CLASSIFYING OBJECTS FROM OVERHEAD SATELLITE IMAGERY USING CAPSULES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Darren G. Rodriguez

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Michael D. Zoltowski, Chair

School of Electrical and Computer Engineering

Dr. David J. Love

School of Electrical and Computer Engineering

Dr. Mark R. Bell

School of Electrical and Computer Engineering

Dr. Charles A. Bouman

School of Electrical and Computer Engineering

Approved by:

Dr. Pedro Irazoqui

Head of the School Graduate Program

For my grandfather, Rudolfo Rodriguez.

ACKNOWLEDGMENTS

I would like to thank numerous people for their constant support throughout this journey.

For my parents Norman and Elberta, thank you for always believing in me and encouraging me to be the best person I can be. From traveling to all of my sports activities in high school to helping me move across the U.S. and back, you've always been there for me and I don't know where I would be without you both.

To my girlfriend Alicia, words can't explain how much you've done for me over the past several years. Your daily video calls always gave me something to look forward to no matter how hard things got. Your words of encouragement (and sometimes stern reminders) helped me to stay on track and remind me that I chose to challenge myself and have to follow through with it. I love you and I'll be home soon.

For my advisor Dr. Zoltowski, thank you for embarking on this journey with me even though it's not your primary area of research. Your advice and suggestions kept me from straying too far off track. I hope this work has helped you learn about the field as much as it helped me. I look forward to reading your future publications.

Lastly to my roommate and best friend Blake the cat, your companionship over the past few years has helped me stay calm amongst the stress of school. Treats and toys are coming your way soon!

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

ANN	Artificial Neural Network
CapsNet	Capsule Network
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit

ABSTRACT

Rodriguez, Darren G. M.S., Purdue University, May 2019. Classifying Objects from Overhead Satellite Imagery Using Capsules. Major Professor: Michael D. Zoltowski.

Convolutional neural networks lie at the heart of nearly every object recognition system today. While their performance continues to improve through new architectures and techniques, some of their deficiencies have not been fully addressed to date. Two of these deficiencies are their inability to distinguish the spatial relationships between features taken from the data, as well as their need for a vast amount of training data. Capsule networks, a new type of convolutional neural network, were designed specifically to address these two issues. In this work, several capsule network architectures are utilized to classify objects taken from overhead satellite imagery. These architectures are trained and tested on small datasets that were constructed from the xView dataset, a comprehensive collection of satellite images originally compiled for the task of object detection. Since the objects in overhead satellite imagery are taken from the same viewpoint, the transformations exhibited within each individual object class consist primarily of rotations and translations. These spatial relationships are exploited by capsule networks. As a result it is shown that capsule networks achieve considerably higher accuracy when classifying images from these constructed datasets than a traditional convolutional neural network of approximately the same complexity.

1. INTRODUCTION

1.1 Background

Object recognition, or image classification in particular has become a highly researched topic in the past several years in large part due to the advent of convolutional neural networks and their respective performance on complex classification tasks. Image classification technology has revolutionized and automated many practical systems that would otherwise require a human operator and in many cases has exceeded the performance of human operators [1]. While object recognition technology has demonstrated near perfect accuracy on large, well defined datasets it often struggles to exhibit the same level of performance on more complex and smaller datasets.

Many of the common convolutional neural network architectures being utilized today are excellent at extracting features from raw data, however they tend to have a few major drawbacks. In a standard convolutional neural network, spatial relationships between objects are not preserved. This is caused by a common operation that takes place between convolutional layers called max-pooling, where some of the data is effectively discarded in an effort to achieve some amount of translational invariance as well as downsampling the data to a more manageable size. Another issue surrounding the use of convolutional neural networks is the immense amount of data required to train them.

In December 2017 Dr. Geoffrey Hinton of Google AI introduced a new type of convolutional neural network [2] aimed at addressing the two problems mentioned above. The so-called capsule network aims to represent data in a way that preserves spatial relationships while requiring less data to train. In this thesis capsule networks are explored as a method of recognizing objects taken from overhead satellite im-

agery which consists of many similar objects in various different poses (rotations and translations).

1.2 Problem Statement

Overhead satellite imagery consists of images of the Earth or other planets that are collected by imaging satellites in either the public or private domain. Among the publicly available satellite imagery datasets is the xView [3] dataset, which consists of images covering over $1400km^2$ of the Earth's surface and contains over 1 million labeled instances split into 60 total classes for the task of object detection, that is localization and identification of objects. As the task of localization of objects within these images is a separate problem on its own, it will not be explored in this work.

Working under the assumption that objects or areas of interest can be identified by some method or algorithm, this thesis aims to demonstrate that capsule networks can be utilized to correctly identify the objects contained within those areas of interest. This is due to the inherent nature of capsule networks to preserve spatial relationships between features and thus be more robust to changes in pose; satellite imagery consists of many similar objects with differences in background and pose. Several capsule network models are developed and compared to a traditional convolutional neural network with approximately the same number of trainable parameters to demonstrate that capsule networks are superior in identifying objects in this domain.

2. RELATED WORK

2.1 Object Recognition History

Modern computer vision research can be traced back at least as far as the 1960s where pattern recognition was applied in an effort to automate office related tasks [4]. Since then many other traditional methods of object recognition have been developed as highlighted in [5], however none of these methods could match the performance of convolutional neural networks once they broke through in 2012 [6]. Since then CNNs have been the center of focus for the majority of object recognition and object detection tasks due to their performance relative to traditional methods.

Much of the research conducted in the field of CNNs serves as the foundation for the work in this thesis. Some of the more notable works are described in more detail in this section.

2.2 Artificial Neural Networks

The term artificial neural network arises from the fact that this type of logical network is designed to mimic parts of the human brain, more specifically the way neurons in the brain communicate with one another. In an ANN neurons are connected to one another and can transmit signals, real numbers in this case, to other neurons. This idea combined with a non-linear function applied at the output of each layer of neurons allows an ANN to classify images into different categories. In general an ANN can have one or more hidden layers that apply weights to their inputs, these weights are updated as learning advances. Figure 2.1 shows a simple multilayer perceptron, a type of ANN.

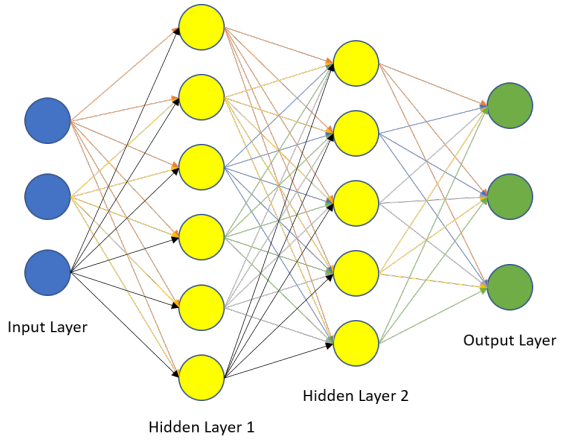


Fig. 2.1. Multilayer Perceptron

The general idea surrounding ANNs is that each neuron learns a weight and applies it to a signal. The weight causes the strength of a signal to either increase or decrease, therefore signals only get passed to deeper neurons if their signal strength is increased by a previous neuron. This can be thought of in reverse as well, as some signals will have their strength decreased to the point that they do not pass to neurons deeper in the network. The neurons that are utilized in ANNs represent features that are learned from the input layer. Unlike traditional machine learning methods these features do not have to be designed and processed by a human prior to their use. Instead the network itself learns features on its own through training which saves considerable time and can offer better performance overall.

2.3 Convolutional Neural Networks

Convolutional Neural Networks are almost synonymous with object recognition and object detection today. CNNs are a class of feed forward neural networks that are composed of one or more convolutional layers, pooling layers, and fully connected layers. While the idea of CNNs has been around for quite some time, it wasn't until 2012 that their application was fully realized in [6], where a CNN won the ImageNet

Large Scale Visual Recognition Competition by a large margin. CNNs have won the competition every year since. A simple CNN architecture is depicted in Figure 2.2. It should be noted that CNN architectures are generally much more complex than the one in Figure 2.2.

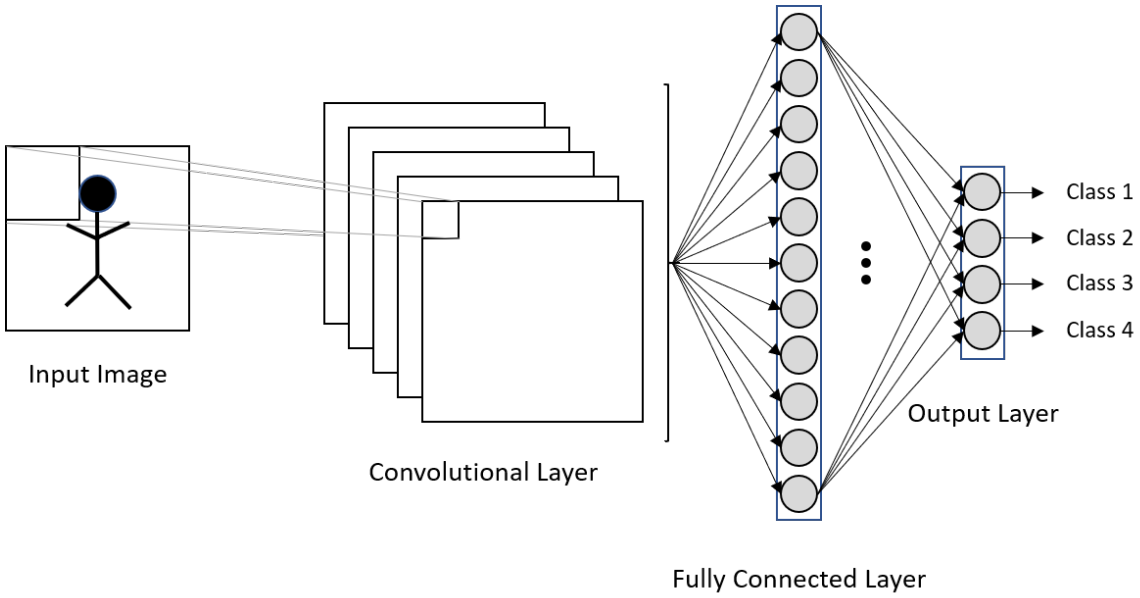


Fig. 2.2. A simple CNN with one convolutional layer. Note that a pooling layer is not shown.

Within the convolutional layers various kernels are used to convolve through the original image as well as the intermediate layers in an effort to extract features from the image. This has a few advantages; since the kernels are applied to the whole image or feature map, location invariance occurs. This allows CNNs to recognize objects regardless of their position within an image. The weight sharing mechanism that occurs within feature maps also helps to reduce overall model complexity by reducing the number of trainable parameters or the number of operations that need to occur.

Between convolutional layers an activation function is applied to the feature map to introduce nonlinearities into the network. Without this function the network would

essentially be a linear regression model and would struggle on complex data. Some of the most common activation functions include the rectified linear unit (ReLU) $f(x) = max(0, x)$, the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$, and the softmax function $f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$, $for\ j = 1\ to\ K$. The softmax function also has the benefit of squeezing each output to lie between zero and one and diving by the sum of the outputs. This allows the outputs to be interpreted as a probability and is often used at the final layer.

The pooling layer which is usually placed between convolutional layers provides a method of reducing the feature map size while also reducing the possibility of the network overfitting to training data. This operation works by dividing a feature map into an $N \times M$ grid and sampling typically only one value from within each cell. Most often used are max pooling, where the max value in each grid cell is kept, and average pooling where the average of each grid cell is taken. While pooling layers have been shown to improve performance of CNNs, there are some inherent problems that arise through their use which will be explored in the next chapter. Figure 2.3 demonstrates max pooling.

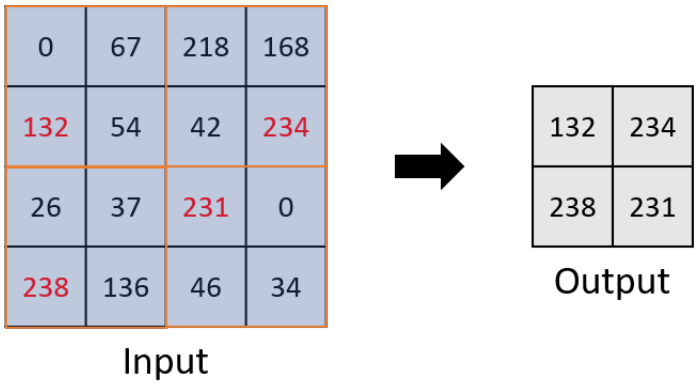


Fig. 2.3. Example of max pooling with a 2×2 kernel and a stride of 2.

Fully connected layers in CNNs are often a bottleneck in performance considering that they are typically of fixed size and require all inputs from the previous layer to

be fed forward to them as inputs. It is quite common to see fully connected layers at the output of a CNN for use as a classifier, where each neuron in the layer is meant to represent a single class, however fully connected layers can also be utilized elsewhere in a CNN usually at the cost of efficiency.

Convolutional neural networks are trained in two stages utilizing forward and backward propagation. In the forward stage the input is fed through the CNN utilizing the current weights and biases to compute an output prediction. Once this is acquired a loss function is used to compute the error between the prediction and a ground truth label. With the loss available backward propagation begins where the gradient of the error with respect to each parameter is computed and utilized to update the parameters for the next forward pass. This process is repeated over multiple iterations and only stops when a sufficiently low error rate is reached.

Training a CNN would not be possible without backpropogation [7] and stochastic gradient descent (SGD) [8]. Since the ultimate goal of a CNN is to map arbitrary inputs to chosen outputs, CNNs can in a sense be thought of as a large optimization problem. Backpropagation provides a method of computing the gradient of a loss function while SGD is one of several methods that can be used to perform learning, or parameter adjusting based on the gradient.

Training of a CNN is a constant battle between two extremes, namely overfitting and underfitting. Overfitting occurs when a CNN adjusts its parameters in such a way that it performs very well on the training dataset and achieves a small error percentage, however when fed input from a testing dataset the CNN performance drops considerably. In other words, overfitting occurs when a CNN cannot generalize to data that was not part of the training dataset. Overfitting can be combated by data augmentation, that is creating synthetic data to add to the training dataset usually by performing rotations, translations, etc. on the original data. Dropout [9] is yet another popular method that is used to combat overfitting, where nodes or neurons within the network are "turned off" or zeroed out during training, usually at random. This is done to encourage the network to learn more robust features.

Underfitting of a neural network occurs when a sufficiently low error rate cannot be obtained. Generally this is indicative of a poor network architecture, not enough training data, or very complex training data. Underfitting is an issue that is difficult to resolve as it usually requires more data to be collected and even then there is no guarantee that more data will allow for better results.

2.4 Satellite Imagery

The history of satellite imagery dates back to at least 1946 when the U.S. launched V2 rocket recorded images from a 65 mile apogee, which was higher than the 1935 Explorer II balloon images. Since then huge advances have been made in both satellite technology as well as the imaging methods that are utilized. Today satellite imagery is used in a number of different applications ranging from meteorology and oceanography to forestry and warfare.

Satellite imagery can be obtained with ground plane resolutions of less than 1 meter. This accompanied with the fact that a single image can cover several kilometers of the Earth's surface results in databases that are extremely large, making satellite imagery a prime candidate for machine learning and deep learning processing techniques. These techniques have already been applied successfully as in [10], however the public availability of data has hampered the progression of research in this area. The DIUx xView challenge [3] aided in ameliorating this issue by presenting a public challenge utilizing satellite imagery and providing a large, well annotated dataset for experimentation. An example image from this dataset with annotations is shown in Figure 2.4. This dataset was compiled for the task of object detection and as such the provided labels consist of the top left and bottom right point of a bounding box that encompasses a single object as well as the object name. These annotations can be used to crop out objects from their parent images as is done in this work.

Fig. 2.4. Example image from xView dataset with annotations.

3. CAPSULES

Since the resurgence of convolutional neural networks in 2012, many efforts have been made to improve their performance in a number of different tasks as well as apply CNNs to new problems with varying degrees of success. While many of these efforts have proved successful, they often fail to address some of the limitations of CNNs in general. Two of these limitations are the amount of data required to train a CNN to an acceptable level of accuracy, and their inability to retain information about the spatial relationships of the objects contained within images.

The amount of data required to successfully train a convolutional neural network varies depending on the complexity of the data itself, however it is generally accepted that more data will lead to better performance. While many datasets are now available to the general public the issue remains that to construct a new dataset for use in object recognition the data itself must be meticulously chosen and labeled, often by hand. This quickly becomes a problem when datasets are expected to contain hundreds of thousands or even millions of instances each with corresponding labels. The popular ImageNet Large Scale Visual Recognition Challenge [11] offers over 14 million images spread across over 20000 categories, with one thousand of those categories used in the yearly ImageNet object recognition challenge. Although this dataset is a great asset for advancing CNN technology it is but one example that fails to address why so many images are needed in the first place.

Convolutional neural networks exhibit translation invariance, a property that allows a network to recognize objects regardless of their spatial location within an image. This property arises mainly from the pooling operation that takes place between layers in a CNN, where any particular feature is noted as being present in an image however the relative location of that feature is cast aside. While this is desirable in the sense that it allows CNNs to generalize to data that it was not trained on,

it is often argued that translation invariance is not ideal and instead we should strive for translation equivariance. Rather than having the ability to recognize an object regardless of its location in an image, it is more beneficial to recognize that the same object is present in multiple images and only its pose (rotation, translation, scale, reflection) has changed. Figure 3.1 demonstrates a simple example of translational invariance and why it is an issue.

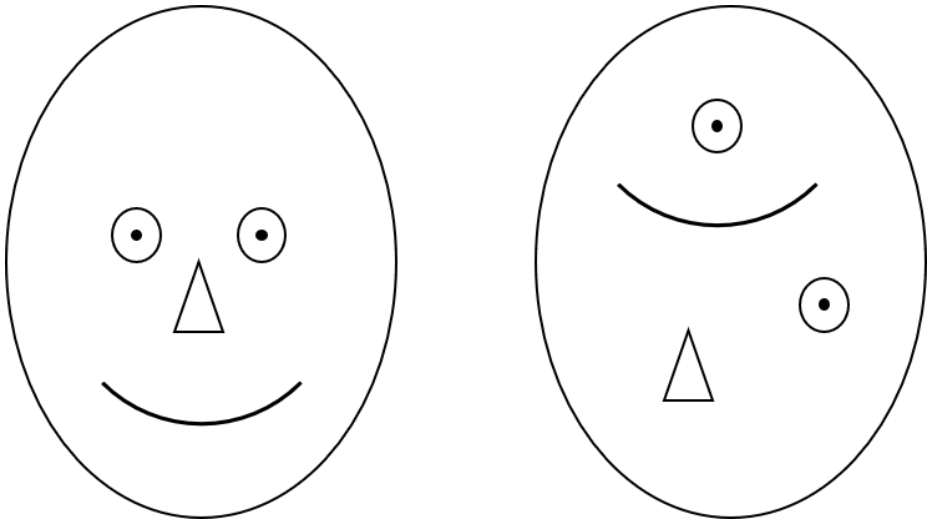


Fig. 3.1. The problem with invariance: To a traditional CNN, these images are the same and both represent a face. The location of features is not important, only their existence matters.

Effort has been made in designing feature detectors that can recognize not only features in images but also their pose information. In the scale-invariant feature transform (SIFT) [12] key points in an image are extracted and stored in a database so that they can later be compared to new images. These points or features vectors are matched to new image based on Euclidean distance, and those that highly agree are considered the same features leading to a match. In an effort to speed up run time variations of SIFT have been developed including speeded up robust features (SURF) [13], however these techniques are generally complicated and hand crafted for individual problems.

Capsules, which were introduced by Geoffrey Hinton in [2], are a group of neurons that together form a vector that represents different properties of an entity. The use of scalar inputs and outputs in traditional CNNs is what leads to their deficiencies as highlighted above. Capsules aim to solve these issues by encoding the probability of an entity being present in an image as well as providing instantiation parameters, or pose information, about that object. This concept is what led to [14, 15] which serve as the foundation for the work in this thesis.

3.1 Transforming Auto-Encoders

The idea of utilizing capsules, or small vectors, to represent visual entities and their corresponding instantiation parameters was explored in 2011 by Geoffrey Hinton and others in [2]. The main motivation for this was that although CNNs had at that point proved effective to solve many complex problems, their implementation was misguided in that they aim for viewpoint invariance through the use of scalar outputs between convolutional layers. Instead viewpoint equivariance should be sought after, where the likelihood of a visual entity being present within an image is encoded as normal but its instantiation parameters change as the entity moves around its domain of viewing conditions. This idea allows for whole visual entities to be broken into parts and the whole entity can only be recognized if its individual parts agree spatially, an issue that traditional CNNs had yet to solve as they search for parts but do not care about their spatial relationship to one another.

To demonstrate the utility of capsules and how an artificial neural network can learn to convert pixel intensities to instantiation parameters, the authors of [2] propose a transforming auto-encoder. This feedforward neural network takes as its input an image and a desired shift amount Δx and Δy. The network learns to output the same image that is shifted by the desired amount. Utilizing the MNIST [16] dataset the auto-encoder is trained utilizing 30 capsules each containing 10 recognition units and 20 generation units. Each image from the MNIST dataset is shifted by at most

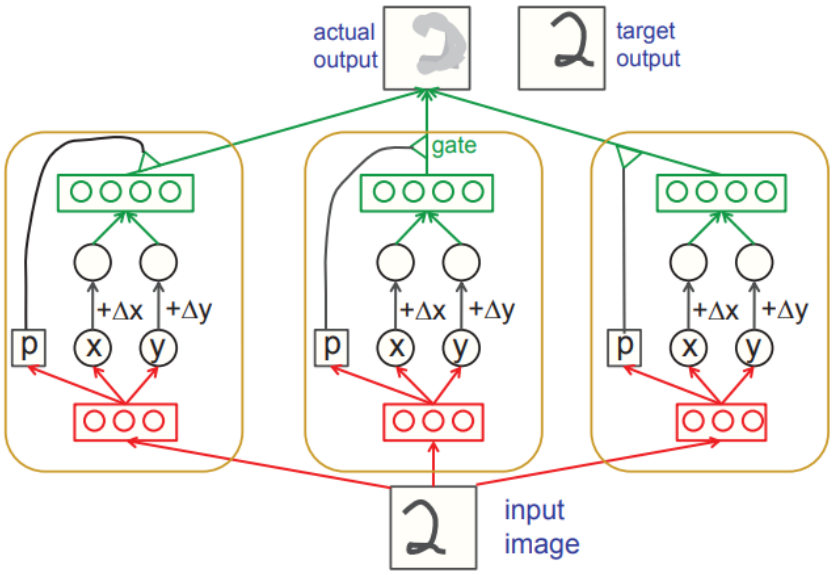


Fig. 3.2. The transforming auto-encoder. Taken from [2]

2 pixels in a random direction and this shift is supplied to the network as an input alongside the image. The network successfully learns to transform the input images by the desired amount. The utility of capsules was further explored by extending to 3-D data, where the auto-encoder was able to learn 3 dimensional transformations between computer generated car models further strengthening the idea of utilizing capsules instead of simple scalars and pooling operations.

3.2 Dynamic Routing Between Capsules

In [14], the capsule network or CapsNet is introduced where layers of capsules are used to make predictions for layers of capsules at a higher level. This is accomplished through the routing by agreement algorithm where lower level capsules send their input to higher level capsules that best "agree" with its input. Termed routing by agreement, this iterative algorithm was proved to not only to be more effective than

the max pooling method in traditional CNNs in [14], but also to exhibit the property of equivariance. Equivariance allows the trained model to recognize objects in differing poses to those that the model was initially trained on. As an added benefit the spatial relationship between features is also preserved. This was all made possible by trading scalar features for vector features, or capsules, and replacing pooling with routing by agreement.

Routing by agreement is a dynamic routing algorithm that is repeated a number of times during each forward pass of the network. For each capsule in the current layer, a vector $\boldsymbol{c_i}$ is calculated which represents the routing weights for a lower level capsule $\boldsymbol{i}$. Initially all coefficients c_{ij} are set to zero representing a state of maximum confusion and uncertainty. Softmax is applied to each $\boldsymbol{c_i}$ to enforce a probabilistic nature of the coefficients c_{ij}. Once the vector $\boldsymbol{c_i}$ is computed, a linear combination of the input capsules $\boldsymbol{u_{j|i}}$ and the routing coefficients $\boldsymbol{c_i}$ is computed and a non-linear squash function is applied to each of the output vectors $\boldsymbol{s_j}$. The routing coefficients get updated as $c_{ij} \leftarrow c_{ij} + \boldsymbol{u_{j|i}} \cdot \boldsymbol{v_j}$ where $\boldsymbol{v_j}$ is the squashed linear combination from the previous step. In the end $\boldsymbol{v_j}$ is returned as the output vector. The dot product in this algorithm is used as a means to measure the similarity between two capsules or vectors. If they highly agree with one another the output of the dot product is positive and large while if they disagree greatly the output is negative and large corresponding to an increase or decrease in the routing weight respectively. Figure 3.3 shows a simple illustration of the weight update process in the routing by agreement algorithm.

Since inputs and outputs of CapsNets are vectors, the non-linear functions such as ReLU and Sigmoid that are applied to traditional CNNs are no longer applicable. Instead [14] replaces these functions with a non-linear vector function titled squashing which takes in a vector as input and returns a vector output that has its direction preserved but its magnitude mapped to between zero and one. This allows the capsules to be interpreted as a probability much like the Softmax function allows scalar neurons to be interpreted as probabilities.

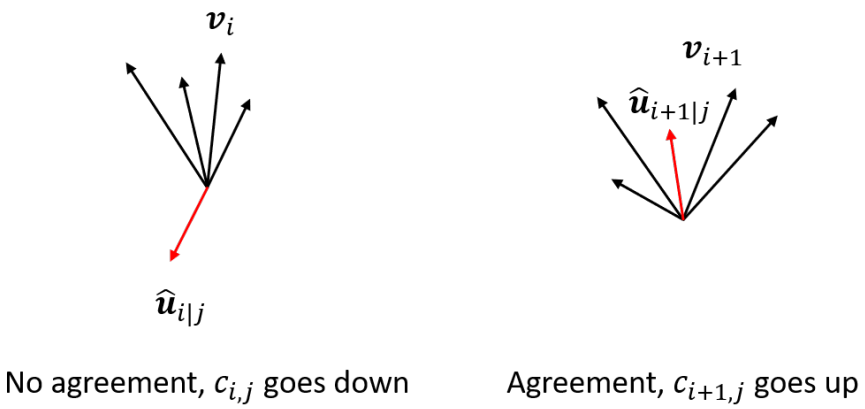


Fig. 3.3. Demonstration of routing by agreement. Left: Current capsule $\hat{u}_{i|j}$ does not agree with higher level capsules causing the routing weight to decrease. Right: Current capsule $\hat{u}_{i+1|j}$ does agree with higher level capsules causing the routing weight to increase.

As all convolutional neural networks require a loss function to optimize during training, so to does CapsNet. In CapsNet two losses are utilized together, a margin loss that pronounces the existence of a class within an image and a reconstruction loss which is used to encode the instantiation parameters of a particular class. In [14] a mask is created for the correct class in an image and is then used to reconstruct the output of the class capsule. This allows the reconstruction to regularize the margin loss while providing some insight into what type of information the capsules are representing. The total loss used for training is the sum of the margin and reconstruction losses with the reconstruction loss being scaled down considerably, by a factor of 0.0005 in the original implementation.

An interesting aside that was investigated in [14] is what the capsules themselves are representing. By tweaking each individual dimension of a capsule by a small amount, the authors were able to utilize the decoder network to generate images of the tweaked capsules and found that they are highly interpretable. Each dimension appeared to represent some feature of the input image, such as the thickness or skew

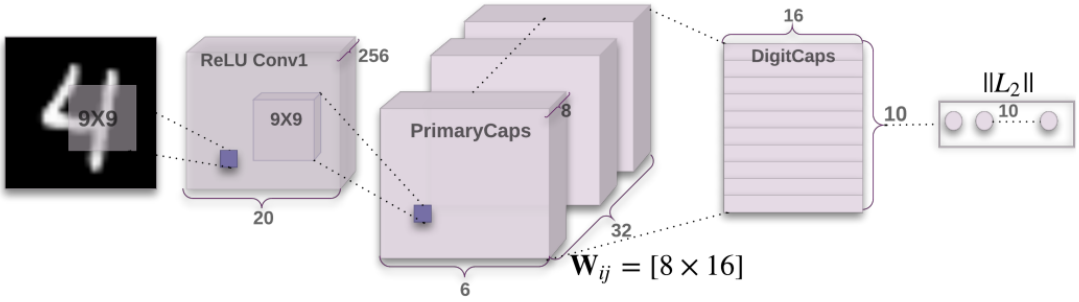


Fig. 3.4. CapsNet architecture as presented in [14].

of lines, or in some class specific cases the diameter of the loops present in digits such as 6 and 0.

The capsule network also demonstrated the ability to segment highly overlapping digits. To do this the MNIST dataset was modified by overlaying a digit on top of another digit with on average 80% overlap. CapsNet is able achieve a 5% error rate on this modified MNIST dataset in large part due to the routing by agreement algorithm; since capsules only get routed to higher level capsules if it makes sense to do so, the ambiguity between which pixels belong to which digits gets "explained away".

3.3 Matrix Capsules with EM Routing

Not long after [14] was published new work with capsules was presented in [15] in which capsules are extended to contain a logistic unit to represent the presence of an entity and also a 4×4 transformation matrix. This new entity is titled a matrix capsule and a variation of the original capsule network architecture is presented to classify images from the smallNORB dataset, a collection of images of 5 different toys at different azimuths, elevations, and lighting conditions.

The modified capsule network architecture consists of a regular convolutional layer with 32 channels, a primary capsule layer, and two convolutional capsule layers instead

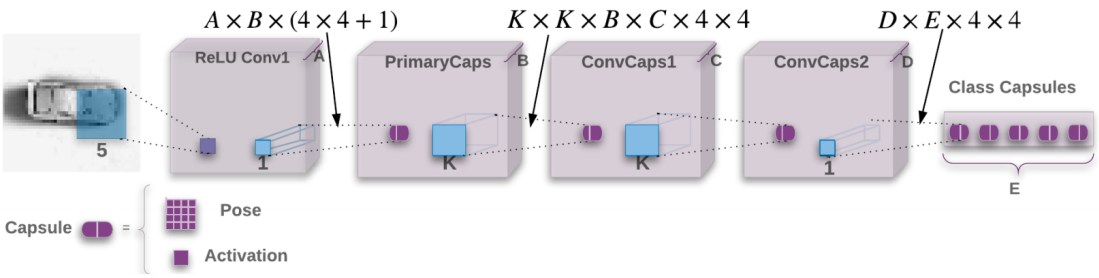


Fig. 3.5. The CapsNet architecture presented in [15].

of one as in [14]. At the output lies the class capsule layer which outputs the five different classes and their corresponding transformation matrix. This architecture is depicted in Figure 3.5.

Aside from the addition of an extra convolutional capsule layer in this implementation, the introduction of the 4×4 transformation matrix required modifications to the routing by agreement algorithm. Computation of this matrix during training is done iteratively using a variation of the expectation maximization method. Traditionally this method is used to fit points into a mixture of Gaussian models by alternating calls to the E-step and M-step. In the context of matrix capsules, the E-step determines the the assignment probability r_{ij} of each datapoint to a parent capsule, while the M-step recalculates the Gaussian models' values based on r_{ij}. This, as in the original CapsNet, takes place over three iterations for each connection between capsule layers.

This work aimed to demonstrate that while viewpoint changes cause complicated effects on pixel intensities, they exhibit simple linear effects on the pose matrix for any particular object. The authors go on to demonstrate that this CapsNet architecture achieves a state-of-the-art test error rate of 2.2% on the smallNORB dataset while only requiring ∼68K trainable parameters.

4. METHODS

4.1 Environment

The work in this thesis was conducted utilizing a Windows 10 Pro based machine containing an Intel Core i7-7700K CPU @ 4.5GHz, Nvidia GTX-1080ti @ 1708Mhz with 11GB GDDR5X memory, and 32GB DDR4 RAM at 2166MHz. Software libraries utilized include but are not limited to Python 3.7.1, PyTorch 1.0.1, Numpy 1.15.4, Scikit-Learn 0.20.2, Spyder 3.3.3, and Matplotlib 3.0.3.

4.2 Data Preprocessing

Data for the work conducted in this thesis comes from the xView [3] dataset, a collection of high resolution overhead satellite images. This data was collected and labeled primarily for the advancement of object detection techniques in overhead satellite imagery, however in this work the task of detecting and localizing objects within the images is not explored. Instead the provided labels are used to crop out objects from images to explore the use of capsule networks as a classifier of those objects. Since the images are all taken from an overhead viewpoint, the main variations between objects of the same class are rotations and translations within a particular scene. Capsule networks have proved effective in recognizing objects even after undergoing transformations while also requiring significantly less data to train overall which provides the motivation for their use in this work.

Each of the images in the xView dataset has been sampled to cover a $1km^2$ area of the earth's surface with 60 total classes annotated. These annotated classes were cropped out from their parent images and placed within their own class folder. It was noted both from the paper as well as from the action of cropping that their is a

large imbalance between the number of instances within each class. The most prevalent objects are small cars and buildings as much of the data is taken from densely populated areas where these two objects exist in abundance. Training convolutional neural networks with datasets exhibiting this type of class imbalance can lead to skewed results since the network may learn to classify each image as the type with the most instances during training. This can provide a high classification accuracy yet prove unusable in practice since the network will always provide the same output regardless of the input image that is passed through it.

During cropping of the objects it was discovered that many of the annotations extend beyond the dimensions of the original image. To combat this the cropped instances were clipped to lie within the bounds of the original image, at which point it was discovered that some of the crops had a height or width of only a few pixels and the object that is contained within the crop was not perceivable. Instances of this nature, in particular those that contained a height or width of less than 3 pixels in total, were excluded from the dataset that was ultimately used during experimentation.

Once each of the object classes was cropped out from their parent images, each cropped image was resized to $30 \times 30 \times 3$. This is done to ensure that the input the the network is of consistent size as well as reduce the amount of physical memory that is required to process batches of these crops. It is noted that the act of resizing the cropped images not only degrades their quality but also alters the aspect ratio of the objects contained within each crop. Because there are 60 total classes each with varying sizes and aspect ratios there is no obvious way to address this issue.

During training, images have random horizontal and vertical flips applied within each batch as well as a random cropping of 30×30 with at most a 2 pixel shift an any direction. This is done to discourage overfitting since the dataset is much smaller than what would typically be used to train a CNN. Images that are held out for testing have no transformations applied and are fed through each network configuration as is.

4.3 Baseline CNN

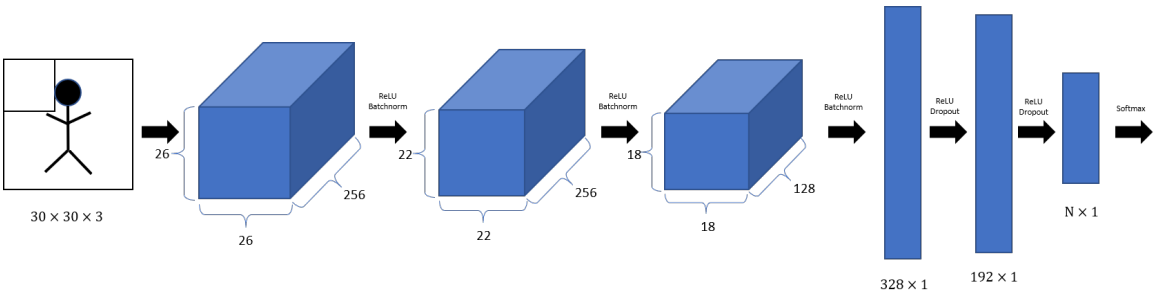


Fig. 4.1. Baseline CNN architecture.

In an effort to gauge performance of capsule networks versus traditional CNNs on the xView dataset, a traditional style CNN was implemented as a baseline network against which the capsule network architectures may be compared. This network was designed to have a comparable number of trainable parameters to the CapsNet implementations.

The baseline CNN consists of three convolutional layers with 256, 256, and 128 channels respectively. Each convolutional layer utilizes a kernel size of 5 $\times$ 5 and a stride of 1. The output of the last convolutional layer is reshaped and fed to the first of three fully connected layers of size 328, 192, and $N_{classes}$. Batch normalization is applied to each of the convolutional layer outputs and the ReLU non-linearity is used at the output of every layer except the final output layer, where the softmax function is used to map the output to each class with cross-entropy loss. Dropout is also applied between each pair of fully connected layers. This configuration is depicted in Figure 4.1.

The baseline CNN was trained until convergence, or 500 epochs, utilizing the Adam optimizer with a batch size of 128, a learning rate of 0.001, and a learning rate decay of 0.99 to avoid decreasing the learning rate too quickly given the relatively small number of training images.

4.4 CapsNet Implementations

4.4.1 Configuration 1

The first proposed model of a capsule network for object recognition in overhead satellite imagery closely resembles that of the original model in [14]. The input images are first fed through a convolutional layer with 256 channels and a kernel size of 9×9. ReLU applied at the output of this layer as well as batch normalization before being fed to a primary capsule layer. The primary capsule layer consists of 32 channels of 8-dimensional capsules which results in 1568 total capsules at its output ($7 \times 7 \times 32$). Each of these outputs is fed to a second capsule layer where classification will take place. This classification layer contains a single 16-dimensional capsule for each class in the dataset. The routing by agreement algorithm will only take place between the two capsule layers.

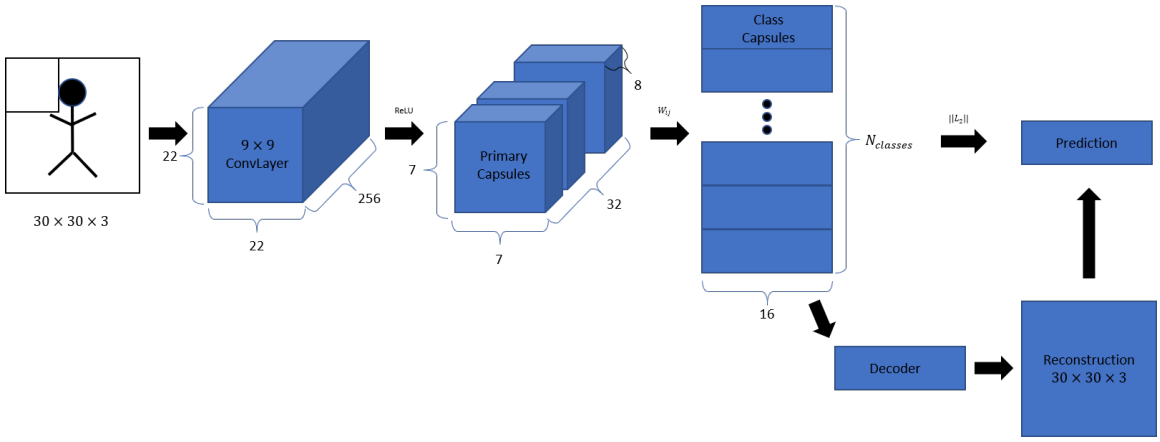


Fig. 4.2. Full capsule network configuration 1.

A decoder network is also utilized in this architecture to provide an image reconstruction from the learned features. This subnetwork consists of 3 fully connected layers with ReLU applied at the output. The final layer's output foregoes ReLU for sigmoid and is reshaped to match the dimensions of the original input image. The error between this reconstructed image and the original input is added to the training

loss although it is scaled down considerably to allow the margin loss to dominate during training. Figure 4.2 provides a graphical illustration of the network configuration.

This network is trained for 500 epochs utilizing the Adam optimizer with a batch size of 128, learning rate of 0.001, and a learning rate decay of 0.99. Two experiments are conducted using this network, the first of which involves training utilizing only 4 classes from the xView dataset. These four classes are aerial vehicles, namely *Cargo Plane*, *Fixed Wing Aircraft*, *Helicopter*, and *Small Aircraft*. There are only ∼1200 images between these four classes which allows for gauging both the capsule network and baseline CNN's response to a small training dataset as well as their ability to distinguish between very similar classes. Figure 4.3 shows a sample from each of the classes in the aerial vehicles dataset.

Fig. 4.3. Sample image from each class of the aerial vehicles (Images have been scaled up for clarity and may contain deformations not found in the dataset used for training).

Due to class imbalance, weighted or stratified sampling is utilized during training to ensure the network sees approximately the same number samples from each class. ∼3% of the data is selected at random and held out for testing while the rest is used during training.

The second test using this configuration utilizes 10 of the 60 classes from the xView dataset. These classes were selected based on their number of occurrences in the dataset with each class containing ∼200 instances, as well as the fundamental differences between each class of objects. All training hyper-parameters are kept the same as in test one and stratified sampling is again used during training with ∼10%

of the data held out for testing. A sample image from each of the classes in this dataset is shown in Figure 4.4.

Fig. 4.4. Sample image from each class of the 10 class training set. TOP (left to right): *Aircraft Hangar*, *Barge*, *Cement Mixer*, *Container Crane*, *Container Ship*. BOTTOM (left to right): *Crane Truck*, *Engineering Vehicle*. *Ferry*, *Flat Car*, *Helipad*.

4.4.2 Configuration 2

The second configuration of a capsule network in this work is created by adding another convolutional capsule layer between the primary capsule layer and the classification layer. This is done in an effort to gauge if a more complex network results in better performance overall. A decoder network is also utilized however it remains unchanged from configuration 1.

The image is fed through a convolutional layer with 256 channels and a kernel size of 9×9 with ReLU applied at the output. This output is fed to the primary capsule layer which similar to the first model consists of 32 channels of 8-dimensional capsules. These outputs are then fed to a capsule layer with 30 capsules of dimension 16. The choice of outputting 30 capsules at this layer is made due to GPU memory constraints as adding even 10 more capsules results in memory overflow errors. The output of this layer is finally fed to the classification layer containing $N_{classes}$ number

of capsules of dimension 16. This network was trained in similar fashion to the first configuration, with both the 4 class and 10 class datasets being utilized. The network configuration is shown in Figure 4.5

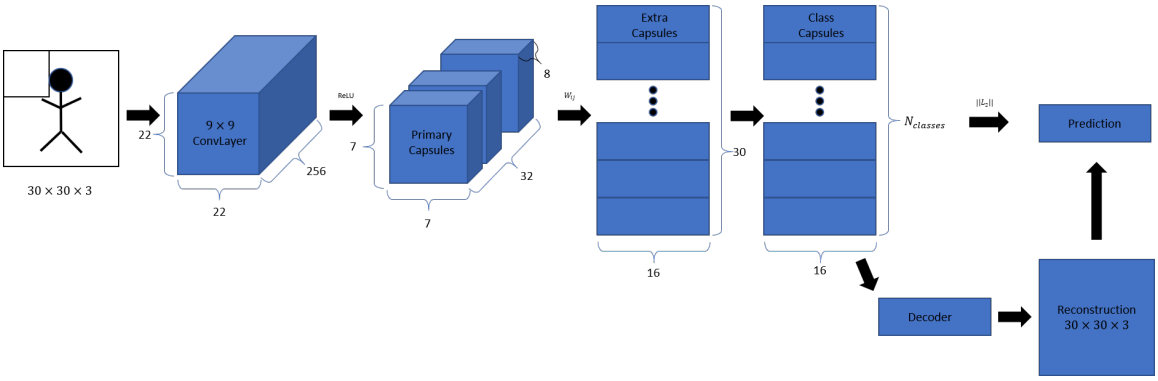


Fig. 4.5. Full capsule network configuration 2.

4.4.3 Configuration 3

A third network configuration is constructed utilizing only the primary and classification capsule layers but with a second convolutional layer placed before the primary capsule layer. Since convolutional layers are used to extract features from images adding another layer should allow for a richer feature set prior to the capsule layers. The convolutional layer contains 256 channels and a kernel size of 9×9. ReLU and batch normalization are applied at the output of each of the two convolutional layers. The decoder subnetwork as well as the primary and class capsule layers remain unchanged from configuration 1. This configuration is again trained similar to the prior two configurations. The network configuration is shown in Figure 4.6.

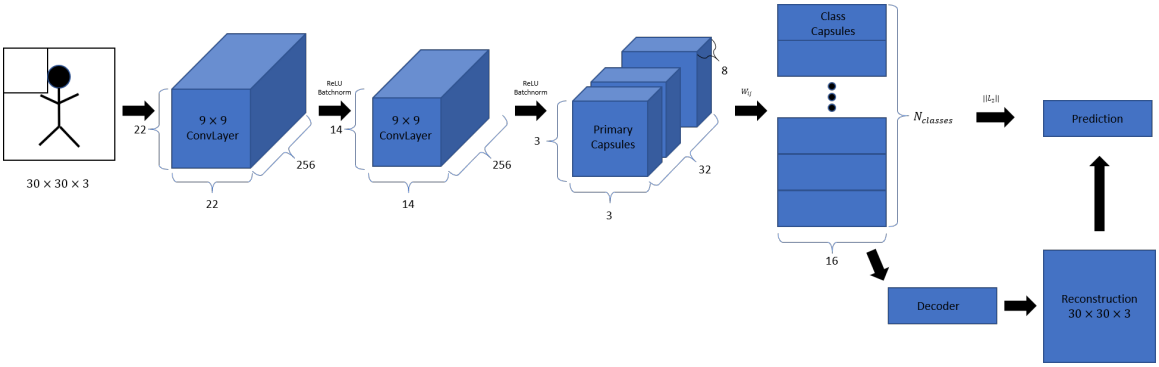


Fig. 4.6. Full capsule network configuration 3.

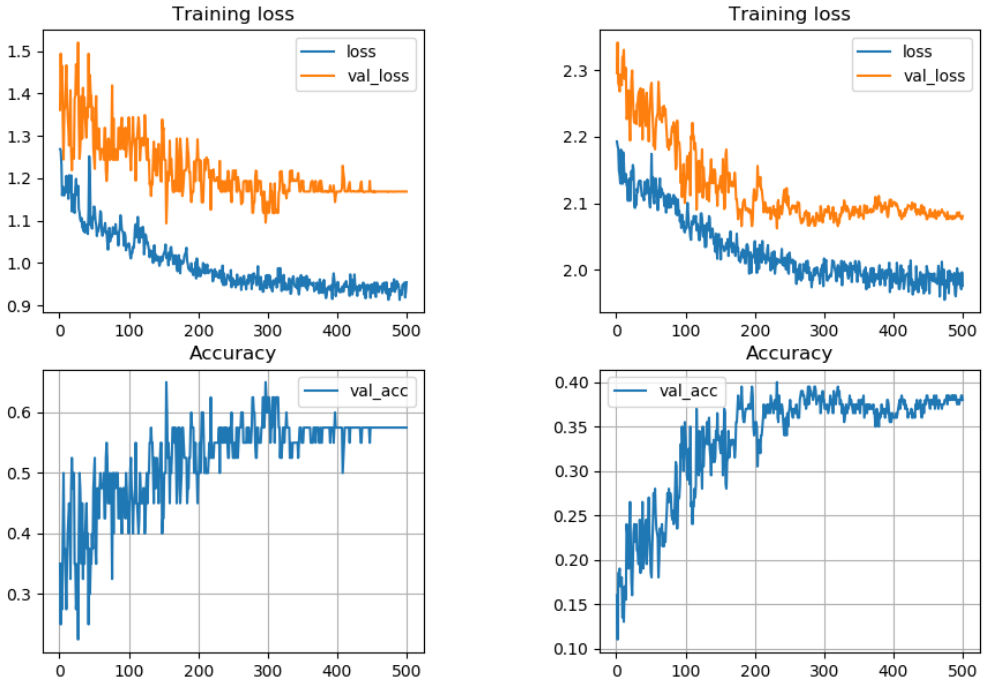
4.5 Results

In the section the results of testing each of the aforementioned capsule networks is presented and assessed. These results are compared to the baseline convolutional neural network.

4.5.1 Baseline CNN

The baseline CNN was trained and tested on two datasets with a 97%/3% and 90%/10% train/test split. Both datasets are small with approximately 1200 and 1900 total images, a much smaller amount than is typically utilized when training CNNs. This is purposely done to test both the capsule network and baseline networks response to small training datasets. The baseline was trained for 500 epochs with the best performing model then being used for testing.

Figure 4.7 shows the training graphs for the baseline CNN. On both datasets this number of training iterations appears to be enough as the training loss and validation accuracy have both plateaued prior to the epoch 500 mark.



(a) 4 class dataset.

(b) 10 class dataset.

Fig. 4.7. Training graphs for the baseline CNN.

Figure 4.8 shows the confusion matrix for the baseline CNN on the 4 class dataset. The *small aircraft* class appears to cause the most trouble for the baseline as it is mistaken as a *cargo plane* and *fixed wing aircraft* 2 and 5 times respectively out of the total 10 instances in the testing set. In total the baseline CNN achieves an accuracy of 65.0% on the 4 class dataset.

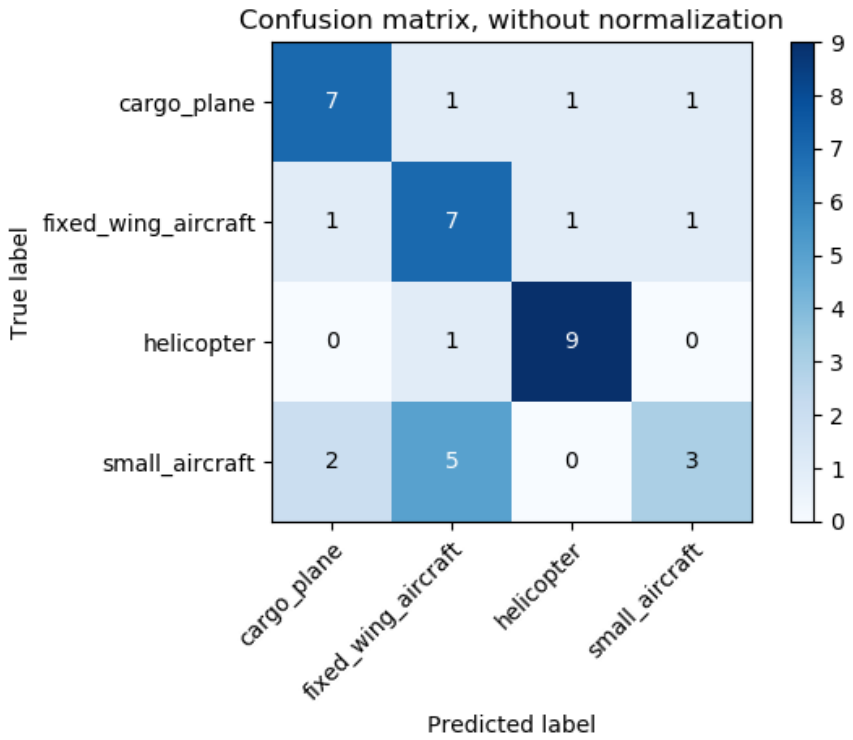


Fig. 4.8. Confusion matrix for the baseline CNN on the 4 class dataset.

The confusion matrix in Figure 4.9 demonstrates that the baseline CNN cannot scale to more object classes. While the *helipad* was correctly classified every time, only the *ferry* and *flat car* also exhibited similar performance. The rest of the predictions are sporadically placed with no obvious misclassification patterns seen. The baseline CNN achieves an accuracy of 40.0% on the 10 class dataset.

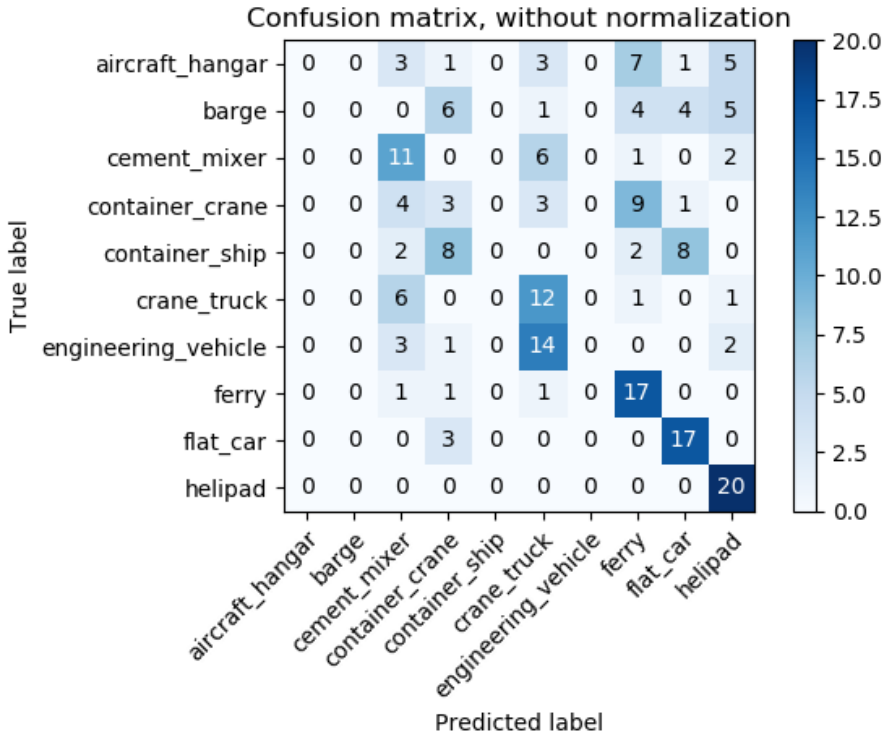
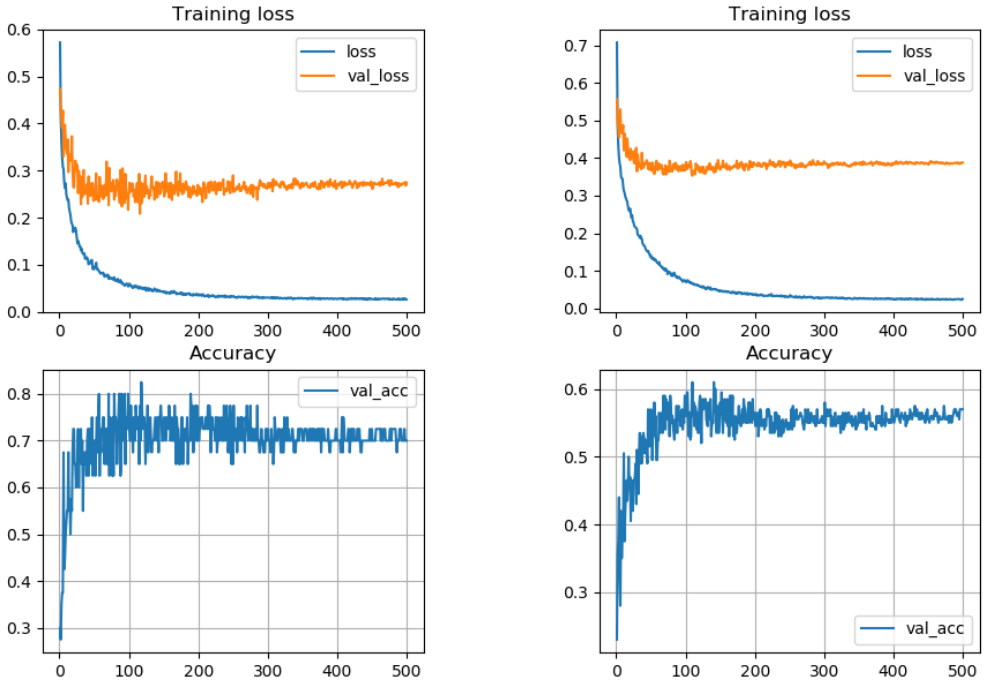


Fig. 4.9. Confusion matrix for baseline CNN on the 10 class dataset.

4.5.2 Configuration 1

The first configuration proposed was trained and tested on the same two datasets as the baseline CNN. This configuration was trained on each dataset for 500 epochs with the highest performing model then being utilized during testing. The training graphs for each dataset are shown in Figure 4.10.



(a) 4 class dataset.

(b) 10 class dataset.

Fig. 4.10. Training graphs for configuration 1.

Interestingly, the training loss appears to converge to a much lower value than the validation loss which indicates that the network is overfitting to the training data. Most commonly this requires more data to be gathered, however given the nature of the data being used for this testing acquiring more data is not feasible.

Table 4.1.
Accuracy of Baseline vs. CapsNet configuration 1.

	CapsNet 1	**Baseline**
4 class	**82.5%**	65.0%
10 class	**61.0%**	40.0%

Table 4.5.2 shows the accuracy of configuration 1 as compared to the baseline convolutional neural network. On both the 4 class and 10 class datasets the capsule network configuration outperforms the baseline CNN. It should be noted that in the 4 class dataset both the *Fixed Wing Aircraft* and the *Helicopter* classes only contain approximately 60 images to train on. The capsule network was able to distinguish between the classes fairly well even under these conditions as is highlighted by the confusion matrix in Figure 4.11.

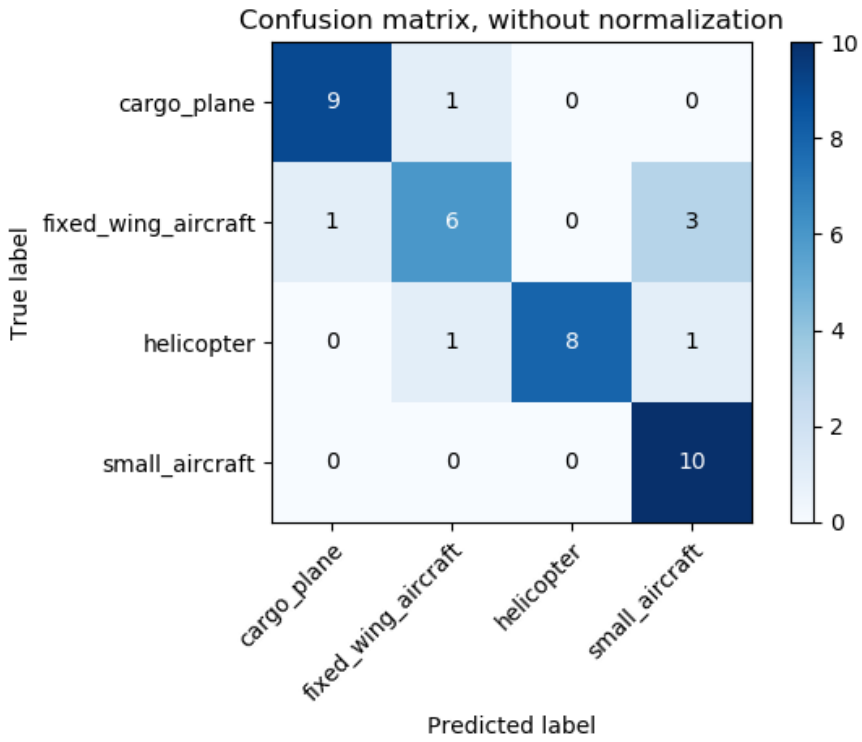


Fig. 4.11. Confusion matrix for CapsNet configuration 1 on the 4 class dataset.

The confusion matrix in Figure 4.12 also indicates that the capsule network configuration can extend to more classes quite well, with the most common mistakes occurring on objects that vary drastically between images (*container crane, crane truck*).

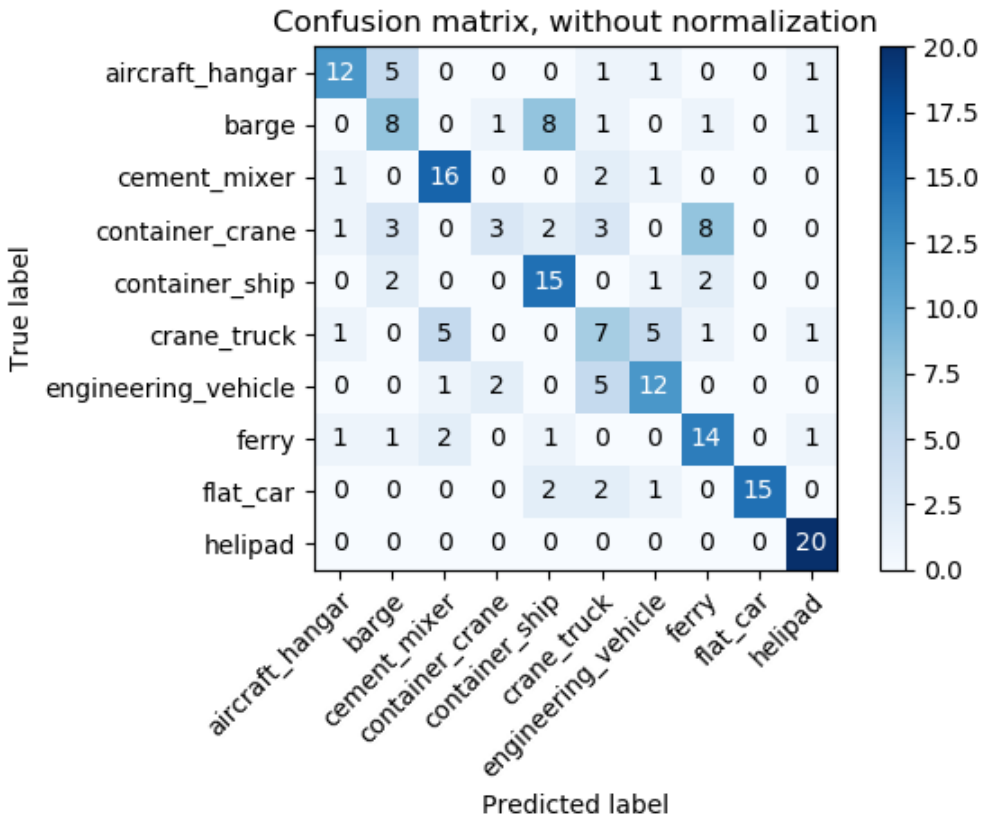


Fig. 4.12. Confusion matrix for CapsNet configuration 1 on the 10 class dataset.

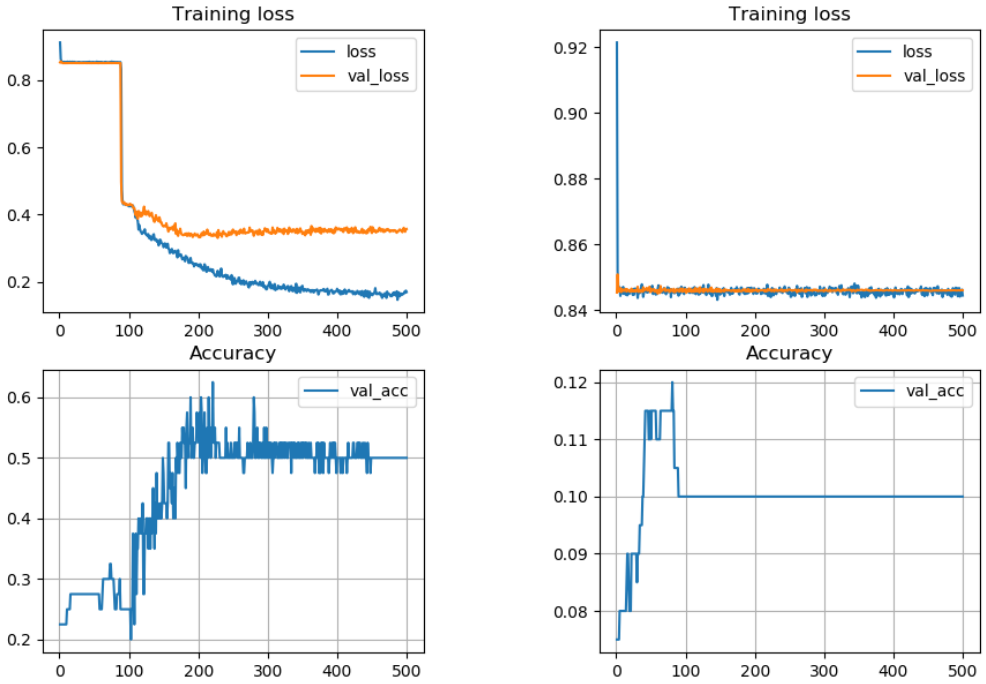
4.5.3 Configuration 2

The second configuration of a capsule network produced the most puzzling results of the three proposed architectures. Adding another convolutional capsule layer not only resulting in poorer performance in terms of accuracy, but in the case of the 10 class dataset resulted in nearly every prediction being chosen as *barge*. This is clearly shown by the confusion matrix in Figure 4.15.

Table 4.2.
Accuracy of Baseline vs. CapsNet configuration 2.

	CapsNet 2	**Baseline**
4 class	62.5%	**65.0%**
10 class	12.0%	**40.0%**

The accuracy of the second capsule network configuration dropped considerably compared to the first configuration. This may be in part due to hardware limitations where the added capsule layer is only allowed 30 capsules in total. With such a low number of capsules the network may not be able to learn enough features to accurately categorize the different classes.



(a) 4 class dataset.

(b) 10 class dataset.

Fig. 4.13. Training graphs for configuration 2.

The training graphs in Figure 4.13 also indicate an initial period of learning where the model does not improve. In the case of the 4 class dataset the model is able to overcome this period at around the epoch 90 mark, however for the 10 class dataset the model is never able to overcome this barrier and thus produces predictions that are on par with random choice. This again may be caused by the limitation of the number of capsules in the added layer. In the four class dataset the 30 capsules may be enough to construct features from the data, however extending to 10 classes requires more parameters to properly construct features from the data.

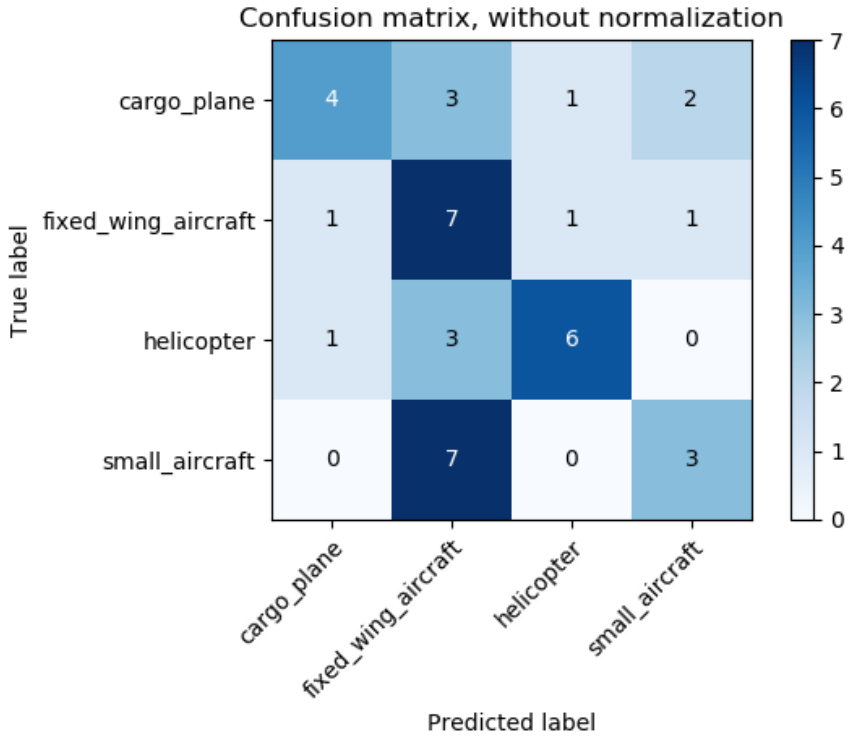


Fig. 4.14. Confusion matrix for CapsNet configuration 2 on the 4 class dataset.

The confusion matrix in Figure 4.14 offers a promising outlook on adding more capsule layers to the network. Overall the model was able to predict the correct class 62.5% of the time which, although less than configuration 1 (82.5%), may improve

provided more data is utilized during training or hyperparameters are selected in just the right way.

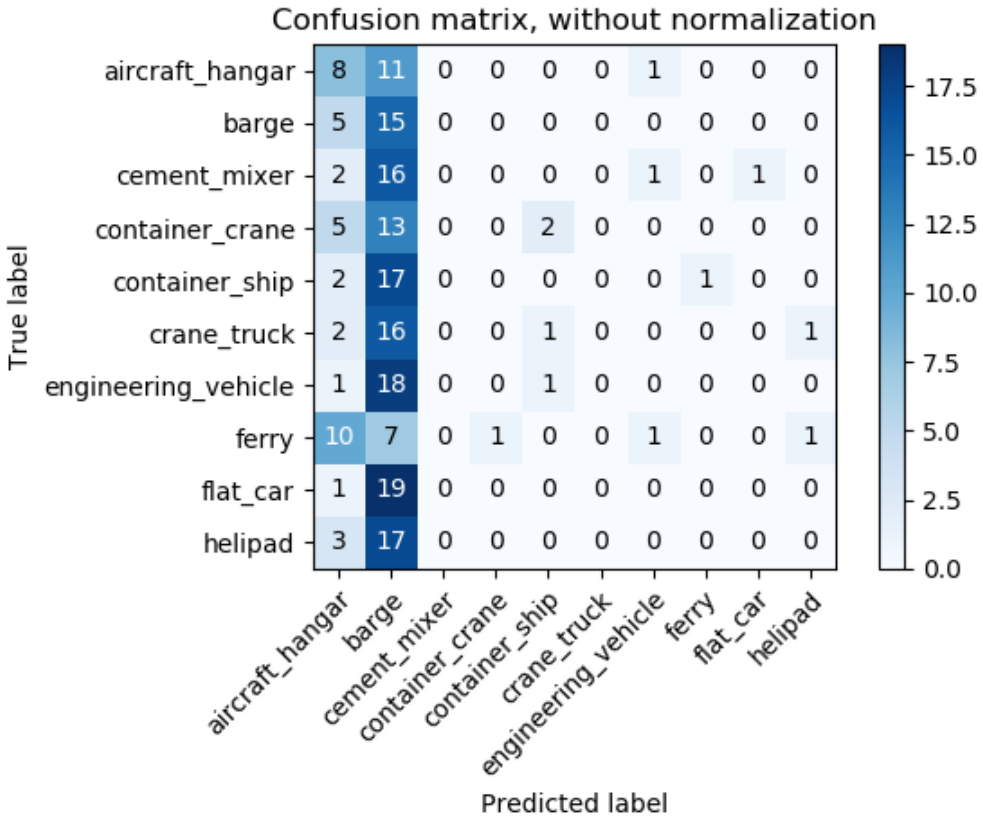


Fig. 4.15. Confusion matrix for CapsNet configuration 2 on the 10 class dataset.

Figure 4.15 shows that the 10 class capsule network was unable to learn useful features from the data. The predictions mostly appear to be of the *barge* class which represents approximately 10% of the test data. This explains the models accuracy of only 12.0% which is significantly less than configuration 1 (61.0%). If provided with more training data and hardware that allows for processing of more capsules within the added layer the model may improve, however testing this hypothesis is not currently feasible.

4.5.4 Configuration 3

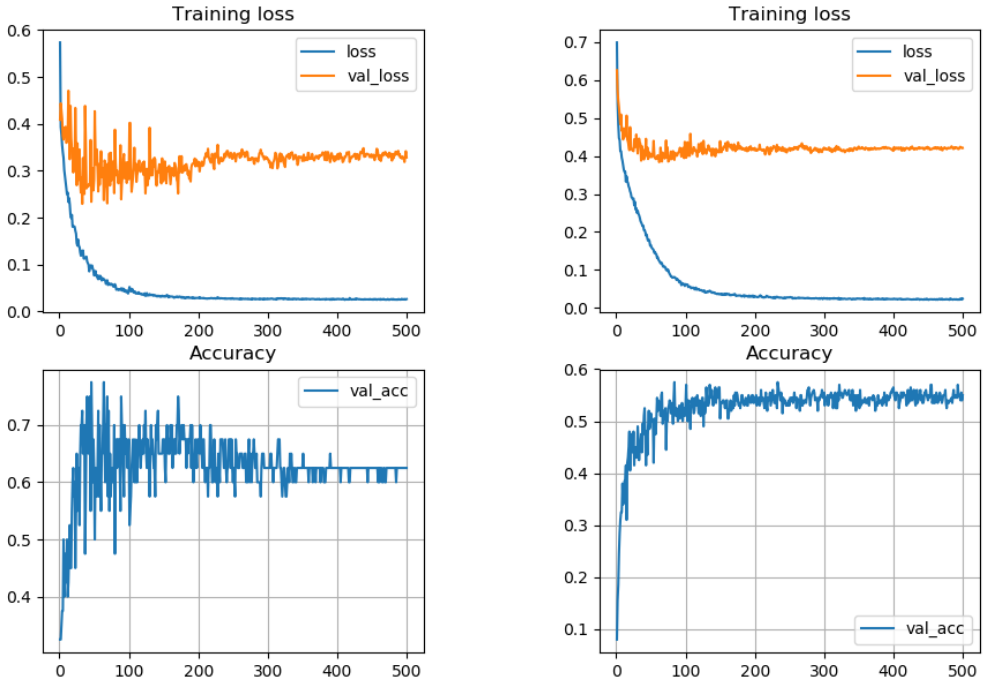
The third capsule network configuration created in this work involved adding a second convolutional layer after the first. Since convolutional layers are used to extract features from data adding a second layer should in theory allow for more complex features to be extracted from each image prior to being passed the the primary capsule layer. The accuracy of this network configuration was actually lower than configuration 1, however it performed better much better than configuration 2.

Table 4.3.
Accuracy of Baseline vs. CapsNet configuration 3.

	CapsNet 3	**Baseline**
4 class	**77.5%**	65.0%
10 class	**57.5%**	40.0%

Table 4.5.4 shows the accuracy of the third capsule network configuration versus the baseline CNN. Again the capsule network is able to outperform the baseline on both the 4 class and 10 class datasets by a considerable margin.

The training graphs for this configuration are shown in Figure 4.16. Interestingly enough the capsule network again exhibits overfitting to the training data on both of the datasets. This may be due in part the the limited amount of training data as well as the hyperparameter selection during training.



(a) 4 class dataset.

(b) 10 class dataset.

Fig. 4.16. Training graphs for configuration 3.

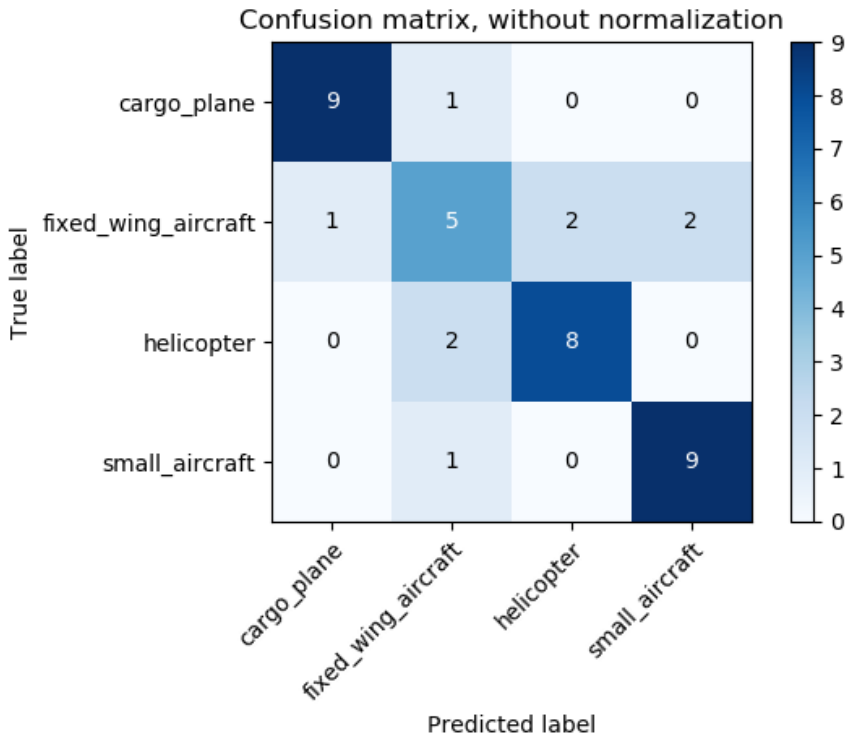


Fig. 4.17. Confusion matrix for CapsNet configuration 3 on the 4 class dataset.

The confusion matrix in Figure 4.17 demonstrates that this network configuration is able to categorize the 4 different classes quite well, with the exception being the *fixed wing aircraft*. This may be due to the inconsistencies between images within this class as *fixed wing aircraft* encompasses many different types of airplanes each with their own subtle differences. There are also only 64 images of this class used during training which may not be quite enough to learn robust features from.

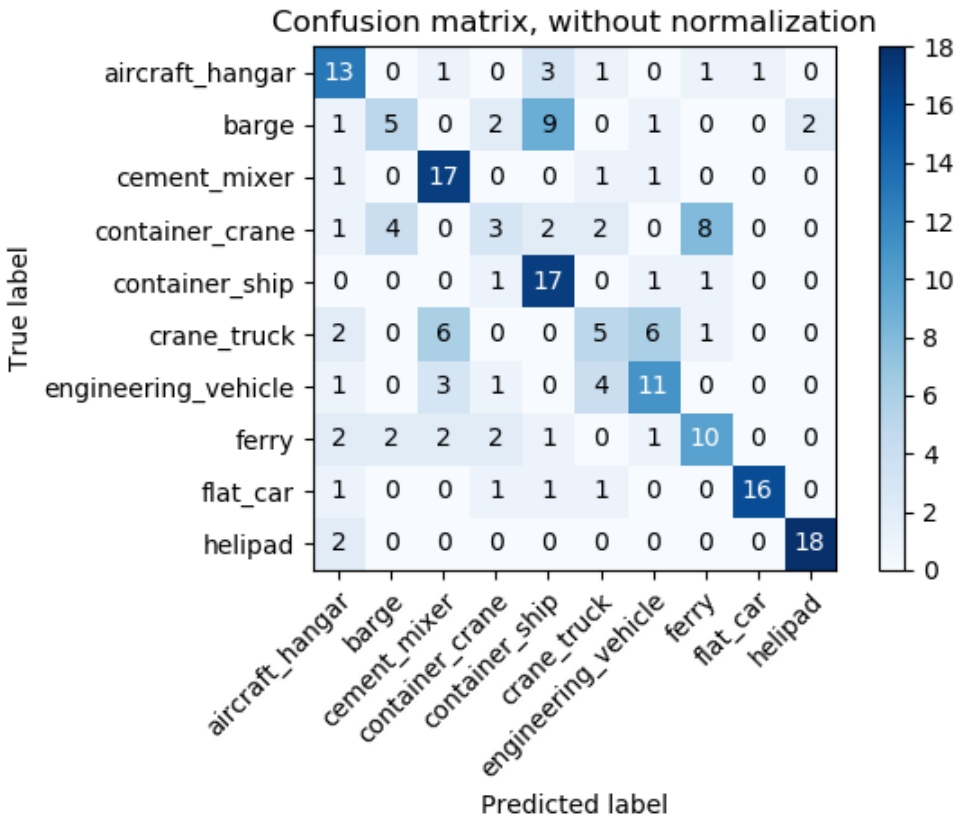


Fig. 4.18. Confusion matrix for CapsNet configuration 3 on the 10 class dataset.

The final confusion matrix in Figure 4.18 shows that adding a convolutional layer to the capsule network architecture can in fact extend to a more diverse dataset. While accuracy falls compared to the first configuration the classification of the 10 different objects is still quite robust with the misclassifications occuring primarily between objects with highly overlapping features, for example *barge* is classified as a *container ship* 9 times whereas the unique classes such as *helipad* is only misclassified twice as an *aircraft hangar*.

5. SUMMARY

5.1 Conclusion

In this thesis capsule networks were proposed as a method for classifying objects taken from overhead satellite imagery given the underlying spatial similarities within each class. The capsule networks proposed are shallow architectures that not only perform well with a limited amount of training data but also perform consistently better than a regular convolutional neural network of approximately the same depth. The capsule networks proposed are able to distinguish between classes that are very similar such as aerial vehicles while also showing promising performance on a more general set of object classes.

While the task of localizing and extracting objects from satellite imagery is left for future research, this work provides a promising method for classifying said objects even in the presence of limited training data. This is key considering there is guaranteed to be a great imbalance between object types within these images. The ability to use a small collection of these objects to produce a viable model is a key step towards the full automation of satellite imagery processing which will in turn open the door to a wide variety of applications utilizing this type of data.

5.2 Recommendations and Future Work

While this work demonstrated that capsule networks can be used to classify objects in the domain of satellite imagery, there are many areas where improvements can be made.

Capsule networks are considerably more difficult to train compared to their traditional CNN counterparts. This is in part due to the routing by agreement algorithm

that takes place between capsule layers as there is currently no simple method to optimize this operation. There is also a practical limitation involved since the use of vector inputs and outputs within capsule layers requires significantly more memory to process. This pushes current hardware to its limits and makes scaling a capsule network up to a more significant number of layers impossible. While one workaround is to offload some of the processing to CPU/RAM, this often comes with a huge performance hit in terms of training and testing speed.

While the network architectures proposed in this work exceeded the performance of a baseline CNN of approximately the same complexity, it would be of great interest to see how these architectures perform compared to larger, more advanced network architectures such as VGG16 and ResNet in the realm of satellite imagery.

This work only utilized two small datasets consisting of 4 and 10 classes respectively. The xView dataset contains labels for 60 different object classes. Experimenting with a model utilizing each of the 60 classes would be of great benefit to gauge how well capsule networks can scale to larger and more complicated datasets. This would require more advanced hardware to process the larger model, however it is worth investigating as the number of object classes contained within satellite imagery can be further expanded to much more than 60.

Satellite imagery in the form of RGB color images is used in this work, however satellite imagery is often collected using many spectral bands at varying resolutions. Testing a capsule network on satellite data outside of the optical range, such as synthetic aperture radar imagery, could prove beneficial and lead to better performance overall while also providing other benefits such as immunity to weather conditions and day and night cycles. This however requires obtaining and annotating a large amount of data prior to experimentation.

Lastly while classification of images is an interesting and difficult area of research on its own, the logical next step seems to be extending this work to both localization and classification of objects within satellite imagery, or object detection.

REFERENCES

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[2] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51.

[3] D. Lam, R. Kuzma, K. McGee, S. Dooley, M. Laielli, M. Klaric, Y. Bulatov, and B. McCord, "xview: Objects in context in overhead imagery," *CoRR*, vol. abs/1802.07856, 2018. [Online]. Available: http://arxiv.org/abs/1802.07856

[4] L. Roberts, "Pattern recognition with an adaptive network," in *Proceedings of the institute of radio engineers*, vol. 48, no. 3, 1960, pp. 398–398.

[5] A. Andreopoulos and J. Tsotsos, "50 years of object recognition: Directions forward," *Computer Vision and Image Understanding*, vol. 117, p. 827891, 08 2013.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[7] D. Rumerlhart, G. Hinton, and R. Williams, "Learning internal representations by backpropagation errors," *Nature*, vol. 323, no. 99, pp. 533–536, 1986.

[8] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] M. J. Friedel, M. Buscema, L. E. Vicente, F. Iwashita, and A. Koga-Vicente, "Mapping fractional landscape soils and vegetation components from hyperion satellite imagery using an unsupervised machine-learning workflow," *International Journal of Digital Earth*, vol. 11, no. 7, pp. 670–690, 2018. [Online]. Available: https://doi.org/10.1080/17538947.2017.1349841

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[12] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features." in *iccv*, vol. 99, no. 2, 1999, pp. 1150–1157.

[13] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[14] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in neural information processing systems*, 2017, pp. 3856–3866.

[15] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing," 2018.

[16] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

APPENDIX

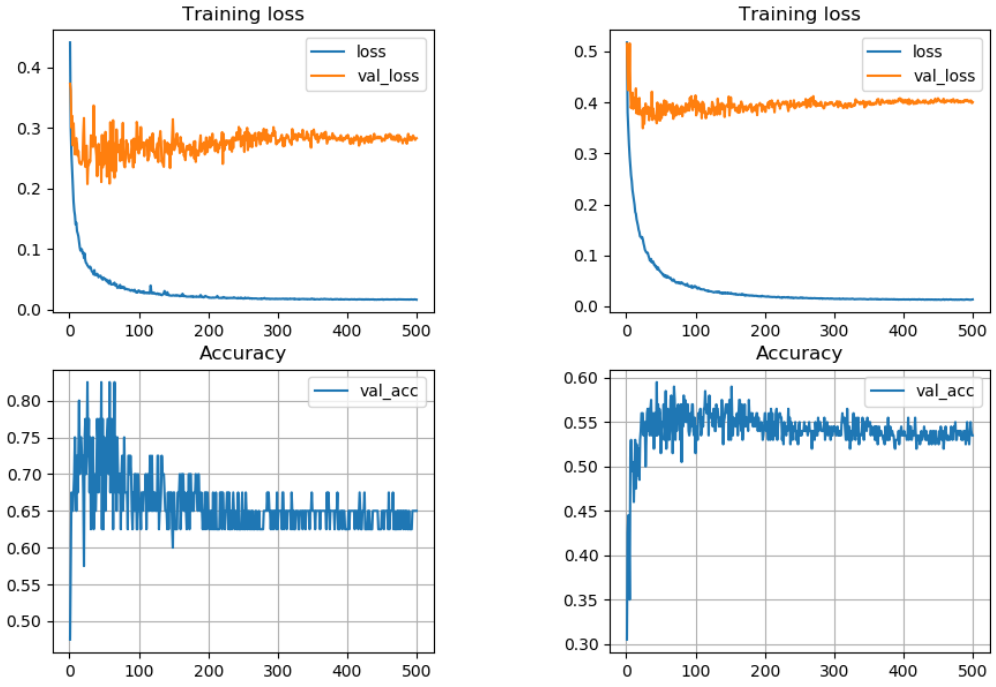
A. EFFECTS OF SYNTHETICALLY CREATING MORE DATA

Two datasets were utilized during the implementation of this work, one being a set of aerial vehicles with 4 total classes and the other being a random selection of 10 classes of the 60 from the xView dataset. These selections were made purposely to test both the capsule network's and baseline CNN's robustness when dealing with small datasets with limited examples of each object class. Utilizing basic image transformations these datasets can be expanded to include more instances of each object class.

In an effort to gauge the benefit of acquiring more data, both of the training sets for each dataset were used to generate more samples of each object class. Rotations were applied to each image in the training sets at 90°, 180°, and 270° with each rotation being added to the training set. These angles were chosen to avoid distorting images since rotating by any other angle will require interpolation and the addition of new pixel data along the edges of each image. These transformations result in each training set increasing in size by a factor of 4 since each image will provide 3 new images at the specified rotations.

A.0.1 CapsNet Config 1

Since one of the main benefits of capsule networks is their ability to retain information about the spatial relationships between objects in each image, the addition of these transformed training instances should have little to no effect on the overall accuracy as the network should be able to identify that the new images are simply items that it has seen before rotated by some amount.



(a) 4 class extended dataset.

(b) 10 class extended dataset.

Fig. A.1. Training graphs for CapsNet config 1 on extended training datasets.

The training graphs for CapsNet configuration 1 on the new training datasets are shown below in Figure A.1. As expected the network did not improve by any significant amount with the best models achieving 82.5% and 59.5% on the 4 class and 10 class datasets respectively. These values are nearly identical to training with the default datasets and the slight drop in accuracy of the 10 class model may be solely from the stochastic nature of the SGD algorithm. The confusion matricies for these two models are shown in Figure A.2 and A.3 respectively.

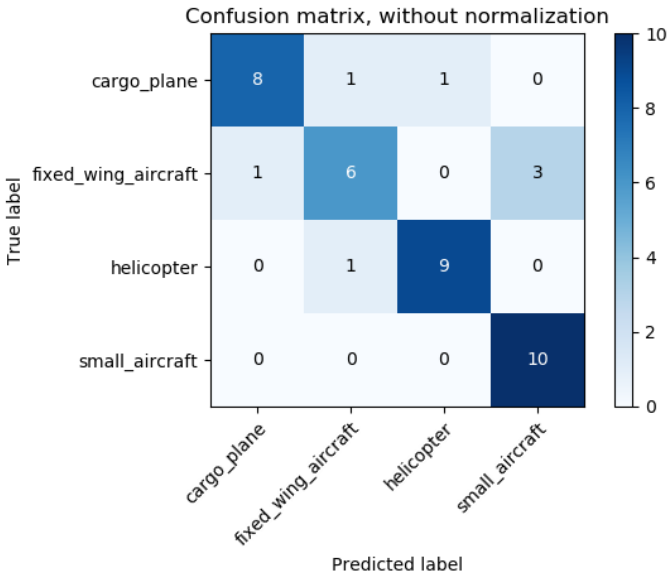


Fig. A.2. Confusion matrix for CapsNet config 1 on the extended 4 class dataset.

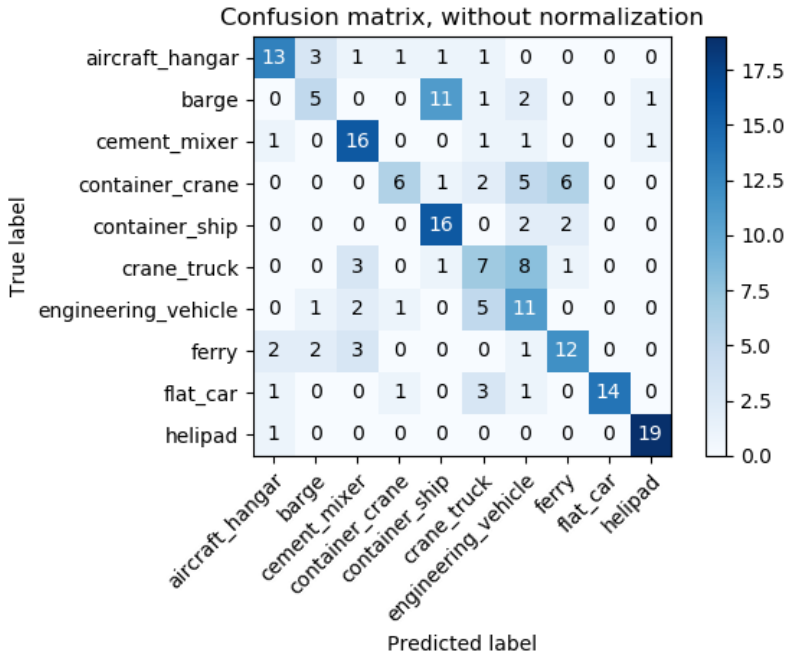
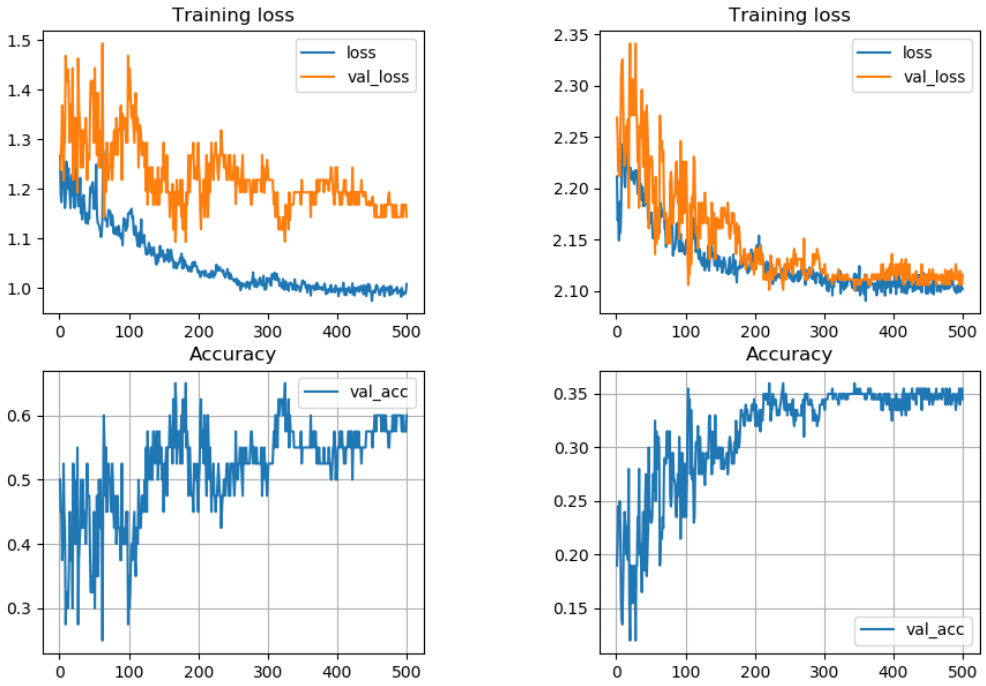


Fig. A.3. Confusion matrix for CapsNet config 1 on the extended 10 class dataset.

A.0.2 Baseline CNN

Convolutional neural networks are notorious for requiring copious amounts of data to properly train. This coupled with the fact that our baseline CNN was designed to contain roughly the same number of parameters as the CapsNet implementations severely limits its ability to perform well on the two datasets used in this work. By incorporating more training data into these datasets the baseline CNN should in theory see a slight increase in accuracy as the extra data should allow more robust features to be learned during training.



(a) 4 class extended dataset.

(b) 10 class extended dataset.

Fig. A.4. Training graphs for baseline CNN on extended training datasets.

Figure A.4 shows the training graphs for the baseline CNN on the new extended 4 class and 10 class datasets respectively. Interestingly, both the 4 class and 10 class model did not seem to be affected at all by the extra data utilized during training as the accuracy remained identical at 65.0% for the 4 class dataset and dropping to

36% for the 10 class dataset. It is possible that the network configuration itself is limiting the models performance as it is shallow with only 3 convolutional layers and 3 fully connected layers. The best performing CNNs contain upwards of 11 layers with various connections taking place between each layer and even feedback connections existing from early layers to deeper layers.

The 10 class model was allowed an extra 500 epochs to train due to its drop in performance with the extra training data. This resulted in the model once again reaching 40% accuracy however it did not go beyond this value further confirming that the network architecture itself is limiting the models performance. The confusion matricies for the baseline CNN trained with more data are shown in Figures A.5 and A.6 respectively.

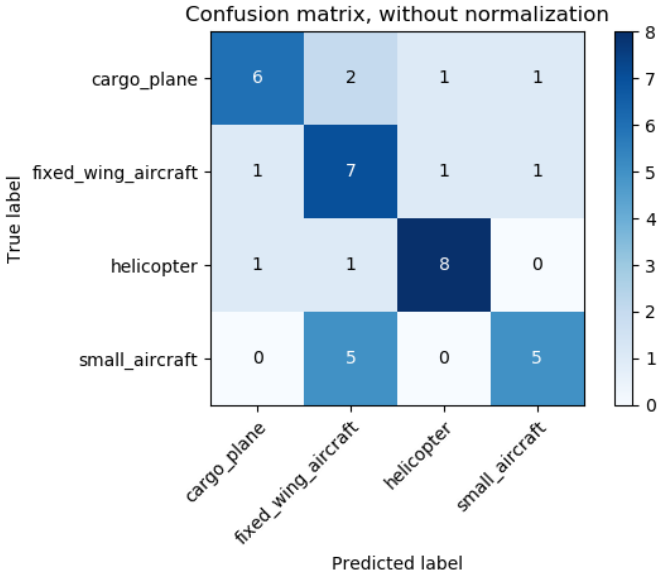


Fig. A.5. Confusion matrix for baseline CNN on extended 4 class dataset.

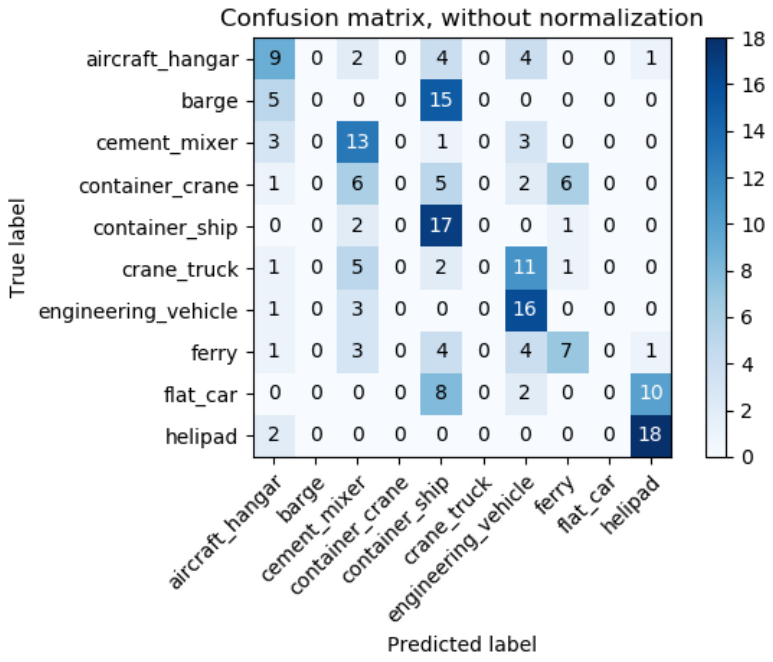


Fig. A.6. Confusion matrix for baseline CNN on extended 10 class dataset.