to explore these last two ideas only involves pulling the authors code and replacing their real layers with complex and then quaternion layers. Including running the experiments the amount of time would be less than 6 months.

Chapter 6

Appendix

6.1 The Generalized Quaternion Chain Rule for a Real-Valued Function

We start by specifying the Jacobian. Let L be a real valued loss function and q be a quaternion variable such that $q = a + i\,b + j\,c + k\,d$ where $a, b, c, d \in \mathbb{R}$ then,

$$\begin{aligned}\nabla_L(q) = \frac{\partial L}{\partial q} &= \frac{\partial L}{\partial a} + i\,\frac{\partial L}{\partial b} + j\,\frac{\partial L}{\partial c} + k\,\frac{\partial L}{\partial d} \\ &= \frac{\partial L}{\partial \mathbb{R}(q)} + i\,\frac{\partial L}{\partial \mathbb{I}(q)} + j\,\frac{\partial L}{\partial \mathbb{J}(q)} + k\,\frac{\partial L}{\partial \mathbb{K}(q)} \\ &= \mathbb{R}(\nabla_L(q)) + \mathbb{I}(\nabla_L(q)) + \mathbb{J}(\nabla_L(q)) + \mathbb{K}(\nabla_L(q))\end{aligned} \tag{6.1}$$

Now let $g = m + i\,n + j\,o + k\,p$ be another quaternion variable where q can be expressed in terms of g and $m, n, o, p \in \mathbb{R}$ we then have,

$$\begin{aligned}
\nabla_L(q) &= \frac{\partial L}{\partial g} = \frac{\partial L}{\partial m} + i\,\frac{\partial L}{\partial n} + j\,\frac{\partial L}{\partial o} + k\,\frac{\partial L}{\partial p} \qquad (6.2)\\
&= \frac{\partial L}{\partial a}\frac{\partial a}{\partial m} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial m} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial m} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial m}\\
&\quad + i\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial n} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial n} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial n} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial n}\right)\\
&\quad + j\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial o} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial o} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial o} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial o}\right)\\
&\quad + k\,\left(\frac{\partial L}{\partial a}\frac{\partial a}{\partial p} + \frac{\partial L}{\partial b}\frac{\partial b}{\partial p} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial p} + \frac{\partial L}{\partial d}\frac{\partial d}{\partial p}\right)\\
&= \frac{\partial L}{\partial a}\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + j\,\frac{\partial a}{\partial o} + k\,\frac{\partial a}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial b}\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + j\,\frac{\partial b}{\partial o} + k\,\frac{\partial b}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial c}\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + j\,\frac{\partial c}{\partial o} + k\,\frac{\partial c}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial d}\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + j\,\frac{\partial d}{\partial o} + k\,\frac{\partial d}{\partial p}\right)\\
&= \frac{\partial L}{\partial \mathbb{R}(q)}\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + j\,\frac{\partial a}{\partial o} + k\,\frac{\partial a}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial \mathbb{I}(q)}\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + j\,\frac{\partial b}{\partial o} + k\,\frac{\partial b}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial \mathbb{J}(q)}\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + j\,\frac{\partial c}{\partial o} + k\,\frac{\partial c}{\partial p}\right)\\
&\quad + \frac{\partial L}{\partial \mathbb{K}(q)}\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + j\,\frac{\partial d}{\partial o} + k\,\frac{\partial d}{\partial p}\right)\\
&= \mathbb{R}(\nabla_L(q))\left(\frac{\partial a}{\partial m} + i\,\frac{\partial a}{\partial n} + j\,\frac{\partial a}{\partial o} + k\,\frac{\partial a}{\partial p}\right)\\
&\quad + \mathbb{I}(\nabla_L(q))\left(\frac{\partial b}{\partial m} + i\,\frac{\partial b}{\partial n} + j\,\frac{\partial b}{\partial o} + k\,\frac{\partial b}{\partial p}\right)\\
&\quad + \mathbb{J}(\nabla_L(q))\left(\frac{\partial c}{\partial m} + i\,\frac{\partial c}{\partial n} + j\,\frac{\partial c}{\partial o} + k\,\frac{\partial c}{\partial p}\right)\\
&\quad + \mathbb{K}(\nabla_L(q))\left(\frac{\partial d}{\partial m} + i\,\frac{\partial d}{\partial n} + j\,\frac{\partial d}{\partial o} + k\,\frac{\partial d}{\partial p}\right)
\end{aligned}$$

6.2 Whitening a Matrix

Let $\mathbf{X}$ be an n x n matrix and cov($\mathbf{X}$) = $\boldsymbol{\Sigma}$ is the symmetric covariance matrix of the same size. Whitening a matrix linearly decorrelates the input dimensions, meaning that whitening transforms $\mathbf{X}$ into $\mathbf{Z}$ such that cov($\mathbf{Z}$) = $\mathbf{I}$ where $\mathbf{I}$ is the identity matrix [KLS17]. The matrix $\mathbf{Z}$ can be written as:

$$\mathbf{Z} = \mathbf{W}(\mathbf{X} - \mu) \tag{6.3}$$

where $\mathbf{W}$ is an n x n 'whitening' matrix. Since cov($\mathbf{Z}$) = $\mathbf{I}$ it follows that:

$$\begin{aligned}
&\mathbb{E}[\mathbf{Z}\mathbf{Z}^T] = \mathbf{I} \\
&\mathbb{E}[\mathbf{W}(\mathbf{X} - \mu)(\mathbf{W}(\mathbf{X} - \mu))^T] = \mathbf{I} \\
&\mathbb{E}[\mathbf{W}(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T\mathbf{W}^T] = \mathbf{I} \\
&\mathbf{W}\Sigma\mathbf{W}^T = \mathbf{I} \\
&\mathbf{W}\Sigma\mathbf{W}^T\mathbf{W} = \mathbf{W} \\
&\mathbf{W}^T\mathbf{W} = \boldsymbol{\Sigma}^{-1} \qquad (6.4)
\end{aligned}$$

From (6.4) it is clear that the Cholesky decomposition provides a suitable (but not unique) method of finding $\mathbf{W}$.

6.3 Cholesky Decomposition

Cholesky decomposition is an efficient way to implement LU decomposition for symmetric matrices, which allows us to find the square root. Consider $\mathbf{AX} = \mathbf{b}$, $\mathbf{A} = [a_{ij}]_{n\times n}$, and $a_{ij} = a_{ji}$, then the Cholesky decomposition of $\mathbf{A}$ is given by $\mathbf{A} = \mathbf{LL}'$ where

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \tag{6.5}$$

Let l_{ki} be the k^{th} row and i^{th} column entry of $\mathbf{L}$, then

$$l_{ki} = \begin{cases} 0, & k < i \\ \sqrt{a_{ii} - \sum_{j=1}^{i-1} l_{kj}^2}, & k = i \\ \frac{1}{l_{ii}}(a_{ki} - \sum_{j=1}^{i-1} l_{ij} l_{kj}), & i < k \end{cases}$$

6.4 4 DOF Independent Normal Distribution

Consider the four-dimensional vector $\mathbf{Y} = (S, T, U, V)$ which has components that are normally distributed, centered at zero, and independent. Then S, T, U, and V all have density functions

$$f_S(x;\sigma) = f_T(x;\sigma) = f_U(x;\sigma) = f_V(x;\sigma) = \frac{e^{-x^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}. \tag{6.6}$$

Let $\mathbf{X}$ be the length of $\mathbf{Y}$, which means $\mathbf{X} = \sqrt{S^2 + T^2 + U^2 + V^2}$. Then $\mathbf{X}$ has the cumulative distribution function

$$F_X(x;\sigma) = \iiiint_{H_x} f_S(\mu;\sigma) f_T(\mu;\sigma) f_U(\mu;\sigma) f_V(\mu;\sigma)\ dA, \tag{6.7}$$

where H_x is the four-dimensional sphere

$$H_x = \left\{ (s,t,u,v)\ :\ \sqrt{s^2 + t^2 + u^2 + v^2} < x \right\}. \tag{6.8}$$

We then can write the integral in polar representation

$$\begin{aligned} F_X(x;\sigma) &= \frac{1}{4\pi^2\sigma^4} \int_0^{\pi}\int_0^{\pi}\int_0^{2\pi}\int_0^{x} r^3 e^{\frac{-r^2}{2\sigma^2}} \sin(\theta)\sin(\phi)\cos(\psi)\ dr d\theta d\phi d\psi \\ &= \frac{1}{2\sigma^4} \int_0^x r^3 e^{-r^2/(2\sigma^2)}\ dr. \end{aligned} \tag{6.9}$$

The probability density function of $\mathbf{X}$ is the derivative of its cumulative distribution function so we use the funamental theorem of calculus on (6.9) to finally arrive at

$$f_X(x;\sigma) = \frac{d}{dx} F_X(x;\sigma) = \frac{1}{2\sigma^4} x^3 e^{-x^2/(2\sigma^2)}. \tag{6.10}$$

Bibliography

[App00] Apple. Performing convolution operations. https://developer.apple.com/library/content/documentation/Performance/Concep
2000.

[BF13] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[BS01] Thomas Bulow and Gerald Sommer. Hypercomplex signals-a novel extension of the analytic signal to the multidimensional case. *IEEE Transactions on signal processing*, 49(11):2844–2852, 2001.

[Bül99] Thomas Bülow. *Hypercomplex spectral signal representations for the processing and analysis of images*. Universität Kiel. Institut für Informatik und Praktische Mathematik, 1999.

[Cho16] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[CXLP13] Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan. Vehicle detection in satellite images by parallel deep convolutional neural networks. In *Pattern Recognition (ACPR), 2013 2nd IAPR Asian Conference on*, pages 181–185. IEEE, 2013.

[DEKLD16] Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, and Larry S Davis. Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection. *arXiv preprint arXiv:1610.03466*, 2016.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[HBF$^+$01] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[HDWF$^+$17] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[HMB17] Warith Harchaoui, Pierre-Alexandre Mattei, and Charles Bouveyron. Deep adversarial gaussian mixture auto-encoder for clustering. 2017.

[HN$^+$88] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[HSK$^+$12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[Hun80] Thomas W Hungerford. Algebra, volume 73 of graduate texts in mathematics, 1980.

[HW68] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[HZRS15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[HZRS15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[Jia08] Yan-Bin Jia. Quaternions and rotations. *Com S*, 477(577):15, 2008.

[Kar13] Andrej Karpathy. Convolutional neural networks (cnns / convnets). http://cs231n.github.io/convolutional-networks/#pool, 2013.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[KLS17] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, (just-accepted), 2017.

[KPU15] Philipp Kainz, Michael Pfeiffer, and Martin Urschler. Semantic segmentation of colon glands with deep convolutional neural networks and total variation segmentation. *arXiv preprint arXiv:1511.06919*, 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBD$^+$90] Y LeCun, B Boser, JS Denker, D Henderson, RE Howard, W Hubbard, and LD Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems 2, NIPS 1989*, pages 396–404. Morgan Kaufmann Publishers, 1990.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[LJH15] Wen Li, Fucang Jia, and Qingmao Hu. Automatic segmentation of liver tumor in ct images with deep convolutional neural networks. *Journal of Computer and Communications*, 3(11):146, 2015.

[LPAL15] Mark Lyksborg, Oula Puonti, Mikael Agn, and Rasmus Larsen. An ensemble of 2d convolutional neural networks for tumor segmentation. In *Scandinavian Conference on Image Analysis*, pages 201–211. Springer, 2015.

[MBLAGJ$^+$07] Saturnino Maldonado-Bascon, Sergio Lafuente-Arroyo, Pedro Gil-Jimenez, Hilario Gomez-Moreno, and Francisco López-Ferreras. Road-sign detection and recognition based on support vector machines. *IEEE transactions on intelligent transportation systems*, 8(2):264–278, 2007.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[MINM17] Toshifumi Minemoto, Teijiro Isokawa, Haruhiko Nishimura, and Nobuyuki Matsui. Feed forward neural network with random quaternionic neurons. *Signal Processing*, 136:59–68, 2017.

[Nes] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2).

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[OL81] Alan V Oppenheim and Jae S Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5):529–541, 1981.

[PT02] Robert Piziak and Danny Turner. The polar form of a quaternion. 2002.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[SE00] Stephen J Sangwine and Todd A Ell. Colour image filters based on hypercomplex convolution. *IEE Proceedings-Vision, Image and Signal Processing*, 147(2):89–93, 2000.

[SF07] Lilong Shi and Brian Funt. Quaternion color texture segmentation. *Computer Vision and image understanding*, 107(1):88–96, 2007.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[TBS$^+$17] Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[TGCd16] Ludovic Trottier, Philippe Giguère, and Brahim Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks. *arXiv preprint arXiv:1605.09332*, 2016.

[TGJ$^+$15] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[WM13] Sida I Wang and Christopher D Manning. Fast dropout training. In *ICML (2)*, pages 118–126, 2013.

[ZRM$^+$13] Matthew D Zeiler, M Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521. IEEE, 2013.

List of Figures

List of Tables