PEDESTRIAN NAVIGATION USING ARTIFICIAL NEURAL NETWORKS AND CLASSICAL FILTERING TECHNIQUES

THESIS

David J. Ellis, Captain, USAF

AFIT-ENG-MS-20-M-018

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-018

Pedestrian Navigation using Artificial Neural Networks and Classical Filtering Techniques

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

David J. Ellis, B.S.E.E.

Captain, USAF

March 19, 2020

AFIT-ENG-MS-20-M-018

Pedestrian Navigation using Artificial Neural Networks and Classical Filtering Techniques

THESIS

David J. Ellis, B.S.E.E.
Captain, USAF

Committee Membership:

Joseph A. Curro, Ph.D
Chair

Aaron J. Canciani, Ph.D
Member

Clark N. Taylor, Ph.D
Member

AFIT-ENG-MS-20-M-018

Abstract

The objective of this thesis is to explore the improvements achieved through using classical filtering methods with Artificial Neural Network (ANN) for pedestrian navigation techniques. ANN have been improving dramatically in their ability to approximate various functions. These neural network solutions have been able to surpass many classical navigation techniques. However, research using ANN to solve problems appears to be solely focused on the ability of neural networks alone. The combination of ANN with classical filtering methods has the potential to bring beneficial aspects of both techniques to increase accuracy in many different applications. Pedestrian navigation is used as a medium to explore this process using a localization and a Pedestrian Dead Reckoning (PDR) approach.

Pedestrian navigation is primarily dominated by Global Positioning System (GPS) based navigation methods, but urban and indoor environments pose difficulties for using GPS for navigation. A novel urban data set is created for testing various localization and PDR based pedestrian navigation solutions. Cell phone data is collected including images, accelerometer, gyroscope, and magnetometer data to train the ANN. The ANN methods are explored first trying to achieve a low root mean square error (RMSE) of the predicted and original trajectory. After analyzing the localization and PDR solutions they are combined into an extended Kalman Filter (EKF) to achieve a 20% reduction in the RMSE. This takes the best localization results of 35m combined with under performing PDR solution with a 171m RMSE to create an EKF solution of 28m of a one hour test collect.

Acknowledgments

I would like to express my sincere gratitude to my advisor for giving me the opportunity to pursue Neural Network development. His continued guidance and knowledge pushed me forward whenever I needed a sounding board or research was stalling. He was constantly engaged in my work pushing my knowledge and helping me to succeed. I would also like to thank my committee chairs for taking the time to provide invaluable feedback and questions about this research that pushed me to expand my thought process on the topics in this research. Thank you to the ANT center lab group for providing hours of discussion and games to distract me from the daunting task of this research.

I would also like to thank my parents, brothers, and sister for listening to me talk about my project and always being a source of light hearted conversation. To my sister who passed away this year, thank you for all your support, advice, and fun times while growing up. You were an inspiration to all of us. Finally I would like to thank my wife and kids for supporting me throughout this program and sacrificing so much family time. You all picked up everything I dropped off when working long hours to complete this thesis and helped push me through. I loved all of the time we did spend together and it provided a much needed reprieve from research. Your support has been amazing and I wouldn't have wanted to do this research without you by my side.

Table of Contents

Page

List of Figures

Figure Page

Figure Page

Figure Page

List of Tables

Pedestrian Navigation using Artificial Neural Networks and Classical Filtering Techniques

I. Introduction

1.1 Pedestrian Navigation

Navigation is an integral part of many technologies civilization has become reliant on it. Current technologies rely heavily on Global Positioning System (GPS) for a relatively precise solution to this problem. However, GPS signals can be challenged or denied by obstacles or buildings in indoor or urban environments. These gaps have led to a variety of proposed solutions for navigation in both of these environments. Some of the approaches that have been researched include Bluetooth Low Energy (BLE) [3], Wireless Fidelity (WIFI) [4], radio frequency (RF) [5], and image based approaches [6]. Two fundamental building blocks of navigation that will be explored in this research are localization and dead reckoning. Although these techniques are old, new methods and algorithms are constantly being researched to achieve the best results. Within these types of environments the primary mode of transportation is walking. The localization and dead reckoning solutions are discussed from a pedestrian navigation perspective.

Localization methods are extremely varied from star tracking to using signals for triangulation. One approach that is very relatable to how a human localizes themselves is landmark detection within a scene. Humans are taught to understand objects, locations, and a multitude of other tasks through sight from a young age. While humans understand images with relative ease, computers algorithms have not developed

enough capabilities to achieve complex tasks in the past. Image recognition software has been developing and improving for decades with significant progress made from Artificial Neural Networks (ANNs) in recent years. With new advancements to ANNs, computational capabilities, and readily available imaging technology on the average person image based localization can prove reliable. These three items are important because ANN techniques have shown reliable results in many benchmark image processing problems. In order for the ANN to be able to train it needs a lot of data to create a generalized solution. With the increase in imaging devices around the world and social media platforms the amount of images for training have skyrocketed. Although even with enough data ANNs and the techniques used to employ them take huge computational demands. In the past computers would not have the computational capacity to handle all of the calculations needed to process large images and train the number of weights in large ANNs. With these three improvements image based ANN have become a prominent technique for image processing.

Dead reckoning navigation has long been utilized in systems such as maritime, aerial, and ground navigation. Many of the old uses of dead reckoning relied on simple techniques such as monitoring speed and time of travel. Within the last century inertial measurement units (IMUs) were developed to create a self contained system that can be utilized on almost any system providing acceleration and gyroscopic information. IMU sensors come in various degrees of performance capabilities from navigation to consumer grade. These sensors drastically differ in price and accuracy. ANNs are not only able to handle image based information, but are also highly capable in handling sequential information. In order for these models to operate they require data and in the two cases just discussed that comes in the form of images and IMU data. Cell phones are one of the most common pieces of technologies the that people own. Additionally, the majority of cell phones contain the required sensors to collect

images and IMU data that is needed for these pedestrian navigation solutions.

Although ANNs have proven successful there are still many solutions that exist. Classical filtering techniques have been in development and use in navigation for decades. These algorithms take in sensor data and are able to estimate a smoother signal from the sensor. In addition to just a single sensor these filters have the ability to take in multiple sources of information to provide the best estimation of a navigation solution. Even with the best sensors, noise and errors invariably creep into any solution. This noise has the potential to creep into any ANN or traditional methods. Additionally ANN do not create a 100% accurate solution. The networks themselves have error in their output. The use of classical filters in navigation has the potential to minimize these errors and create a robust navigation solution.

1.2 Problem Statement

Classical filtering methods have long been the standard for increasing reliability and smoothing out sensor information for navigation. Within the last decade research has been exploding utilizing new advancements in ANN technology. However, much of this research looks to solve problems with ANNs as the only solution. This work tries to explore the problem of pedestrian navigation using multiple ANN in combination with classical filtering techniques. By fusing classical filtering techniques with advancements in ANN, the solutions have the potential to create a more robust and better performing algorithm than either method on its own. Different techniques and neural network architectures are first explored to try to achieve the best results on the specific data set used for training. Metrics needed for the extended Kalman Filter (EKF) are obtained from the results from these neural networks and then additional tests are computed analyzing the fused results.

1.3 Assumptions

This work is not trying to create the best performance metrics for pedestrian navigation. It looks to utilize a new urban environment data collection with low precision positioning to test the viability of different techniques for localization and Pedestrian Dead Reckoning (PDR). With these models tested results are compared between the ANN solutions separately and those fused together with an EKF. Additionally, the techniques in this research assumes the data set collection remains in a flat two dimension plain. It's important to mention all neural network models took advantage of previously developed layers through TensorFlow[7] and Keras[8].

1.4 Thesis Outline

The remaining thesis covers four main sections of information. Chapter II contains relevant information to understand the techniques used to complete analysis on this subject. Chapter III discusses the process of data collection, data processing, ANN design, and EKF design. Chapter IV covers the results of the various experimentation. Finally a summary concludes this documentation outlining the results and providing additional work to be done on this topic in the future.

II. Background and Literature Review

2.1 Overview

This section provides background methods used in this research described in Chapter III. This chapter will cover key concepts in Global Positioning System (GPS), reference frames, vision navigation, Artificial Neural Networks (ANNs), state estimation, and filtering. In addition to information on these topics, descriptions on a few key alternative pedestrian navigation technologies are explored to include signals of opportunities (SoOP) and step counting methods.

2.2 Process Knowledge

2.2.1 Global Positioning System

GPS has become the preeminent form of localization and forms the basis of most navigation solutions. It is a satellite based radio positioning system that provides three dimensional positioning and velocity data as well as precise timing for a variety of tasks worldwide. The spacing of the satellites are arranged in six orbital planes to have general visibility by any receiver while maintaining on average six satellites within sight. Standard receivers in cellular phones utilize the Standard Positioning Service (SPS) signal which has a reduced capability when compared to the Precise Positioning Service (PPS). During normal operating conditions the SPS has a 95% accuracy to be within 12.8m and a 99.94% accuracy to be within 30m of the actual position at any point in the GPS coverage [9]. Whether using SPS or PPS the position data is calculated by trilateration of measurements from the satellites in view.

2.2.2 Reference Frames and Transformations

In order for GPS to provide accurate positions around the earth it utilizes a reference frame known as WGS-84. The reference frame plots the earth on an ellipsoidal coordinate system with the earths center of mass defined as (0,0,0). The standard includes constants for the earths ellipsoidal coordinate system, gravitational model, and magnetic model. The coordinates for this system are latitude(ψ), longitude(λ) and altitude above the reference ellipsoid(h) in meters and are showcased in Figure 1. This model has been updated and aligned extremely closely to the international terrestrial reference system as well. This alignment creates a common reference frame for analysis allowing different technologies to communicate with minimal conversions needed. Earth centered earth fixed (ECEF) is another common reference frame used in this research. The ECEF is very similar to the WGS-84 model except ECEF uses x,y,z instead of latitude,longitude, and altitude. The coordinate systems and their relationship are shown in Figure 1. In order to calculate the transition from WGS-84 to ECEF equations 1 - 6 are used where a is the equatorial earth radius, b is the polar radius, f is the flattening parameter, e is eccentricity of the earth, N is the distance from the surface to the Z-axis along the ellipsoid normal, and x,y,z are the ECEF coordinates[10].

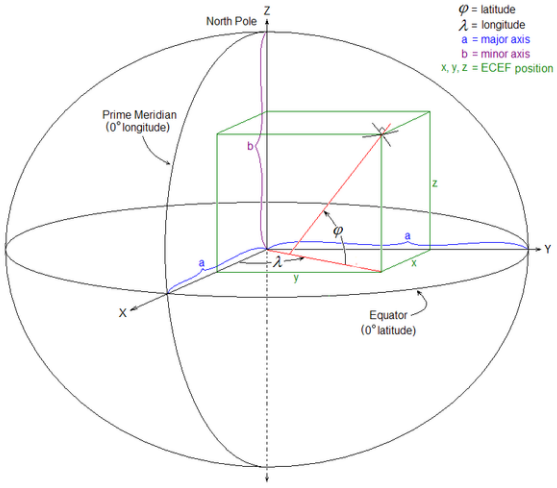

Figure 1: Ellipsoid ECEF coordinate system

$$f = \frac{a - b}{a} \tag{1}$$

$$e = sqrt(2f - f^2) \tag{2}$$

$$N = \frac{a}{\sqrt{1 - e^2 sin^2(\phi)}} \tag{3}$$

$$x = (N + h)cos(\psi)cos(\lambda) \tag{4}$$

$$y = (N + h)cos(\psi)sin(\lambda) \tag{5}$$

$$z = (\frac{b^2}{a^2}N + h)sin(\psi) \tag{6}$$

These common reference frames are used for a wide variety of calculations on the earth, but when working on navigation problems that only traverse a small area it can make mathematically and computationally more sense to convert to a north, east, down (NED) reference frame. This thesis works on a relatively small scale in the range of hundreds of meters making the conversion from a global ECEF to a local NED reference frame beneficial. The NED frame can be seen in Figure 2 where it utilizes a Cartesian coordinate system as well. The origin is placed on the surface of the earth

at any specified point. The x axis points to the north pole, y axis runs parallel to the longitude line, and z axis points towards the center of the earth. Within the NED reference frame the data is collected in its own reference frame known as the body frame. The body reference frame sets the cell phone as the center of another Cartesian coordinate system with the cell phone displayed horizontally. In this body frame the y direction points outwards from the front facing camera and x points perpendicular to y, and z points down towards center of the earth. This body frame moves within the NED reference frame.

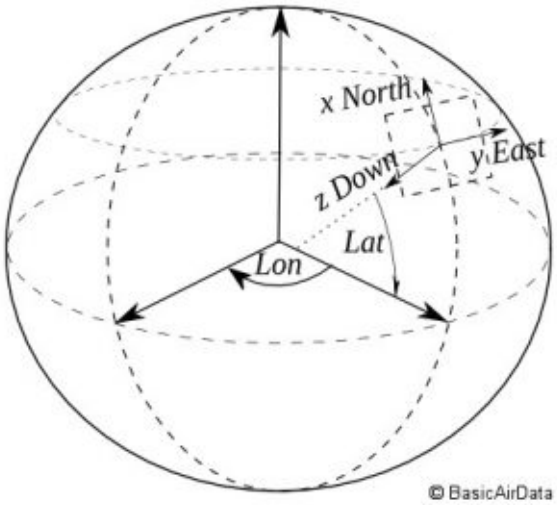

Figure 2: NED reference frame

In order to correlate these three reference frames a series of matrix multiplications by direct cosine matrices (DCM) are used. DCMs create a rotation from one coordinate frame to another. In certain coordinate transformations a translation is also needed. The calculations from WGS-84 have already been shown so the ECEF to NED DCM is showcased in equation 7. One important aspect to note is that the DCM utilized the NED origin reference point in terms of latitude, longitude, and height above ellipsoid. Latitude and longitude is a more common initialization format so the DCM from ECEF to NED uses ψ and λ as a base for conversion instead

of ECEF$_{x,y,z}$. The translation from the ECEF origin to the NED origin just uses equation 1-6 to obtain x,y, and z for that specific spot on the map.

$$\begin{bmatrix} -sin(\psi)cos(\lambda) & -sin(\psi)sin(\lambda) & cos(\psi) \\ -sin(\lambda) & cos(\lambda) & 0 \\ -cos(\psi)cos(\lambda) & -cos(\psi)sin(\lambda) & -sin(\psi) \end{bmatrix} \tag{7}$$

The conversion from the NED to the body frame requires both a translation from the NED origin to the center of the body and a rotation to align with the body frame. The rotation uses the DCM in equation 8 where angles are based off of roll(ϕ), pitch(θ), and yaw(ψ).

$$\begin{bmatrix} cos(\psi_s)cos(\theta_S) & cos(\theta_s)sin(\psi_s) & -sin(\theta_s) \\ cos(\psi_s)sin(\phi_s)sin(\theta_s) - cos(\phi_s)sin(\psi_s) & cos(\phi_s)cos(\psi_s) + sin(\phi_s)sin(\psi_s)sin(\theta_s) & cos(\theta_s)sin(\phi_s) \\ sin(\phi_s)sin(\psi_s) + cos(\phi_s)cos(\psi_s)sin(\theta_s) & cos(\phi_S)sin(\psi_s)sin(\theta_S) - cos(\psi_s)sin(\phi_s) & cos(\phi_s)cos(\theta_s) \end{bmatrix} \tag{8}$$

2.2.3 State Estimation

Within the study of control theory the concept of state-space representation utilizes mathematical models to represent various states of systems and processes (plants). States can be categorized into either discrete or continuous. The state-space utilizes inputs, outputs and state variables to attempt to control or observe a state. The state space models the current state based on the previous state and any measurement inputs into the system. A simplified example of this process could be trying to monitor and control the speed of an airplane. The airplane might be modeled with three states: plane velocity, thrust, and wind speed. Velocity would be calculated based on the thrust and wind speed. Sensors are used to provide a more accurate measurement of the desired state, but still introduce errors. Probabilistic mathematical models are an important tool used to represent many system dynamics

and real world phenomenon. These models help to predict the next states of a system or estimate how much error a sensor may have in its measurement. The mathematical model used to represent these systems is generally a linear time invariant system and measurement equation:

$$\boldsymbol{x_{k+1}} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k \tag{9}$$

$$\boldsymbol{z}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k + \boldsymbol{v}_k \tag{10}$$

Where $\boldsymbol{x}$ represents the state vector, the subscript k is the discrete time index of the system, $\boldsymbol{A}$ is the contribution of the current state to the next state, $\boldsymbol{u}_k$ is the input, $\boldsymbol{z}$ is the measurement, $\boldsymbol{C}$ shows the relationship between the current state and the measurement, D shows how the system inputs relate to the measurement, $\boldsymbol{w}$ is the error introduced to the current state, and $\boldsymbol{v}$ is the system error that relates to the measurement. The observed value in this system helps to maintain more accurate representation of the states and correct for the noise. One common method to model error in the systems is to use Gaussian Markov Processes shaping filter.

2.2.4 Gaussian Markov Process

There are a variety of different types of mathematical models for complex systems and errors. Many simple random processes that occur can be modelled as an additive Gaussian white noise (AWGN). This type of model allows for multiple different noise sources to combine together to be modelled as a single source. The model has a Gaussian distribution and uniform power across the frequency band. In some cases models are unknown and it's desired to generate an empirical autocorrelation and a mathematical model that matches it. For these cases collecting empirical data and utilizing linear shaping filters becomes a useful modelling tool. One set of models are called Gaussian Markov Processes. These models contain Gaussian distributions only

storing information of the last state. One model that is important in this research is an Exponentially time-correlated or First Order Gaussian Markov (FOGM) process. This model produces an autocorrelation function(Ψ) shown in equation 11 where σ^2 is the variance and τ is the time the data will de-correlate to 36.8% of its starting value. This model is useful in a variety of band-limited noises[11]. These linear models of system noise are extremely important in converting linear systems into more complicated systems.

$$\Psi_{xx}(\tau) = \sigma^2 e^{-|\tau|/T} \tag{11}$$

2.2.5 Filtering

Filtering methods are an important tool used in a variety of controls and navigation algorithms. An extremely common filter in navigation is the Kalman filter. The Kalman filter is a state space method of utilizing minimum mean square error to estimate new states from previous states and measurements [12]. The Kalman filter assumes a linear system as well as AWGN. The Kalman filter can be broken into three different sections. The dynamics model using state space representation can be written as [11]:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{G}w_k \tag{12}$$

$$\boldsymbol{z}_k = \boldsymbol{H}x_k + \boldsymbol{v}_k \tag{13}$$

Equation 12 and 13 are the same as the linear time invariant equation 9 and 10 in section 2.2.3. The naming convention used for these equations are different in navigation literature. Where $\boldsymbol{x}_k$, $\boldsymbol{B}$, $\boldsymbol{u}$, $\boldsymbol{w_k}$, $\boldsymbol{z}$, and $\boldsymbol{v_k}$ are all the same from the previous equations. The $\boldsymbol{A}$ matrix is re-written as $\boldsymbol{F}$, $\boldsymbol{C}$ is re-written as $\boldsymbol{H}$, $\boldsymbol{D}$ is not used and omitted [12]. The filter predicts the state by using the measurement and covariance

to propagate the dynamics equations forward using the following equations[11]:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1|k-1} \tag{14}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k|k-1} \boldsymbol{F}_k + \boldsymbol{Q}_k \tag{15}$$

$\hat{x}_{k|k-1}$ is the current estimate of the state given the previous state and $\hat{P}_{k|k-1}$ is the current covariance of the state given the previous covariance. The final output is determined by the Kalman filter update equations which are [11]:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1} \tag{16}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \hat{\boldsymbol{x}}_{k-1|k-1} [\boldsymbol{z}_k - \boldsymbol{H} \hat{\boldsymbol{x}}_{k|k-1}] \tag{17}$$

$$\hat{\boldsymbol{P}}_k = \hat{\boldsymbol{P}}_{k|k-1} - \boldsymbol{K}_k [\boldsymbol{H} \boldsymbol{P}_{k|k-1}] \tag{18}$$

These final equations update the estimated state and covariance matrix based off the Kalman gain $\boldsymbol{K}$. This set of equations help to propagate forward a system's state based off of measurements, signal covariances, and previous states. Although these mathematical models are extremely important in characterizing pedestrian navigation there are a variety of different types of measurement update techniques.

2.2.6 Vision Navigation

Vision navigation is a multifaceted topic with various techniques and algorithms used to accomplish the goal of navigating an environment using images. Some of the key aspects of image navigation include object detection, scene mapping, object motion, visual odometry, and localization. These techniques require the images to be processed in a way that creates geometric links from the imagery to the real world.

At the source of vision navigation is the image type that is used.

2.2.6.1 Images

In the basic sense an image can be represented as a matrix with each value representing an intensity of some value. A variety of imaging techniques are available depending on the goal such as x-rays, ultra sound, and radar. However, most standard images are created by utilizing three wavelengths from the visible light spectrum red, green, and blue (RGB). This format creates three separate intensity matrices on a scale from 0 to 255 of the three colors. When combined in various intensity values most of all the other colors a human can see are able to be captured in this information. One popular processing technique for images is to convert them into gray scale. Gray scale images converts the three RGB matrices into a single matrix. Analysis on images in this thesis is either done with RGB or gray scale images and will be denoted as such.

2.2.6.2 Image Processing

The most common image processing tool in relation to vision navigation is feature and descriptor detection. The features are utilized to identify items within an image without having to understand the full context of the image. This is important due to the fact computers only see the images as a set of matrices and don't inherently see contextual cues humans have evolved and trained to understand. Features can be as simple as an edge detection algorithm to more advanced algorithms like scale invariant feature transform (SIFT). SIFT is one of the most popular techniques of determining features and descriptors written by David Lowe[13]. This technique uses a method called difference of Gaussian blurs creating minima and maxima within the scene which are denoted as the key-points or features. These key points are

partially invariant to translation, rotation, and scaling. The descriptor takes into consideration key information surrounding each key-point. Eight bin histograms of the magnitude and orientation values are taken. These histograms are obtained from four by four sub-regions of a sixteen by sixteen matrix around the key-point. These magnitudes and orientation are sampled, rotated, and Gaussian weighting applied to the create a 128 bit vector of information. This descriptor vector creates additional information about the key-point that helps make key-points robust from artifacts such as illumination [13]. Calculating these features and descriptors from an image hundreds to thousands of points can be discovered to uniquely identify a keypoint from any other. From these features and descriptors a variety of things can be done such as key feature matching to match images and triangulation for video odometry. These techniques require a breadth of fundamental knowledge on the overall procedures as well as specific techniques for each algorithm. One way to to either increase the efficiency of techniques like this or completely circumvent them is to utilize machine learning.

2.2.7 Machine Learning

Machine learning is large field of mathematics, technology, computer science, and algorithms that try to develop different techniques to map mathematical function approximations to real world data or phenomenon. Machine learning utilizes applied statistics to estimate complicated functions with a decreased emphasis on proving confidence intervals[14]. Many of the statistical methods have been around since before computers, but are limited in what they can represent. Some of these methods include linear regression, logistic, and quadratic discriminant analysis to name a few [15]. These models are able to represent small dimensional problems that have fundamentally known and understood models. Advancements were made in the form

of decision trees, support vector machines, and nearest neighbors [15] to expand the scope of machine learning algorithms and their use cases. Many of these algorithms can handle regression problems or classification problems and have been able to solve a variety of problems and are still a primary solution for many problem sets. However, some of the major advancements in machine learning in the last decade comes from research on ANNs.

2.2.8 Artificial Neural Networks

ANNs are a form of deep machine learning. They are a method of stacking a large number of small mathematical nodes called perceptrons together in various ways. The goal of these networks is to approximate a function that maps a nonlinear transformation from some input to an output. Some of the key concepts of these ANNs include the perceptron, Activation functions, regularization, training, and the cost function. This list is nowhere near comprehensive of the items needed to fully understand neural networks, but gives a broad enough coverage for this thesis. Additional resources can be obtained by reading Goodfellow et-al[14], Chollet [1], or any number of neural network papers referenced.

Generally speaking neural networks most fundamental component is matrix multiplication and addition. Each neural network output(z) layer is the dot product of an input tensor($\boldsymbol{x}$) and weight tensor($\boldsymbol{w}$) and a bias(b) term as shown in equation 19. The tensor can be thought of as just a multidimensional array of any given size. For example a vector is just a one dimension (1D) tensor and a matrix is a two dimension (2D) tensor.

$$z = \boldsymbol{w} * \boldsymbol{x} + b \tag{19}$$

The output of this dot product is passed through an activation function(a) which is generally non-linear. This non-linear activation function allows the networks to learn

non-linear aspects of the data. There are a variety of different types of non-linear activation functions that could be used. Three will be focused on in this research and include the rectified linear unit (ReLU) [16], described and shown in Figure 3, the sigmoid, described and shown in Figure 4, and the hyperbolic tangent function (tanh), described and shown in Figure 5. The tanh and sigmoid function are relatively similar but have different slopes and bottom range values. The ReLU function has become the most used activation function in modern deep learning models, but is not a catch all and certain problems perform better with other activation functions [14].

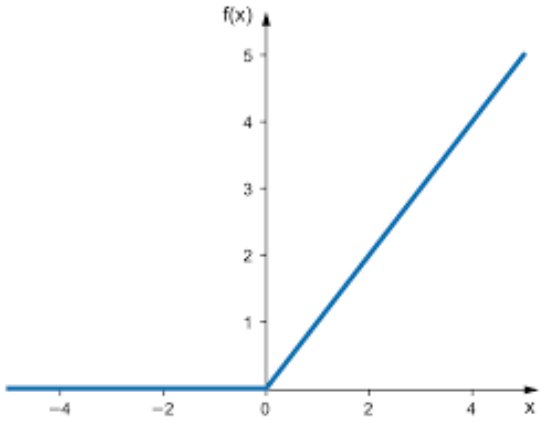

Figure 3: Graphical representation of a ReLU function. This functions mathematical notation is written as $a(z) = max(0, z)$

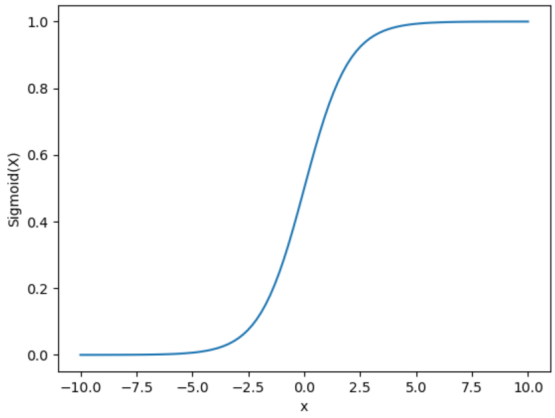

Figure 4: Graphical representation of a sigmoid function. This functions mathematical notation is written as $a(z) = \frac{1}{1+e^{-z}}$

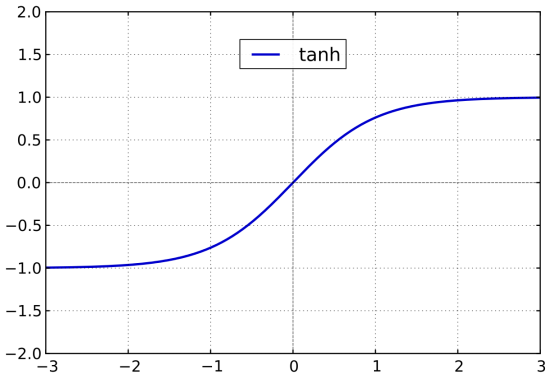

Figure 5: Graphical representation of hyperbolic tangent function. This functions mathematical notation is written as $a(z) = \frac{e^{2z}-1}{e^{2z}+1}$

These mathematical equations underlay the basics of the feedfoward portion of ANN. However, the primary advantage of using ANN is their ability to adjust the weights based off of some desired cost function. The cost function is utilized to create a real target value that measures the performance of the neural network. Similar to the activation function there is a wide range of different cost functions(J) available depending on the desired output of the neural network. Mean squared error (MSE) was chosen as the cost function for the majority of models due to them being regression

based. MSE is an L2 based cost function to measure performance where the error is calculated from equation 20. MSE takes the average difference between the sum of the squared difference between the predicted value and the true value.

$$MSE; J = \frac{1}{n}\Sigma_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{20}$$

Once the cost function is assessed the network determines a gradient of the cost function in relation to the weights and bias of each layer. This gradient is determined through a method called Backpropagation. Backpropagation is a fast algorithm that utilizes the partial derivatives of the cost function in relation to the weights and bias to calculate the gradient starting from the output layer back through the network to the first hidden layer. The partial derivative of the cost function with respect to the weights of a single layer can be shown in equation 21. The bias equation is the same as equation 21 except instead of the weights w there is a bias b.

$$\frac{dJ}{dW_2} = \frac{dJ}{da_2}\frac{da_2}{dz_2}\frac{dz_2}{dw_2} \tag{21}$$

When propagating the partial derivatives back through additional layers this first partial derivative can be reused as shown in equation 22. The ability to reuse the partial derivatives from previous layers and only recalculate the current layers partial derivative is how the back propagation method is able to calculate the gradient quickly.

$$\frac{dJ}{dw_1} = \frac{dJ}{da_2}\frac{da_2}{dz_2}\frac{dz_2}{da_1}\frac{da_1}{dz_1}\frac{dz_1}{dw_1} \tag{22}$$

These calculations will continue on until the gradient has been found for all hidden layers. The final gradient vector of all the weights and biases is shown in equation 23 [14].

$$-\nabla J = \begin{bmatrix} \frac{dJ}{dw_0} \\ \frac{dJ}{db_0} \\ \frac{dJ}{dw_1} \\ \frac{dJ}{db_1} \\ \vdots \\ \frac{dJ}{dw_n} \\ \frac{dJ}{db_n} \end{bmatrix} \tag{23}$$

With the gradient determined the weights can be updated and trained. The weights are trained based off of optimizer algorithms that utilize gradient decent based methods. Three different optimizers are used throughout this research including stochastic gradient decent (SGD) [17][18], root mean squared propagation (RMSProp)[19], and ADAM [20]. SGD takes the gradients of the layers determined from back-propagation and multiplies them by some learning rate (α). The weights are then updated by moving the weights in this desired direction. The function is trying to optimize each weight and find the lowest minimum value and steps closer to that minimum based on the gradient. Data is randomly sampled from the entire data set and updates the weights based off of the results of a given batch. In SGD that batch size is a single sample and the weights are updated after every sample. Implementations of SGD in this research had the added functionality of a momentum term to adjust the speed of change as well. This allows the signal to change faster when previous changes to the weights are high or slower if the changes are small. Optimizers are also driven based off of an adjustable learning rate. Depending on how large the learning rate is the algorithms may prevent the function from obtaining an optimal solution. This can happen by either having to low of a learning rate and getting stuck in a local minimum or to large of one and over correcting and never

obtaining the lowest minimum of the solution. RMSProp takes the basic model of SGD and tries to correct the learning rate as it progresses through the training. It accomplishes this by dividing the learning rate by an exponentially decaying average of the gradients. Adaptive moment estimation (ADAM) can be seen as an extension of the SGD optimizer as well. It utilizes both a momentum term and an adaptive learning rate to try to converge on the optimal solution faster. These topics make up the majority of information on how to actually train a neural network. These can be adjusted to improve optimization however the remaining ANN topics are not required to be used in a neural network, but can help improve optimization of a ANN.

Regularization is a process used to try to prevent overfitting [21]. Overfitting occurs when the machine learning algorithm learns the training set too well and has a hard time generalizing to cases outside of the data it already learned. Two common regularization techniques used in ANN design are batch normalization and dropout. Batch normalization normalizes the outputs of a previous layer before entering the inputs of another. Normalizing these weights speeds up the system, reduces reliance on low learning rate, and redistributes the outputs into a normalized range for the new input [22]. The normalization is based off of a running average of the data provided into the batch normalization layer. Dropout takes a different approach in that it randomly drops out weights and their connections during training. This is an attempt to reduce the neural network from creating strong similarities or co-adaptions during training[23]. Two types of dropout have been developed for recurrent layers. Originally dropout would only affect recurrent networks units at specific time steps. Current methods employ an approach that uses variational inference based dropout techniques [24]. This allows dropout to occur randomly at the input, output, and recurrent connections.

Another method for producing better results is taking into consideration the ini-

tialization of the weights. The network models require some initialization point of the weights. One of the most popular methods is called Glorot normal initialization [25]. This method draws samples based off of a truncated normal distribution about zero. The distributions standard deviation takes into consideration the number of input weights and output weights. Another popular weight initialization exists where weights are initially trained on a large data set and then transferred to the desired data set. This technique of pretraining weights is called transfer learning [26]. Transfer learning has the potential to reduce the size of the new data set needed in order to reach an optimal solution. Additionally, many of the pictures the networks are pretrained on may not exist in the new data set which will help create a more generalized solution. The items discussed underline the basic concepts of ANNs; however, there are two neural network layer structures that require additional information due to their importance to this research.

2.2.9 Convolutional Neural Networks

Two dimensional convolutional neural network (CNN)s are an important subbranch of ANNs designed to handle matrix like data. The layer implements convolution inspired function to find important statistical relationships within data being analyzed. The modified convolution operation operates by sliding a matrix kernel across the input and taking the dot product of the two matrices. A 2D representation of this dot product operation can be seen in Figure 6

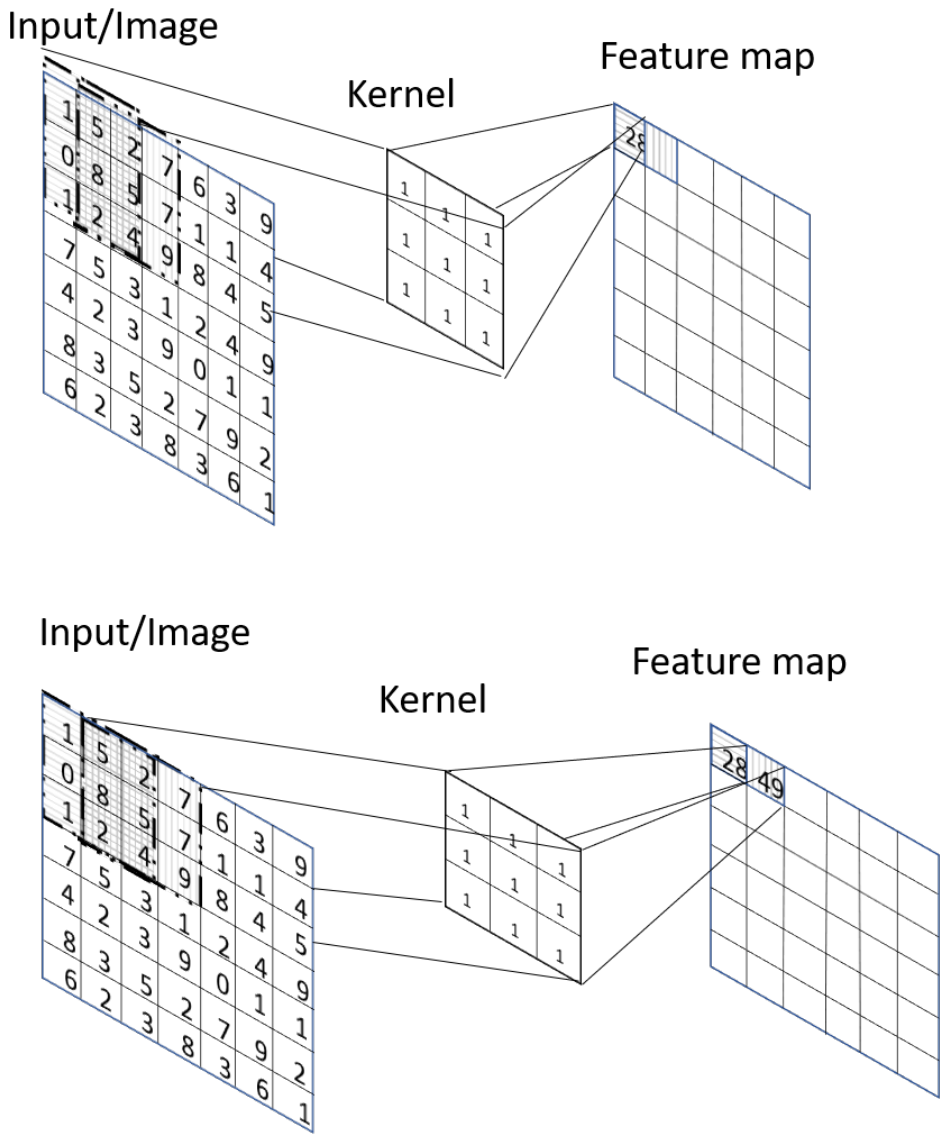

Figure 6: kernel based convolution method employed by convolution layers.

This convolution technique helps the model to learn features such as sparse interactions, parameter sharing, and equivariant representations. Sparse interactions help to define similarities within the data that only occur in a subset of pixels such as edge detection. Parameter sharing is the effect of keeping the weights for the kernel the same for an entire feature map to reduce the amount of weight parameters of the system. Finally equivariant representation allows the system output to have transfor-

mations appear that occured on the input data. These interactions drastically reduce parameters and increase statistical efficiency. Additionally, as the layers interact with further layers it links back to more and more of the initial nodes. This is one way convolution layers maintain a large degree of connections to the input image without requiring a direct connection. Most of the current progress in convolution layers is done in 2D layers. These CNN layers employ a process taking different kernels and sliding it across the input feature map creating a specified number of channels or feature maps. The kernels can learn to represent different types of attributes within an image to optimize the solution. These attributes can be as simple as an edge detector or as complex as determining a car. The kernel operations slide across the input images creating a feature response maps to determine if the image contains these filters within the image. Comparing the kernel filters to the entire image helps the neural network learn local patterns that are translation invariant. Additionally, a method called pooling is used to down sample the data helping the networks learn spatial hierarchies[1]. Maxpooling is the primary source of pooling and down samples the feature map based off a maximum intensity of a $n \times n$ window. Another down sampling method called stride can be used in conjunction with the pooling layers. Stride moves the maxpooling window a specified number of cells for each operation. If the stride is set to m it would reduce the feature map by $\frac{1}{m}$. Many of these attributes of a CNN are demonstrated pictorially in Figure 7 where an input image is being multiplied y a kernel to create a feature map and downsampled using Maxpooling layers until the final desired outputs are formed.

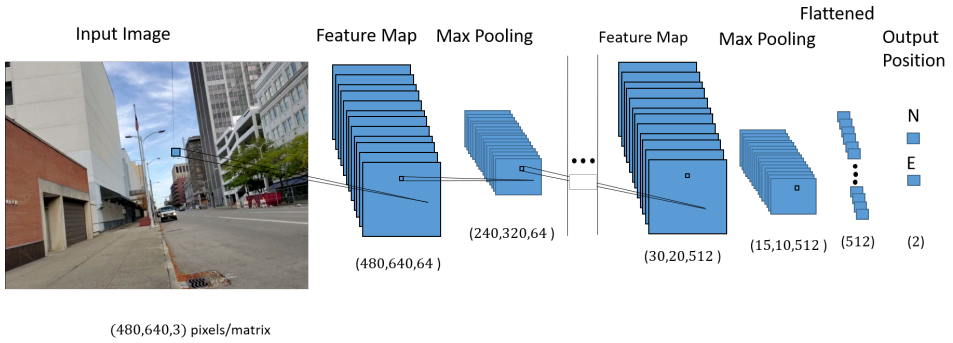

Figure 7: CNN pictorial representation. Showcases an Image and the kernel multiplication into the feature maps. Also indicates the reduction of the feature maps until the desired output with maxpooling layers

2.2.10 Recurrent Neural Networks

Where CNNs were made to understand spatially related data like images, RNNs were made to understand sequential data. Recurrent Neural Networks (RNNs) use a sequence of input data to produce output estimates either for a given sequence or for each time step in a sequence. The length of the sequence data input into the neural network is generally referred to as the lookback of the network. This takes the current time step and looks back a specified number of inputs and the entire duration is input into the network. Additionally, RNNs create a memory store to learn from past data points. This is typically accomplish by using some sort of output bus similar to Figure 8 that maintains memory of past outputs or information to be used at later steps. This helps to reduce the problem of older outputs becoming less important in training. The mathematical notation can be seen in the computational graph in Figure 9. This function takes an input x at each time step and is fed through some weight matrix h to obtain an output o. The weights are then updated by some loss function $\mathcal{L}$ corresponding to a training target y [14]. There are many different implementations of RNNs using the memory store but the two most popular architectures are Long

Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Both LSTM and GRU networks fall under the category of a gated unit which are help eliminate the problem of the gradient used to train the weights either vanishing as back-propagation works its way into early layers or explodes into extremely high numbers for specific nodes [14].

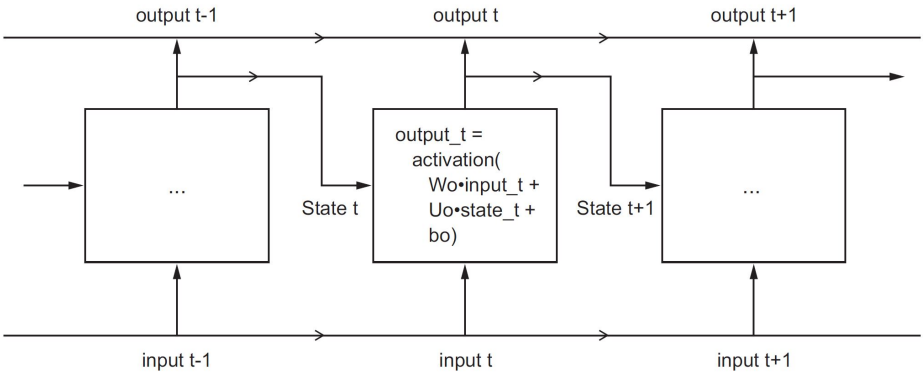

Figure 8: Simple RNN Model [1]. Three cells of a single RNN unit are showcased. The sequential inputs feed into each cell with an information bus moving forward through the system as well as potential outputs.

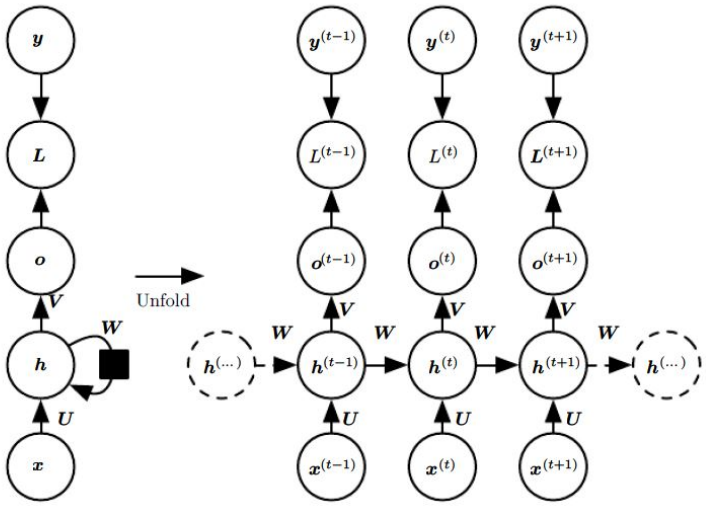

Figure 9: RNN computational graph showcasing the mathematical representation of the RNN algorithm flow

The LSTM and GRU are RNNs primarily suited for solving sequential type data. Additional information and clarification about LSTM layers can be found in Hochreiter et-al's paper[27] and GRU can be found in Merri's paper [28]. Both layer types are designed to eliminate the problem of back-propagation gradient techniques values gradually reducing to zero the earlier the layer occurs in the network by using gates. The GRU utilizes a reset and an update gate as shown in Figure 10. This reduces the number of operations needed for training and speeds up the model The LSTM basic block can be seen in figure 11. This model has three gates the input, output, and forget gate which are indicated by the sigmoid functions on the Figure. Each layer type stores information from previous state and uses that memory to create better predictions for the current state.

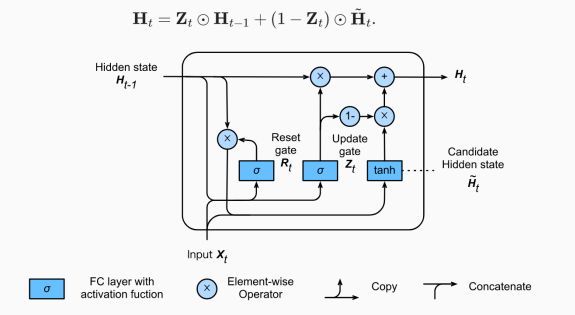

Figure 10: A single GRU cell within the GRU units. Showcases the mathematical notation as well as the internal mechanisms of each cell.

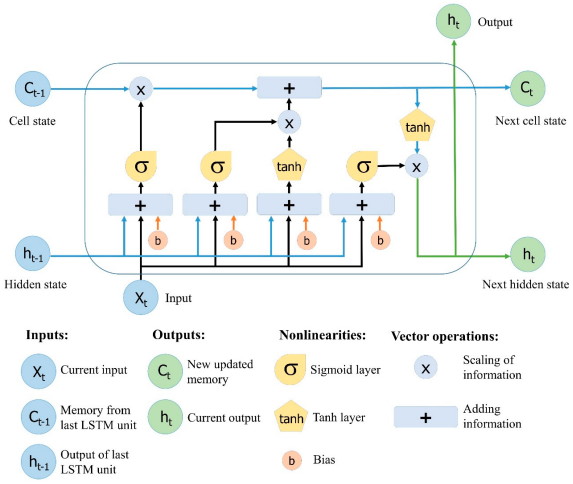

Figure 11: A single LSTM cell of a LSTM unit. Showcases the mathematical representation of the model as well as all of the gate functions.

2.2.10.1 Temporal Convolution Network

1D Convolution layers are a way to take advantage of the convolution aspects on sequential data. In general recurrent layers are the dominant solution to sequential problems; however, 1D convolution layers have been shown to solve sequential data as well[2]. 1D convolution layers operates by taking in the sequential data and having a kernel operate on a set size of the data taking neighboring inputs to try to determine a solution. There are a variety of different methods and architectures that have

shown success with 1D convolution layers. One specific architecture was developed that has shown improved success in a variety of tasks. This architecture incorporates 1D convolution layers, batch Normalization, dropout, and activation function. The convolution layers stack up on top of each other with a dilation of the kernel between each layer increasing by a factor of 2^n. This set up constitutes a Temporal Convolutional Network (TCN) block and is shown in Figure 12. These TCN blocks can also stack on top of each other to create larger and larger sequences. This block stacking allows the output to access further information into the past and create a something similar to memory in the system [2].

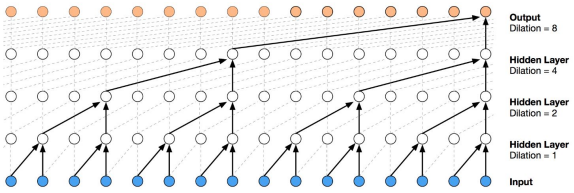

Figure 12: TCN block as outlined in [2]. Showcases important concepts such as dilation and the layering of 1D convolution layers and their contribution to the overall TCN block

2.2.11 Key Convolutional Neural Network Architectures

There are a number of models that have been developed and published to tackle image classification problems. These models were designed to try to solve different problems within neural networks at the time of their development. Most of the neural network architectures compare results based on the ImageNet data set. This data set consists of over fourteen million images to categorize one thousand different classifications[29]. As these top models continue to improve and showcase high performance results it's advantageous to explore these models on additional data sets. When using preexisting architectures the initialization of the parameters is impor-

tant. Using transfer learning is common and many of the networks are available pre-trained on the ImageNet data set. It's important to give a brief overview of the various models that were either replicated and adjusted or used as it was originally designed in order to understand why one model might perform better than another.

2.2.11.1 VGG16

The VGG model explored the differences in a variety of neural network architectures, but most notably the depth of layers. Authors Simonyan and Zisserman tested six different layer depths ranging from 11-19 layers as showcased in Figure 13. Moreover, they took advantage of previous work testing filte kernel size and optimized to three by three matrix filter kernels. The model used in this research is the VGG16-D in Figure 13[30].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 13: The original six architectures tested under the VGG name. Model VGG16-D is used in this paper for analysis

2.2.11.2 Residual network (ResNet)

The ResNet technique stacks residual blocks together making an extremely deep network as noted in He's paper[31]. These residual blocks copy the input data, pass one copy through a series of convolution layers, and then adds the result of the output of the convolution layers and the original copied input data together. In order to make this work the input layer has to have the same dimensions as the output of the convolution layers. These blocks are then stacked in series creating the deep

networks. Creating a path that allows the input information to flow down allows the information to continue to create useful connections without increasing the number of weights of a model. Moreover, it gives paths back up the network keeping the gradient from vanishing when it gets to earlier layers. The authors hypothesized that if a neural network mapping could fit a function with a few stacked layers then it could also learn a residual function. A graphical representation of the mapping from input to output of a block is showcased in Figure 14. Multiple different variations have been explored of the ResNet model with one of the highest performing being the ResNet50[31].

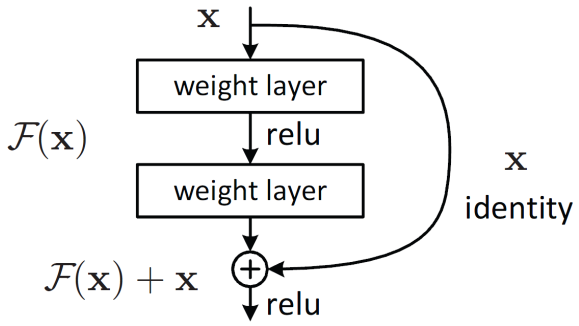

Figure 14: ResNet block showing the flow of data, activation function, and mathematical operations done.

2.2.11.3 Wide residual network models

The wide residual network models were inspired by problems with deep residual networks. The authors Zagoruyko and Komodakis[32] indicate the lack of necessity forcing the gradient to flow through the residual blocks. They believed it could create a scenario where certain blocks held all of the information and many others didn't have any useful information [32]. Wide residual networks have the problem of dramatically increasing weights, but these weights can be trained in a parallel fashion to greatly increase speed. The papers main objective was to test varying degrees of depths and

widths of the neural network. Their base model consisted of alternating a two layer convolution residual block and maxpooling layers six times. From the base model the authors explored deepening the model by adding convolutional layers in each of the residual blocks. Additionally, they tested increasing the feature maps created from each convolution layer and different combinations of the two methods. Their work found that by keeping a smaller depth, but increasing the width it had greater performance when testing on ImageNet compared to the ResNet inspired models.

2.2.11.4 Inception and Xception

The Xception model is an architecture designed by Francois Chollet[33] which extends the work and hypothesis of the Inception model [34]. The main hypothesis for the Inception model is to separate the cross-channel correlations and spatial correlations. This is due to the models being tasked with learning 2D image spatial data and a channel dimension. The Inception model tries to separate these into independent learning processes. The cross-channel correlation is handled by a 1x1 convolution layer and then mapping these into 3x3 and 5x5 convolution kernels. This can be illustrated in Figure 15. The Xception model makes the assumption that the cross-channel and spatial correlations can be learned completely separate from each other. The Xception model is made up of 36 separable convolution layers with ReLU activation layers and batch normalization. The model can be seen in Figure 16 taken from the Chollet paper [33].

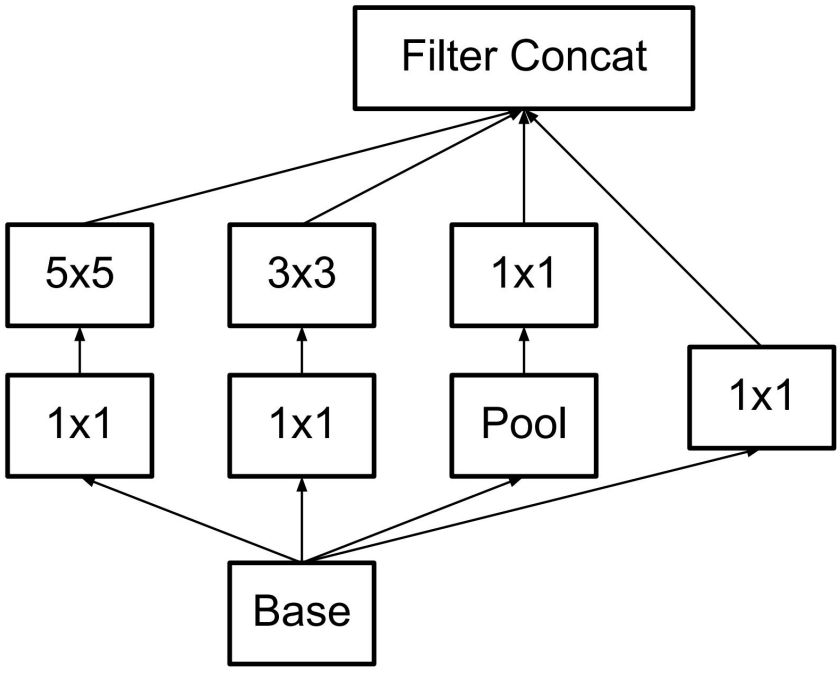

Figure 15: base inception model

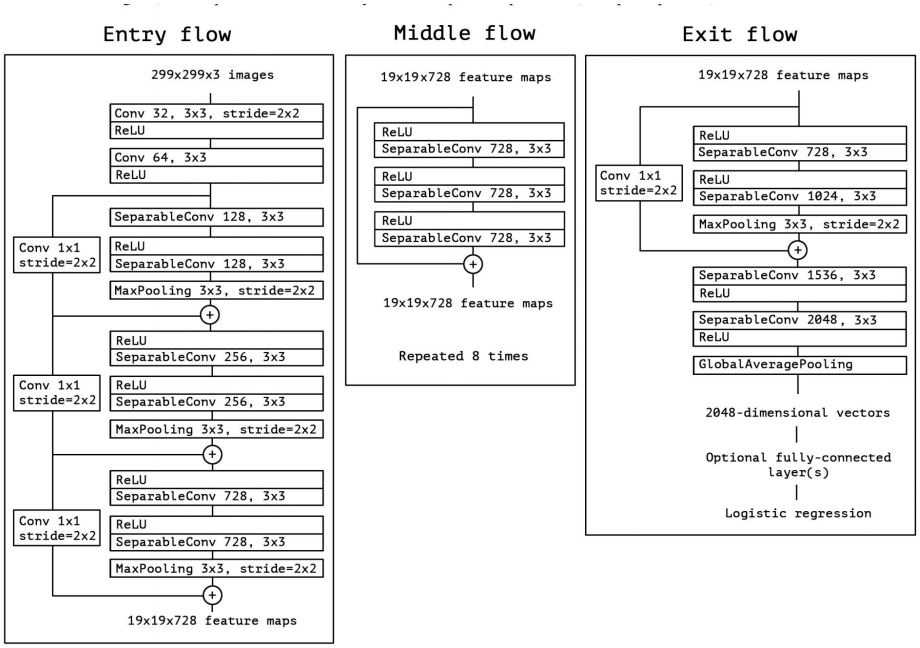

Figure 16: Xception model and image flow through. Note: after each convolution layer a batch normalization layer exists, but is not shown

2.3 Related Work

With some background information for understanding the fundamentals of the work in this thesis out of the way, discussing potential alternative solutions is important. Pedestrian navigation is a well studied field with different levels of investment and maturation. Two different techniques will be examined including a localization and Pedestrian Dead Reckoning (PDR) based method. The localization methods explored fall under a category of solutions considered SoOP. In addition to SoOP an image based navigation method is discussed. One of the most common PDR algorithms utilizes inertial measurement unit (IMU) sensors for step counting and heading algorithms. Finally two ANN solutions for localization and a PDR solution will be explored to show alternative techniques. There are a quite few techniques for localization as mentioned earlier. One group of techniques is considered SoOP which include methods such as Bluetooth Low Energy (BLE) [3], Wireless Fidelity (WIFI) [4], and other SoOP solutions [5]. SoOP has achieved various degrees of accuracy and can be explored in the papers listed, but another method is more pertinent to this research. Image processing techniques have long been explored for use in localization. Image processing techniques utilize features and descriptors of an image to create landmarks that are invariant to position in the scene. One of the most common types of image feature detection is called SIFT as described in section 2.2.6.2, but additional exist such as Oriented FAST and rotated BRIEF (ORB) [35] and speeded up robust features (SURF) [36]. One problem that arises from utilizing these methods is processing speed and their inability to fully handle rotation. Additionally, the detection methods are not fully learning the features and are commonly mis-identified and need reduction algorithms such as random sample consensus (RANSAC) [37]. These feature based methods provide the benefit of not needing extremely large training sets like ANNs. In addition to localization methods a step counting method is explored

for PDR.

Step counting algorithms are fairly straight forward and easily implemented. This technique generally utilizes at a minimum an accelerometer, but usually employs a nine degree of freedom magnetic and inertial measuring unit (MIMU). This allows a three degree of freedom accelerometer to determine when a step was taken based on spikes in the sensor readings regardless of sensor orientation. The downside of this technique is that it only provides the time duration between steps and the time a step occurs. Human bio-metrics are required to calculate the step length. These are generally averaged for individuals based on the impact magnitude and time between impacts [38]. Secondly a three degree of freedom gyroscope and magnetometer are used in order to track changes in orientation and heading. The orientation algorithms generally have a high degree of drift and are not reliable for long periods of time. The magnetometer data is prone to be noisy in areas with ferrous material. Finally, barometric pressure sensors are generally utilized to increase accuracy of elevation and can generally find results within one meter [38]. Figure 17 showcases the exact point where an algorithm would classify the data as taking a step based off of its received sensor values. Finally, all of these individual techniques are combined in order to navigate using step counting. Step counting can be utilized for PDR, but doesn't accomplish localization. The errors in this type of solution compound on one another and predicted positions drift over time.

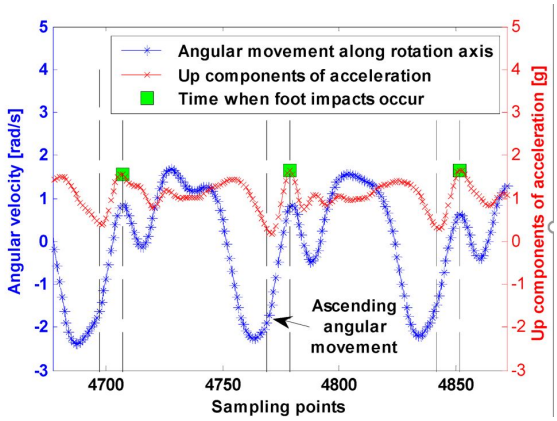

Figure 17: IMU data showcasing point of impact of a step and angular rotation calculations

The first ANN localization solution discussed is for localization of a camera within a scene. The method has been named PoseNet [39] and its designed to take images of an urban scene and determine its 3D position and quaternion represented orientation. The labels are created using a technique called structure from motion which uses features in the images to create a 3D representation of the environment. The image labels are then determined by matching the features to the scene. The authors used transfer learning techniques with the GoogLeNet [40] architecture to solve the problem. The classification outputs were adjusted to perform regression based loss function based off the position and quaternion absolute error. This method was able to achieve results within two meters and five degrees of the truth labels.

The second ANN localization solution tested in Dr. Curro's disertation [41] utilized information collected in two different unique data sets including indoor and outdoor environments. Multiple different ANN techniques where employed to determine a navigation solution using these frequencies. One method particularly important to this research is taking continuous trajectory points within the scene and clustering them into discreet points. This puts all of the available positions into a few categories. Having only a few categories allows the networks to employ classification

based learning instead of regression. CNNs are generally trained to perform classification and using this in conjunction with high performing classification models may improve localization.

The ANN solution for the PDR as been named the Oxford data set. It's a PDR data collection of 158 sequences covering 42.587 km of IMU and magnetometer data. The data was collected from five different users on four different cell phones inside a small room using Optical Motion Capturing System (Vicon) for providing high precision training labels. The training was tested with the phone in four different positions including in the hand, pants pocket, inside a briefcase, and inside a cart. This provided various methods for testing different algorithms. The authors tried various recurrent neural networks combinations to solve the PDR problem, but determined a two layer LSTM network achieved the greatest results.

III. Methodology

The research presented looks at testing the improvements achieved to Artificial Neural Network (ANN) solutions using classical filtering techniques on a pedestrian navigation problem. In order to use the filtering methods two different ANN techniques are explored to solve pedestrian navigation. These ANN algorithms are trained on a new urban data set. Cell phones were chosen as the tool to collect this data as they have a robust technology suite and are widely available and used within society. Moreover, using this type of equipment over other sources creates a solution that has a minimal impact on preexisting structure and reaches a large percentage of the population. After data collection the data processing is discussed. Methods for determining a valid pedestrian navigation solution are explored using ANN solutions based on localization with imagery, Pedestrian Dead Reckoning (PDR) with accelerometer, gyroscope, and magnetometer data. Finally, with both ANN solutions discussed the method of combining both of these solutions with an extended Kalman Filter (EKF) into a single solution is discussed. Workflow of the process follows figure 18.

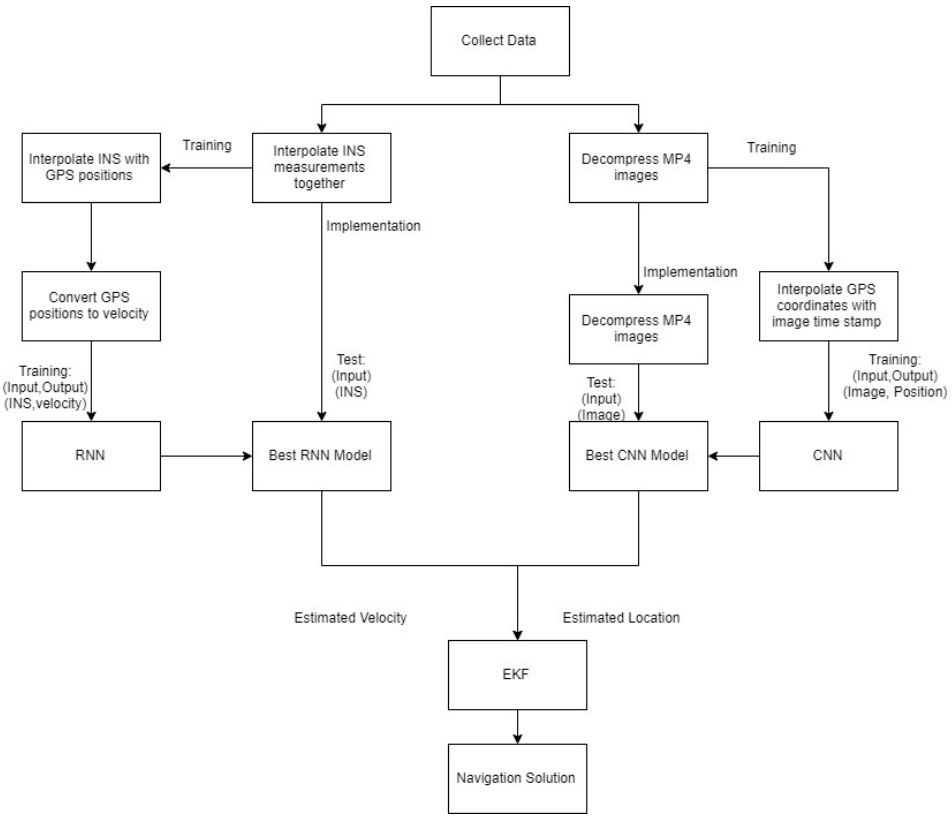

Figure 18: Algorithm Workflow from data collection through ANN testing and deployment

3.1 Data Collection

As mentioned a cell phone was used to collect the data needed for analysis. Any cell phone with the sensors mentioned should be capable of collecting the required data, but the data in this research was obtained with a Samsung Galaxy S10. The sensors in this phone are higher quality sensors within the commercial cell phone sensor market. During data collection the inertial measurement unit (IMU) Micro-Electro-Mechanical Systems (MEMS) components accelerometer, gyroscope, and magnetometer, as well as GPS location, barometric readings, and mp4 videos were all collected and stored for later analysis. All data was collected with the cell phone attached to the same person using a chest harness and the front camera/screen facing outwards

shown in figure 19. The data was collected in a two block radius within an urban downtown environment over a three month period of time.

Figure 19: location of Chest harness worn during data collection.

3.2 Data Processing

All of the data was post processed and analyzed to create training data formatted properly for the algorithm they are used in. Each session of collecting data was processed into .hdf5 files because of the large storage capabilities and fast slicing of the data. The mp4 videos are deconstructed into image frames taken at 30 hz and 640x480 pixel resolution. The images were stored in both a gray-scale and red, green, and blue (RGB) format for testing different ANNs. No data augmentation was performed on the images. However, due to the motion of the camera while walking images contained rotation up to 30° off camera vertical axis, 45° rotation off camera horizontal axis, and

image blurring through out the data set. This introduced rotation and smearing into the training sets, mimicking what would be seen in a real world test. Each of the other sensors have different update frequencies creating timing mismatches for each event of data. The IMU, magnetometer, and Global Positioning System (GPS) sensor data was interpolated at one hundredth of a second to provide consistent timestamps for each input and output pair. For the images generated the GPS position information and timing are interpolated to match image timestamps, geo-tagging the image data. Interpolation starts at the sensor with the latest first timestamp and finishes at the sensor with the earliest final time step. After interpolation it was important to note the data was then scaled down to get the majority of values between -1 and 1. The IMU and magnetometer data had the z-score taken which subtracts the mean and divides by the standard deviation to create a Gaussian distribution centered about the origin. The north and east positions and images were scaled down to make all data between zero and one. The positions were scaled down by a factor of 400m and the images were scaled down by a factor of 255.

The GPS coordinates are obtained in WGS-84 format using latitude, longitude, and height from the ellipsoid and used to geo-tag both the PDR and localization solutions as the supervised training results. Latitude and longitude format doesn't have a large degree of variation of values in a small reference frame so the GPS coordinates are converted into a local North, East, Down reference frame. This is important because neural networks train best when data sets have the greatest spread between a negative one to one range. Topology of the collected data stays relatively flat so for ease of computation the frame was flattened into just two dimensions North and East. The North and East coordinates are then normalized between a zero and one. During testing the GPS absolute error can't be determined for any data point because there was no known reference points for the collections. The relative error

based on average distance travelled between each update was used to assess any position error. It was assumed the error would take a Gaussian distribution and any error greater than five σ of the mean was considered to be too large of a difference between time steps and removed. Remaining data points were smoothed by a local average, shown in equation 24. The average was taken over eleven time steps including the current position, five before, and five after.

$$x = \frac{1}{n}\Sigma_{n=-5}^{n=5} x_n \tag{24}$$

In order to obtain the training targets for the PDR ANN the GPS positions need to be converted into different metrics due too the sensor information. IMU sensor calculates relative speed of change and rotations of the camera for a specific time period. The magnetometer sensor tries to calculate a heading based solely on the magnetic north pole of the earth. Neither of these sensors have the ability to sense anything at a given position that will anchor it to a real world position. This also eliminates the ability to use absolute velocity measurements in training as there isn't an easy or accurate way to calculate a true north or east velocity from the IMU and magnetometer data. To account for this, the target output for the PDR models use a relative position change calculating the distance(d) between two points and a delta angle (ϕ) can be calculated for each time step. This creates a relative distance traveled in a given time frame and can be found using equation 25. The change in orientation from each time step can be calculated based off these two positions as well. The angle θ_k in equation 26 and shown in figure 20 is first calculated as a reference point to each position. Then ϕ_k is calculated for each time step using equation 27. The desired angle ϕ and distance(d) are pictured in figure 20.

$$d_k = \sqrt{\Delta north_k^2 + \Delta east_k^2} \tag{25}$$

$$\theta_k = \text{atan2}(\Delta north_k, \Delta east_k) \tag{26}$$

$$\phi_k = \theta_k - \theta_{k-1}; \phi_0 = \theta_0 \tag{27}$$

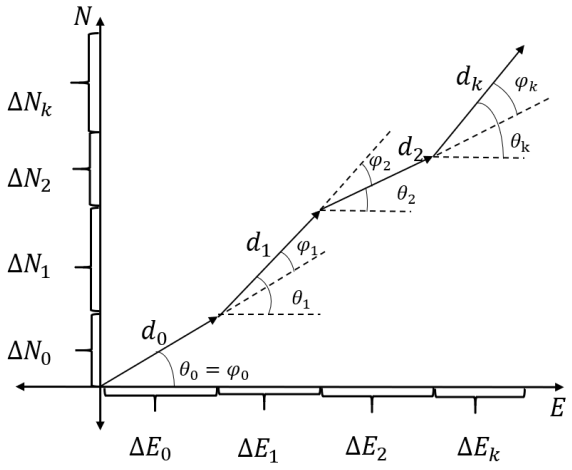

Figure 20: Position displacement used for PDR. d and ϕ are used as target outputs and θ is used to calculate ϕ at each time step

The distance measurements for the PDR target output weren't scaled as the inputs already mentioned. Angle target outputs were modulated between -π and π as the vast majority of the data is between negative one and one. The distance and angle change were tested with changes at every time step as well as with a windowing effect on the data. The windowing takes two points that are a specific time step apart and calculates the total distance and angle change from these two positions. An example of the windowing effect on the distance and angle change data can be seen in figure 21.

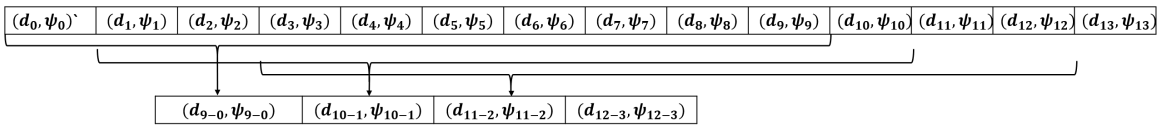

Figure 21: The windowing effect of the output data. This takes data points a specific time apart and calculates the total distance and angle change between the time steps before sliding to the next iteration.

3.3 Neural Networks

Two different work paths exist for the neural network to include imagery based localization and IMU and magnetometer based PDR. A variety of different ANN architectures based on convolution layers are tested to achieve the best localization results. Similarly, ANN structures based primarily on recurrent type layers are explored to solve the PDR navigation problem.

3.3.1 Neural Networks for localization

The localization models are all based on popular techniques and were each trained and tested to determine the smallest localization error. Three different model architecture types were trained with a Glorot initialization[25]. These networks were being optimized to the urban data set only and did not employ any transfer learning techniques. These base models were differentiated into basic sequential, residual network, and widenet overarching model types. In addition to these the Xception and VGG16 models were trained and tested on the current data set with weights initialized using transfer learning from the ImageNet data set. All localization neural networks tested utilize a supervised learning method of training based on images geotagged in a localized north, east, down (NED) reference frame. All of the models tested utilized a variety of different hyper-parameters for batch Normalization, drop out,

activation function, and optimization function to optimize the system to the lowest mean squared error (MSE) and are explored in chapter IV.

3.3.1.1 Image Based Glorot Initialized Models

Three overarching model architectures were initialized using the Glorot method that took advantage of different popular techniques in convolutional neural network (CNN) architecture. All of the architectures these models are based off of provide some different approach in trying to solve the image recognition problem. Additionally, each model performed well on bench mark data sets showcasing their ability to work. During testing a base model for each architecture type was chosen to provide the smallest size to minimize computational requirements of the system. Testing then expanded from there increasing various attributes and testing for various parameters of the neural network. These base models consist of a convolution based sequential feed-forward, residual network, and a wide-network system.

The sequential model refers to the sequential layout of the convolution layers in a feed-forward based system similar to the VGG16 model[30]. The base model of the basic sequential type consists of six two dimension (2D) convolution layers alternating with six maxpooling layers and finishing off with two dense layers. Different variations of this model were trained with various hyperparameters. Most of the testing of this models was in the feature map size and the amount of convolution layers between maxpooling layers. Adjusting the size of the network creates more parameters and greater feature maps to potentially learn additional features in the images.

The residual network inspired architectures are all based off of the fundamental residual block showcased in the residual network (ResNet)50 model. The models generally consist of residual blocks stacked sequentially creating a deep network. Most of the models inspired in this category were tested with variations of the block archi-

tectures. The blocks were set up in varying sizes and arrays consisting of convolution layers ranging from two to eight layers. Increasing the layers within the block increases the amount of parameters that can adjust and train to those specific data points and reduces the effects of the residual connection. This increases the chance of the gradient vanishing and the network not being able to adjust the weights properly. Architectures were also adjusted by adding various number of residual blocks before completing a maxpooling layer. This creates larger number of weights at a given spatial size, but keeps residual connections interval small. This allows the gradient to be generated easier, but introduces more information from each previous block.

3.3.1.2 Image Based Transfer Learning Models

The models in this section utilized transfer learning technique to pre train all of the network weights. Tests were completed using the Xception and VGG16 model with various training methods. The ImageNet classification data set was used for pretraining finding optimal weights to solve 1000 seperate classifications. The model weights are downloaded from the keras library using the training methods described in the models papers. The model is then converted from a classification network into a regression network by replacing the bottom layers. In the Xception model from figure 16 section 2.2.11.4 the layers removed are all within the exit flow block. Everything after the globalpooling layer is removed and replaced with two dense layers. For the VGG network shown in figure 13 section 2.2.11.1 the last three fully connected layers and the soft-max output are removed and a globalaveragepooling layer and two fully connected dense layers are added. This eliminates the previous output layers that are heavily trained on the previous data sets. The new layers will only be trained on the current urban data set and will have a more focused output since it's only trained on the urban environment. The activation function for the additional layers

were a rectified linear unit (ReLU) on the first dense layer and linear on the final output layer. These added layers were trained on the current data set with all of the pre-trained network weights for 20 epochs. Models were then retrained with the urban data set by unfreezing layers in the model. Weights were unfrozen at multiple different layers within each model, but were always done after a maxpooling layer to keep spatial information the same. Mean square error was used as the loss metric to train and track the error as the modeled trained. Additional information on the models can be found from either Chollet [33] or Szegedy [34].

3.3.1.3 Classification Based Navigation

One final approach to improve localization results was used which includes a technique of clustering the trajectory points along the paths into discreet positions. This was accomplished by using a kmeans clustering technique into 100 unique clusters. The clusters centers were determined from the combination of all of the training data. Once the center points were determined the kmeans algorithm placed each point to its closest cluster. The cluster center point label was then used as a classification problem with 100 unique labels. These labels were one hot encoded placing a one in the correct cluster label and zeros every where else. The cluster points were weighted to remove any uneven training. The weights were determined by the inverse of the percent each label class obtained shown in equation 28. The ANNs tested using this classification method were the VGG16, Xception, and WideNet model. The last layer outputs were replaced with a softmax layer. Moreover, the loss function used a categorical crossentropy and the single highest probability value was used to track accuracy.

$$\frac{1}{\frac{labels\ in\ category}{total\ \#\ of\ labels}} * 100 \tag{28}$$

3.3.2 Pedestrian Dead reckoning

The PDR function was approximated with an ANN as well. Networks are originally designed and tested on the Oxford data set [42] and then tested on urban data set. The majority of the networks chosen to try and solve this approximation fall under the recurrent neural network architectures. Many of the architectures and methodology used for this analysis are derived from research done by Chens-et-al [43]. Three different Recurrent Neural Network (RNN) layer types were utilized for training including Long Short-Term Memory (LSTM),Gated Recurrent Unit (GRU), and Temporal Convolutional Network (TCN). The LSTM and GRU layers were also trained using a bidirectional approach analyzing the inputs from both directions. Training of the networks was done with a supervised learning method feeding it the expected input and output data. Two separate input data sets were tested for training. The first utilized three axis gyroscope and accelerometer data and the second used 3 axis gyroscope, accelerometer, and magnetometer data. The accelerometer and gyroscope data were used in both scenarios as they provide cell phone orientation information. Magnetometer data can be extremely noisy in an urban environment due to ferrous materials in the building, vehicles, infrastructure, etc. However, the neural networks are thought to either train around these anomalies or use them as additional information to the velocities as they are passed. Additionally, magnetometers are widely used in cell phone compass headings and the neural network may have been able to learn heading information to improve the solution.

Output Type	TimeStep	T_9									T_{10}									
Output for every input	Input	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
	Output	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}
Single final step output	Input	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
	Output	O_9									O_{10}									
Summed	Input	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
	Output	$Sum(O_9 - O_1)$									$Sum(O_{10} - O_2)$									

Figure 22: Three variations of the PDR outputs. All Three models are given the same inputs; however, the first one is the single output of the final step. The second model only output a step for every input received and the final output was a single sum of the entire time duration of input data.

The output target data for the PDR was analyzed three different ways as showcased in figure 22. Each method has the same input of time sequenced data of a specified length. The first method outputs a distance and angle displacement for each input. The second method only outputs the distance and angle change of the last step. The final method outputs the total distance and angle change for the time duration of the entire input. The RNN models, like the CNN models above, had various parameter sweeps to dial in the best model with the lowest error. The parameters that were tested were the number of layers, look back of sensor data, learning rate, activation function, optimization function, batch size, number of epochs, and number of hidden nodes. In addition to PDR and localization ANN solutions an EKF was designed to combine both solutions into a single one.

3.3.3 Extended Kalman Filtering

The localization and PDR ANN solutions produce an estimated solution for localization and PDR based on real world sensors. These solution invariably have error

from not only the sensors but the ANN algorithms as well. In order to compensate and filter these errors an EKF is used. The EKF is similar to the Kalman filter already explained in section 2.2.5, but the EKF can better handle nonlinear systems with slight tweaks to the algorithm. The EKF state space model consisted of four states including north position, east position, north velocity, and east velocity. The state update equation showcasing the dynamics of the system and dynamics noise is written in equation 29. No input is propagated into the state equation so matrix **B** from equation **??** is left out. The dynamics matrix **F** of the system only updates the position based off of the velocity.

$$\hat{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{north} \\ P_{east} \\ V_{north} \\ V_{east} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 e^{\frac{-|t|}{\tau}} & 0 \\ 0 & 0 & 0 & \sigma^2 e^{\frac{-|t|}{\tau}} \end{bmatrix} \tag{29}$$

Although there is no velocity component in the dynamics matrix there is velocity dynamic noise which is modeled as a first order Gaussian Markov (FOGM). This drives the velocity to zero if no updates are provided. This system relies heavily on measurement updates to propagate forward and the trajectory motion would stall after a short period of time without measurements. The localization outputs from the CNN were used as measurement updates for the north and east EKF positions. PDR results of position displacement and change were given an estimated current orientation based on the current and previous estimated states of the EKF system. This PDR change in angle was added to the orientation estimate and then the north and east velocity components is determined for the time step and used as a measurement update for the EKF. The measurement updates happened at two separate frequencies and therefore each were given a separate two $\times$ four H matrix as shown

in equation 30 and 31.

$$H_{localization} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{30}$$

$$H_{velocity} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31}$$

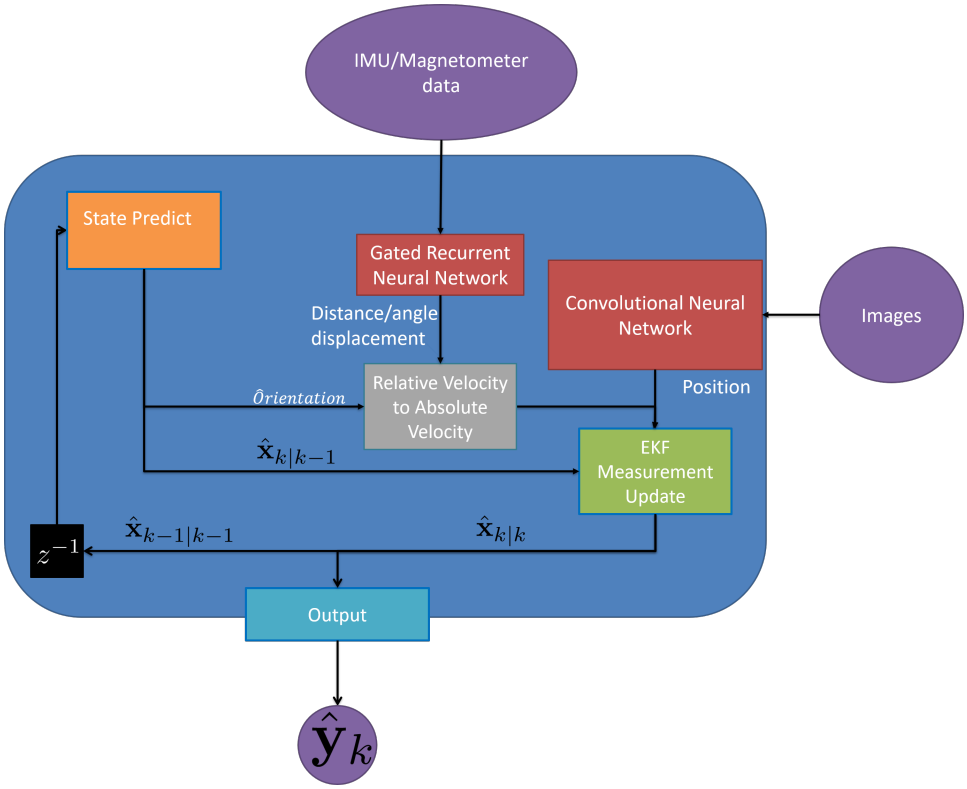

Figure 23: Extended Kalman filter diagram. Showcases the measurement update processes and how the input data feeds through the ANNs and into the EKF

The workflow of the EKF is highlighted in figure 23. This shows how the input data feeds through the corresponding ANN and into the measurement update step. Many of the specific parameters of the EKF such as the dynamics noise, measurement

noise, and FOGM characteristics are determined by the results of the ANN and will be discussed in the next chapter.

3.4 Chapter Summary

In this chapter the methods for for collecting and processing the training data was described. Three different image based ANNs using Glorot initialization and two methods using transfer based learning were discussed for localization based pedestrian navigation. Three different PDR model architectures were discussed for pedestrian navigation. Additionally, the PDR windowing inputs were discussed as well as three different training target types for determining the best results. Finally, a description of the EKF model combining the two ANN was described.

IV. Results and Analysis

This chapter analyzes results from the data collection, localization Artificial Neural Network (ANN), Pedestrian Dead Reckoning (PDR) ANN, and combined extended Kalman Filter (EKF) results for a single one hour test collection set. The primary objective is to determine the results of the EKF solution and see if any increase in capabilities is found by combining ANN derived solutions into a single output. In order to accomplish this both ANN based methods of navigation must be explored. The localization based ANN is analyzed first comparing the Glorot initialized models only trained on the urban data set to the transfer learning based methods. Then the results for the PDR are discussed in relation to previous results on the Oxford data set [42]. The best models are chosen and incorporated into the EKF where its results are then compared to the results of the individual ANN based methods. The models are all compared to there propogated final trajectory solution compared to the original trajectory. Errors in the north, east, and combined dimensions are explored to obtain the standard deviation of error of each model. Root mean square error (RMSE) is the primary method used to compare between model types. The standard deviation of the error is also considered when comparing similar models to achieve the optimal solution.

4.1 Data

The data was collected in a two square city block approximately 250m/820ft by 350m/1150ft within an urban downtown environment. One block included an open park covering around 60 percent with a large seven story building and an outdoor pavilion covering the remaining area. The second block consisted of a variety of sky rises ranging from three to twenty stories tall. This area was chosen to provide a range

of different types of structures and scenes within an urban area for the neural network to have a robust library of information to generalize for future collections. The time of day of collect ranged from 1000 to 1600 creating different illumination between data sets. An aerial view of the area with Global Positioning System (GPS) data points for one of the collections can be seen in Figure 24. The urban data set currently spans over three months with thirty collects ranging between forty minutes and 1.5 hours. The weather varied between sunny, cloudy, rainy, and snowy. Additionally, the environment changed slightly with construction work on the adjacent streets next to the walking path. Due to the construction, large equipment was in various images at different times and locations for about fifteen of the collections, but the path walked remained relatively unobstructed. Vegetation changes occurred during the collections due to the seasonal change from autumn to winter. The data is split between training, validation, and testing with twenty-seven collects used for training, two used for validation, and one used for testing. Generally a larger split is preferred between training, validation, and test, but additional data seemed to be needed for training due to poor initial model results that will be discussed in results section 4.2 for localization and 4.3 for PDR.

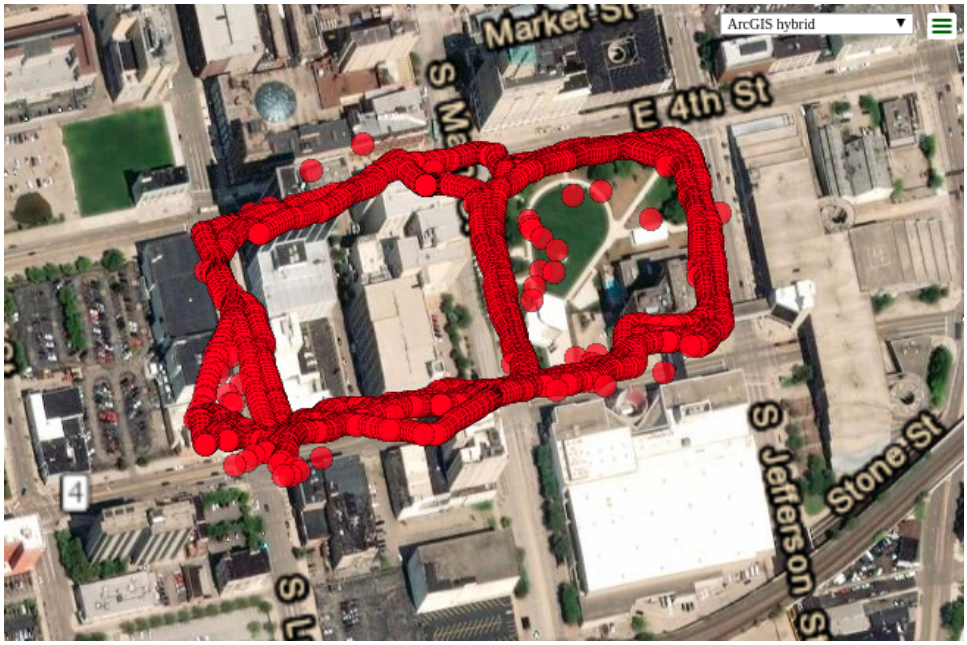

Figure 24: GPS data points for one collect in the urban environment. Note the multipath effect on the GPS points along the left street

The data collected had various errors within it that could potentially reduce training capabilities. The GPS data collected was the primary source of errors that needed to be resolved before being used. The original trajectory of one of the collections wihtout the satellite view is shown in Figure 25. Both Figure 24 and 25 showcase errors in the system such as multipath and position errors of the sensors. These two problems seemed to be the greatest source of GPS error. A relative GPS error was used to determine error instead of absolute error because there was no reference points for the data to calibrate to. The relative error is based on average distance travelled between each updates. It was assumed the error would take a Gaussian distribution and any error greater than five standard deviations from the mean would be removed. A large standard deviation was chosen because the update frequency of the GPS compared to the other sensors was already over one hundred times slower. Removing too many data points would create large time steps between each update during interpolation.

A local average of the remaining GPS points was taken to smooth out the trajectory further matching Figure 26. Another problem the GPS data encountered was signal multipath. The multipath of the signals resulted in a wandering trajectory on the north western vertical path closest to the origin in Figure 25. No action was taken to remedy this effect creating variance in the training targets position. A more precise geotagging of location could remedy this solution, but resources weren't available. The multipath effects on the routes varied between each collect and believed to create a Gaussian like distribution around the actual location on that street. This error was deemed acceptable and the neural networks may be able to generalize an average of the positions as its prediction.

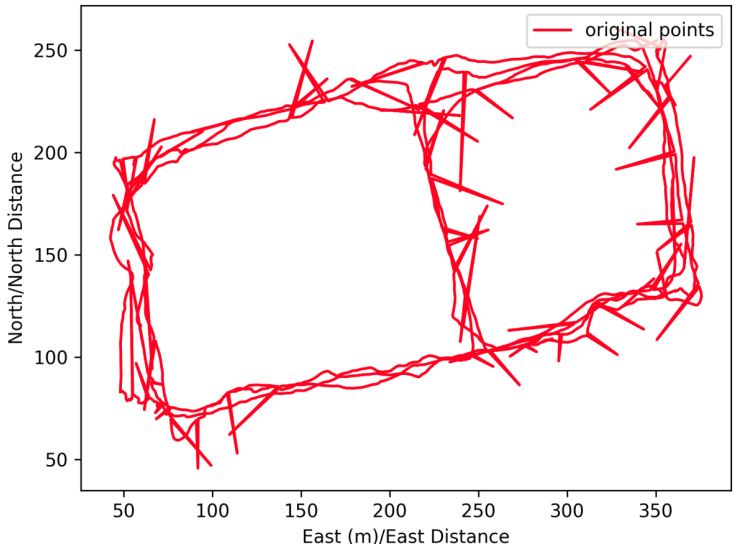

Figure 25: GPS based position trajectory before error processing has occured

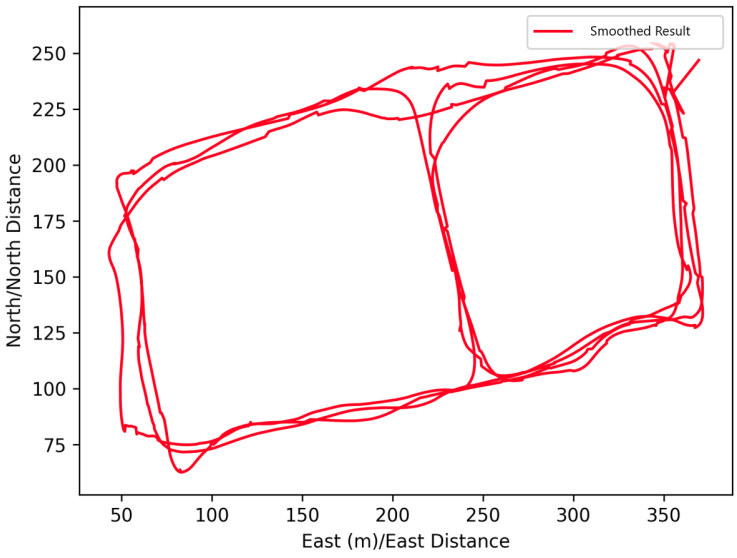

Figure 26: GPS based position trajectory with errors smoothed

The key image frames processed from the mp4 videos had minimal degradation and an example can be seen in Figure 27. As previously mentioned the images contained slight blurring and rotation about the vertical and horizontal axis. This is an accurate representation of real world image collections and should help the neural network generalize the solution to accommodate these variations.

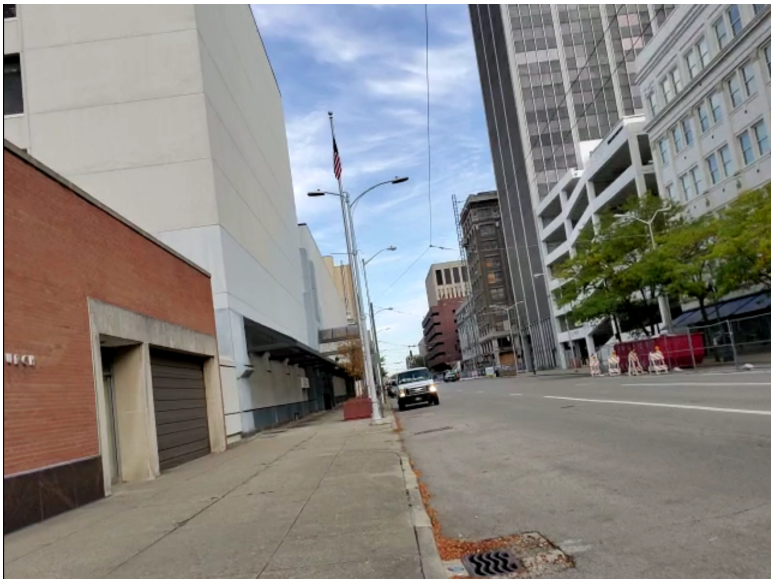

Figure 27: Image example from a collection data used to train localization neural networks

The inertial measurement unit (IMU) and magnetometer sensor data show relative similarities throughout all of the collections. Important information from this complete collection shown in Figure 28 are the points where it reamains flat and the two large spikes. These points indicate where the data collector was standing still for the flat spot and running for the spikes. The gyroscope plots are similar to the acceleration plot. If the magnetometer data shown in Figure 29 was not being affected by ferrous objects the data would be less noisy. Additionally, it would be expected to have a similar reading as the same area is visited during the collect. There are some key similarities in Figure 29 that make it appear as though it is picking up similar readings. There is still a large degree of noise in the magnetometer data set that is most likely the cause of ferrous objects nearby the sensor. This data could have been smoothed to try to remove this noise, but the data could have held small bits of information that is not noticeable by human interpretation and may provide benefits to the neural network during training.

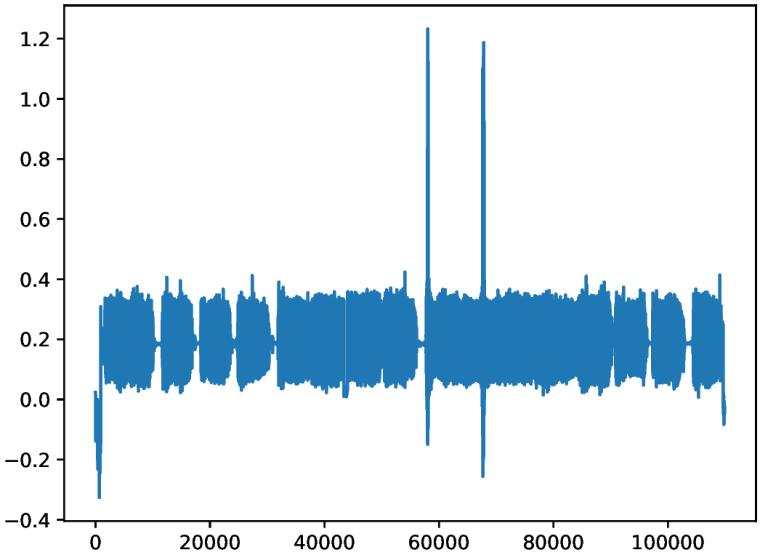

Figure 28: Accelerometer readings for the x-axis for the duration of one collect. Two spikes are where data collector was running and flat spots indicate no movement.

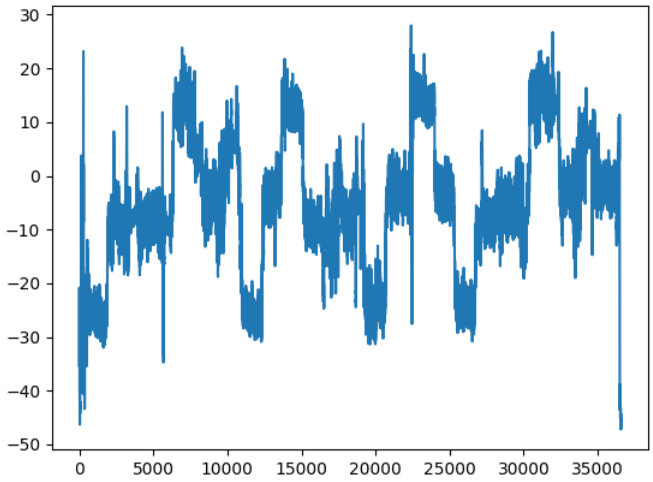

Figure 29: Magnetometer readings for the x-axis for the duration of one collect

4.2 Localization

Model architectures results are split into two different initialization types including standard Glorot which doesn't utilize training of the weights apriori and transfer learning which trained the model weights on another data set to initialize the weights. ANN results utilizing only the urban data set for training with Glorot initialization are explored first and then transfer learning based initialized weights are explored.

4.2.1 Image Based Glorot Initialized models

The ANN architectures with Glorot initialization included basic sequential, residual, and WideNet style base models that were altered in various ways to try to achieve greater optimization. There are large commonalities for each of the three types that were done in every training method. Learning rate always started at 10^{-4} with a learning rate reduction on plateau. This reduced the optimizer learning rate by twenty percent if the validation loss did not reduce by a noticeable portion within ten epochs. The base models of the three types were initially trained for 500 epochs, batch size of eight images, and 400 batches per epoch. Each of the base models had the lowest number of parameters to try to increase the computation speed and allow it to operate on a cell phone. After initial training if models did not converge on an optimal solution that tracked the trajectory, variations of the models were tested for fifty epochs. The epoch size could be reduced drastically because the type of solutions the network generally achieved occurred within the first twenty to thirty epochs. A more thorough description of the remaining training and results are described separately for the basic sequential, residual network, and widenet model types. The original basic sequential model architecture experimented with is shown in Figure 30. This model used a rectified linear unit (ReLU) activation function, root mean squared propagation (RMSProp) optimizer, Glorot initialization, 6 convolution layers, and

feature maps for each layer of $16, 16, 64, 64, 128, 128$. Variations from this original architecture exist and the parameters adjusted for the basic sequential model are listed in table 1. Every combination of this table was tested and experimented to try to obtain optimal solutions. At the beginning of experimentation this smaller network was tested to increase speed of training, have the ability to compute a solution faster, and be able to be used on a cell phone. This provided the benefit of quick analysis to try to find a baseline to expand model hyper-parameters while also determining if smaller networks held enough parameters to approximate the localization function. These smaller weight models were not able to obtain results other than an average value for all predictions indicating there may not be enough capacity in these smaller networks.

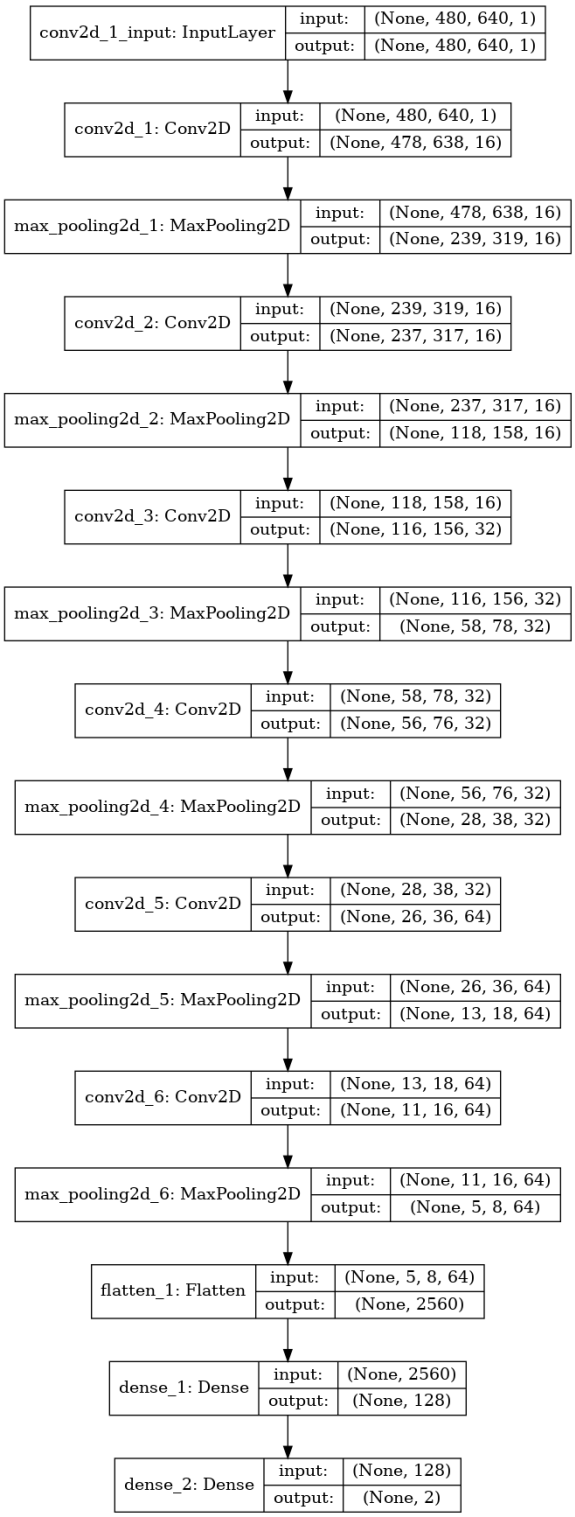

Figure 30: Basic sequential model base architecture showcasing the minimum layer and feature map sizes used.

Table 1: Hyper parameters tested for basic sequential model. Every combination was tested on this algorithm for at least 50 epochs.

Activation Functions	ReLU	tanh	sigmoid
Optimizer	RMSProp	ADAM	SGD
Convolution layers	6	12	18
Feature maps per layer group	$(32, 64, 96)_1$ $(64, 128, 256)_4$	$(32, 64, 96)_2$ $(128, 256, 512)_5$	$(64, 128, 256)_3$ $(128, 256, 512)_6$
Image channels	3 (RGB)	1 (grayscale)	

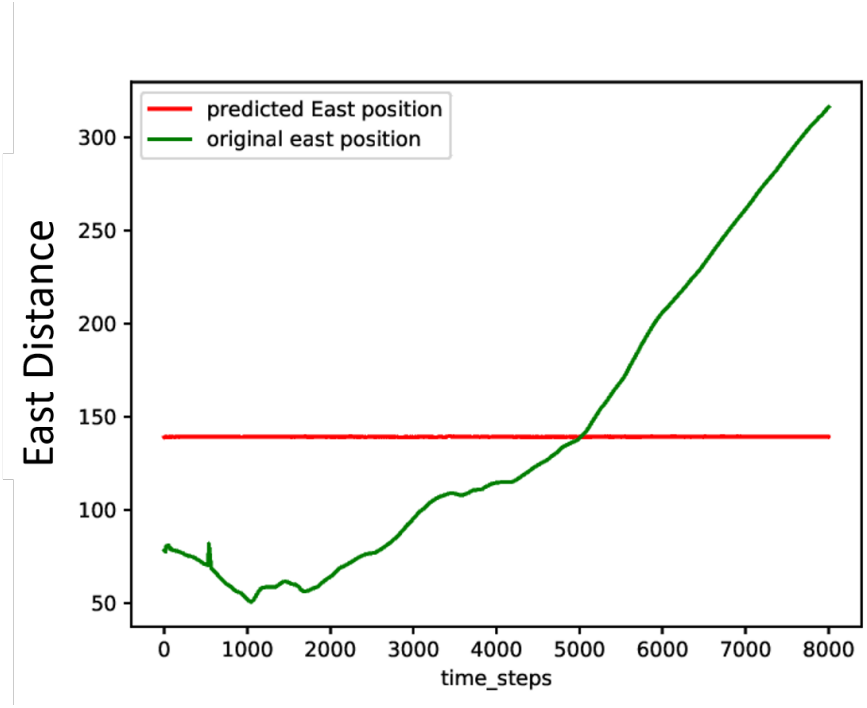

Figure 31: Basic sequential model East error position for four minutes of test collect. Model had seven convolution layers, ReLU activation, RMSProp optimizer, and Glorot initialization

In order to test notion of not enough capacity in the network additional convolution layers were added in between each maxpooling layer and feature sizes were

increased. These models achieved similar results as the previous models and obtained an average predicted position for all images tested. All variations of the parameters listed in table 1 provided similar results. An example of the prediction trajectory compared to the true trajectory of the North position over the course of four minutes of the test collect is shown in Figure 31. This prediction remains constant for all images provided for each time step. When comparing north and east trajectories simultaneously the prediction becomes a single point on the map and is not easily recognizable in an image. After increasing the model capacity, trying different optimization and activation functions, and testing input channels the basic sequential model was scrapped and additional more complex model architectures were tested. The next model tested was the residual network model.

The residual network inspired architectures start from the base model. Figure 32 shows the initial input layers connected to one residual block. After each residual block in the base model a connection to a maxpooling layer and then a dropout layer occurs before entering another residual block. These connections occur six times before entering the final two dense layers. Initially testing was done on the smaller base model to see if the residual connection alone could change the solution outcome. This model utilized a ReLU activation function for all layers except the final dense layer which used a linear activation. Additionally, the RMSProp optimizer was used on this model. Regularization was included with a dropout of thirty percent and batch normalization after each convolution layer. These training parameters were used as well as the ones already listed to test the initial results of a residual type network. The initial results for the base model of the residual networks proved to be similar to the basic sequential model. The base model only learned to output an average value of the predicted positions of all input images. The base residual network models predicted and true positions for the east dimension are compared in

Figure 33.

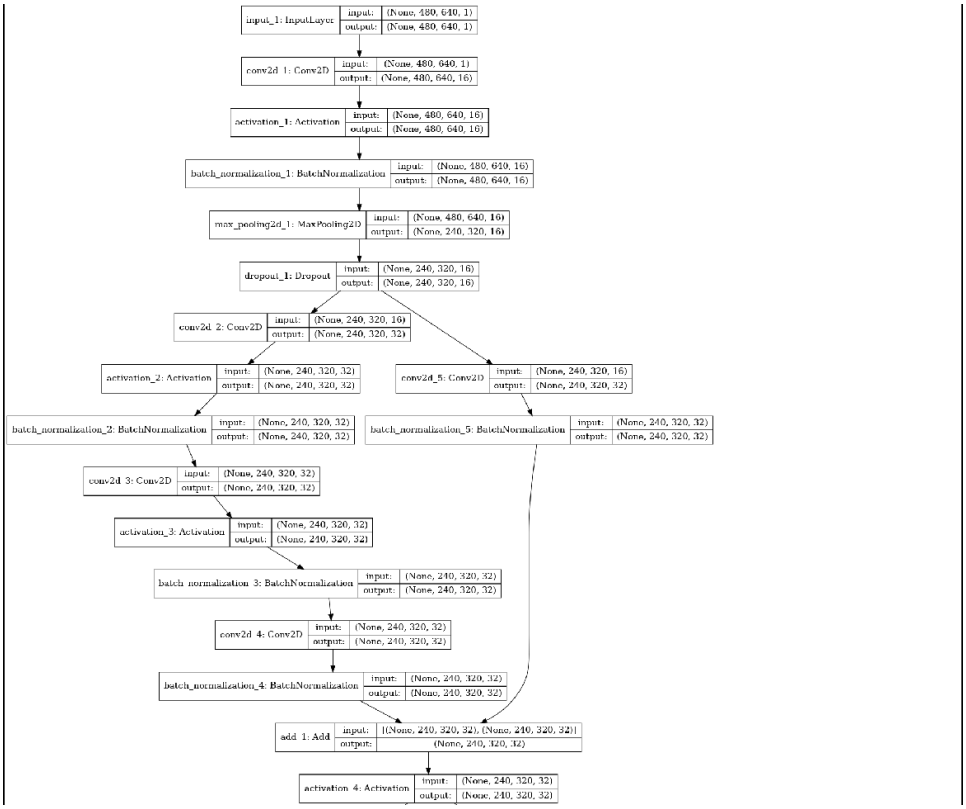

Figure 32: Building block of the base residual network. Showcasing the input block and initial layers as well as one residual block containing two convolution layers.

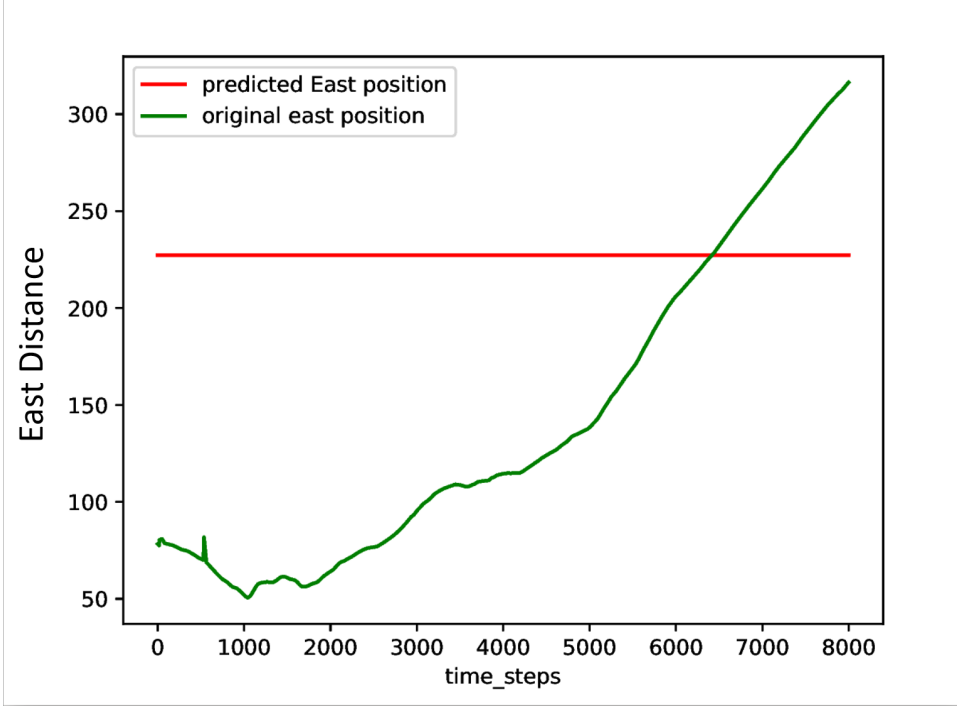

Figure 33: Residual network models east error position for four minutes of test collect

The base model is one of the shallower versions and subsequent models went deeper with additional layers within the blocks and entire blocks between the maxpooling layers. The hyper parameters and the variations tested that deviate from this basic model are showcased in table 2. Every combination of hyper-parameters in table 2 were trained. The tested models first went deeper before testing additional activation or optimazation functions. These deeper models were used to more closely mimic the RESNET50 model and potentially improve results by increasing the number of weights in the network and creating deeper networks. All of the variations of the models tested had predictions similar to Figure 33 where the model would predict a single location for all images.

Table 2: Hyper parameters tested for RESNET model. Every item was tested at least once, but not every combination was tested.

Activation Functions	ReLU	tanh	sigmoid
Optimizer	RMSProp	ADAM	SGD
number of residual blocks	6	12	18
convolution layers per residual block	2	3	4

The results from both the basic sequential and residual base models and their variations were not performing well so only one version of the widenet was tested. This is because widenet has similarities to both networks and if the base model wasn't going to converge on an solution, variations most likely wouldn't either. The widenet base model results for the east position are shown in Figure 34. This model had six residual blocks with two convolution layers per block. The model had feature map sizes of $256, 256, 512, 512, 1024, 1024, 2048, 2048$. The features map size is four times the size of the largest basic sequential for each convolution layer. Unfortunately this model predicted an average position for all images like the previous models as shown in Figure 34

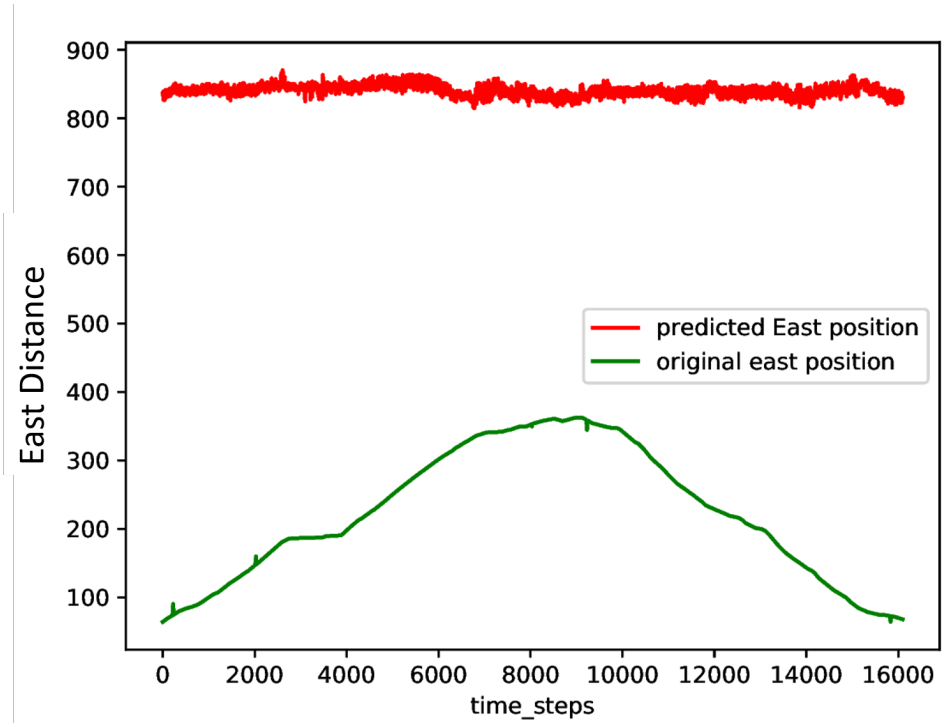

Figure 34: Widenetwork predicted compared to the original results for the east positions over four minutes of collects

All of the models tested with a Glorot initialization did not prove to be usable for this navigation solution. Many different parameters were adjusted to try to achieve an optimal solution. Position averages appeared to be learned within the first fifteen to thirty epochs and continued to predict values extremely close for the duration of training. For example the true north and east location for training targets range between 50m and 350m in the north, east, down (NED) frame and the predicted solution would be 170m on the north axis and 200m on the east axis for every image within the collection. Some potential causes for this could be due to the size of the networks not having enough capacity to learn the models, the urban data set didn't have enough variation in it to generalize a solution, or potentially the Glorot initialization of the weights placed the network into a local minimum solution that it couldn't escape from. To showcase these results the east predicted position compared

to the original position for three base models for popular techniques are shown in Figure 31, 33, and 34. Since these models were trained with varying number of weights, it seemed to minimize the theory the network capacity was the problem to training. The next step is to test networks that take advantage of transfer learning. If these models obtain results it may indicate the urban data set did not have either enough variation or sample size in the training set to converge on an adequate solution.

4.2.2 Image based Transfer learning initialization

Two base models were tested using transfer learning techniques including the Xception and VGG16 model. Variations in training weights were used to try to achieve more optimal results. Each variation had the weights pre-trained on the ImageNet data provided by the keras library. In order to train the additional dense layers added to convert to the regression output training was completed keeping all layers of the original network frozen. The models were then trained for 20 epochs with the urban data set to adjust the new layers based off of the ImageNet weights and the urban data set. Once this initial training was complete, models were then retrained with various layer weights unfrozen. These new unfrozen weights were retrained on the current data set between 400 and 1000 epochs, 32 images per batch, and 400 batches per epoch. The unfrozen layers were chosen to always start after a maxpooling was done. This contained changes to the network to similar spatial regions of the image. The layers that remained frozen are particular to each model type and explained in the following paragraphs. Once training was completed the lowest value for the validation loss of each epoch was chosen for the final model. The best epoch performance was based off of the mean squared error (MSE) loss of the validation sets. The outputs were tested for quantitative results including position error per time step, the complete RMSE of the test collect, and the error

variance. By comparing these values between all models the model with the lowest values for complete RMSE was chosen. If multiple models were similar the remaining two factors would be taken into consideration in determining the best model. One model variation was chosen from both the Xception and VGG16 models to explore in more detail.

The results for all Xception models tested are shown in table 3. The model had many of the exit flow block removed and adjustments are shown in Figure 35. The Xception model performed best when the layer weights from the input to layer seventy-six remained unchanged and the remaining layer weights were retrained with the urban data set. This corresponds to the middle flow models block, fifth repetition, first separable convolution layer. The other two tested weights were unfrozen for layer fifty-six and ninety-six, which corresponds to the third middle flow block and the seventh middle flow blocks first seperable convolution layers.

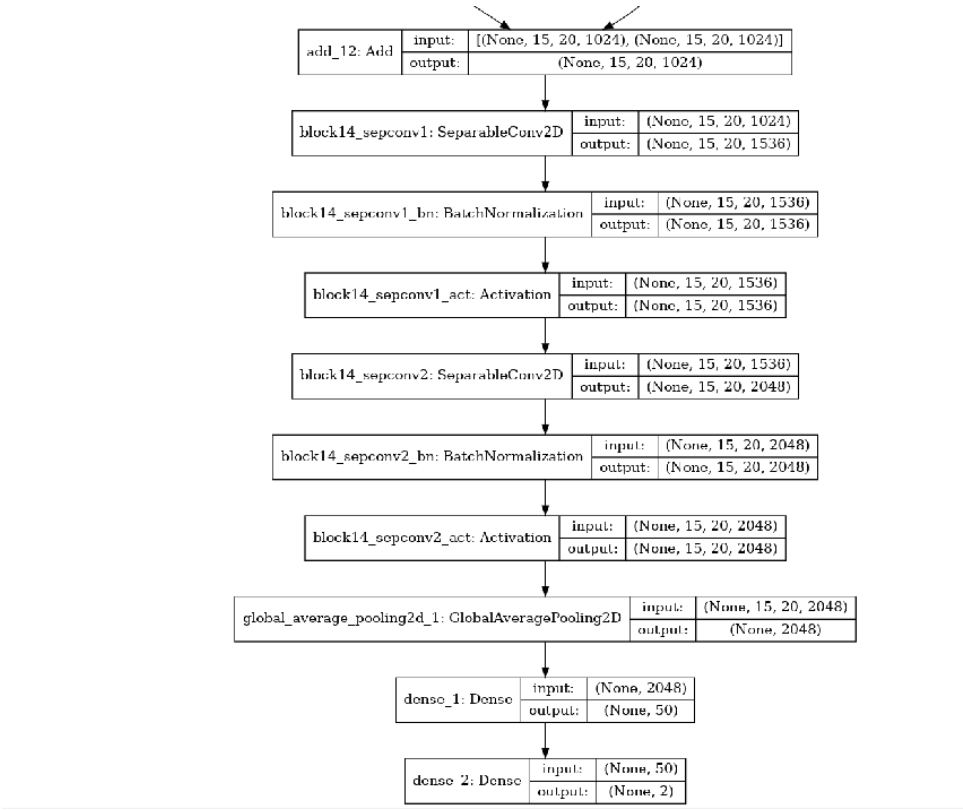

Figure 35: The adjustments made to the final exit block of the Xception model from figure **??**

The following results exploration are in reference to the model unfrozen on layer seventy six only, but all models were tested similarly. The unfrozen layers were retrained for 1000 epochs and during training the loss continued to decrease for the training MSE shown in Figure 36. The MSE validation loss values for this model during training can be seen in Figure 37. The validation loss values remained relatively stagnant and didn't move considerably. The validation loss values have a low starting point of 0.023 MSE with a slightly decreasing trajectory over the first seventy epochs reducing validation loss to an average of 0.0125 MSE for remaining epochs. However, the MSE loss values are extremely noisy from epoch to epoch and never get below 0.007 MSE. There is a difference in the models training and vali-

dation loss MSE curves. The training starts at a higher value and never reaches as low of an MSE as the validation result. This difference could potentially be due to the validation images matched the network better than the training batches. The noisy nature of the validation loss function results could indicate the neural network is either training and updating the weights to generalize to the entire urban data set or the similarity between images at different points on the path cause the model to predict values further away from the true value. Even with these differences the best performing epoch is still chosen based on the validation loss curve. The epoch with the lowest MSE error is chosen for testing and final results.

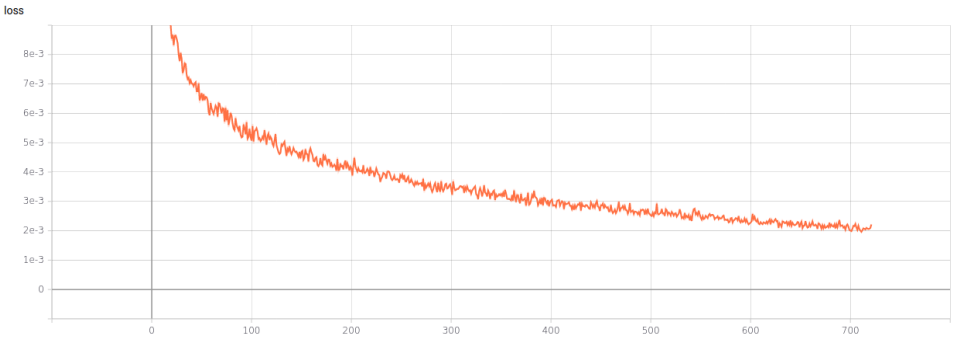

Figure 36: Xception training loss results per epoch for the highest performing Xception model.

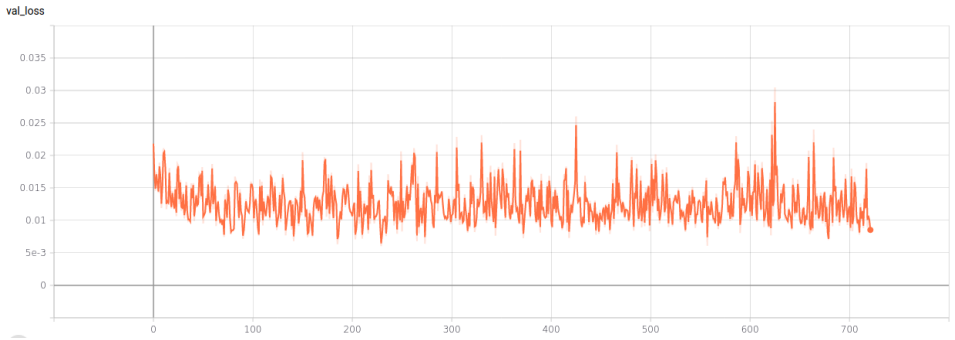

Figure 37: Xception validation loss results per epoch for the highest performing Xception model.

The best performing epoch for this model was chosen to be epoch North and east position errors for the Xception model are shown in Figure 38 and have an RMSE of 32.67m and 50.5m as shown in table 3. Although the position errors were large and noisy, the predicted values maintained similar overall position trajectory to the original trajectory as shown in Figure 39 and 40. Moreover, the position errors are cyclical as shown in Figure 39 for the north dimension and Figure 38 for the east dimension. The estimates are biased towards the center of the data set. The predicted values undershoot the position when it has a high position value and overshoots when it has a lower position value. This is potentially a reason why the validation loss was sporadic and noisy as well. Observing both positions mapped together against the original trajectory the errors become more prominent. The whole state plot is shown in Figure 41 and has an overall RMSE of 60.2m. The center bias errors of the two position dimensions are able to be seen when comparing the predicted trajectory to the tre trajectory.

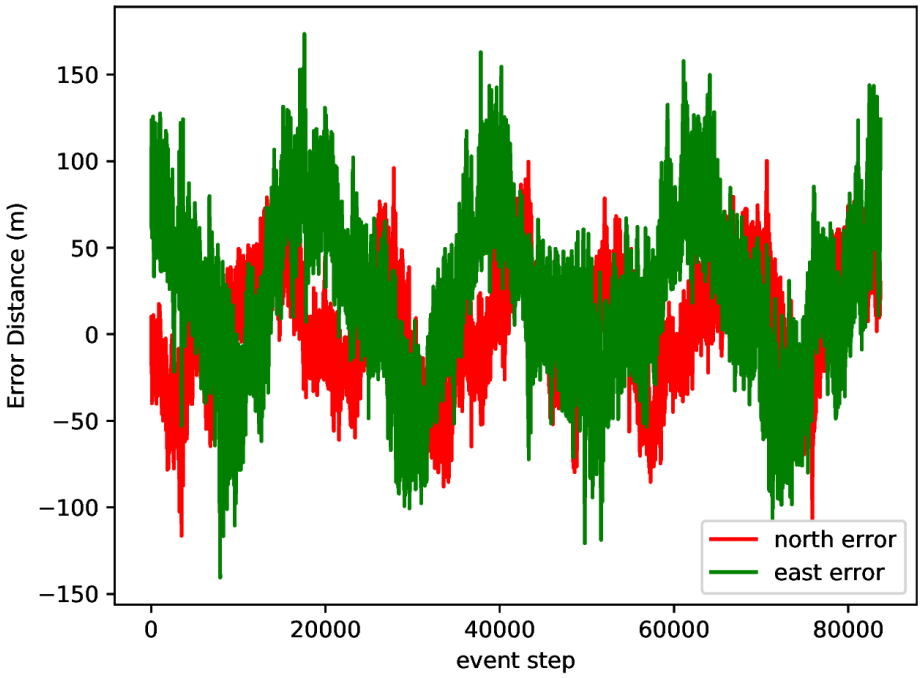

Figure 38: Xception model position variance for North and East over the course of the single test collect

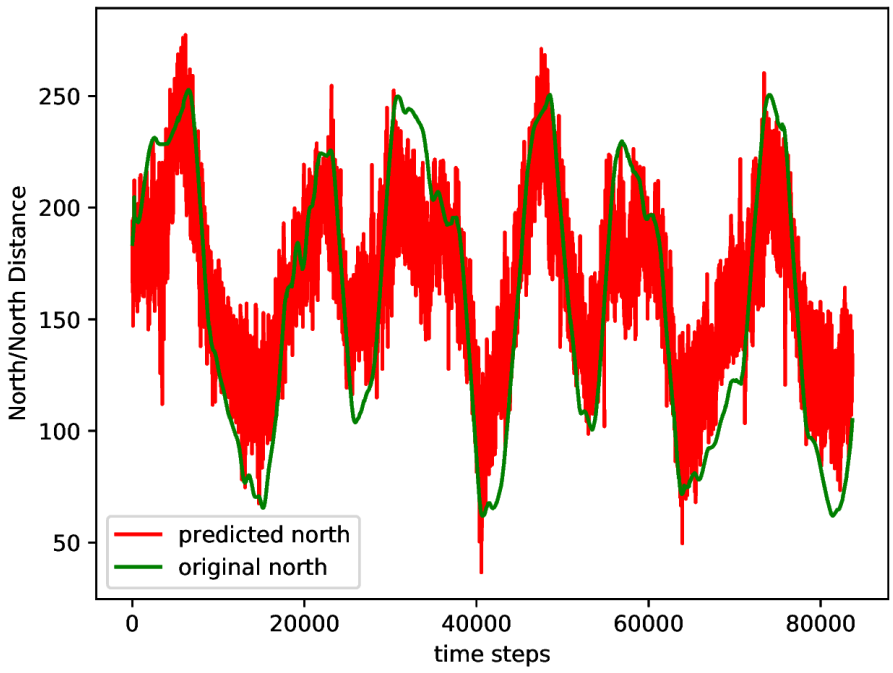

Figure 39: Xception model north position truth vs predicted

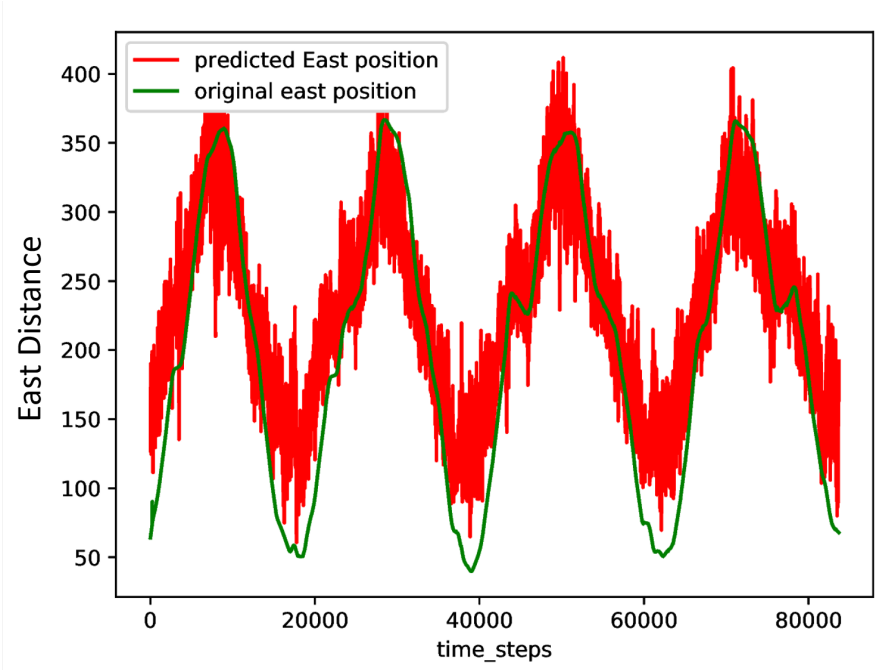

Figure 40: Xception model east position truth vs predicted

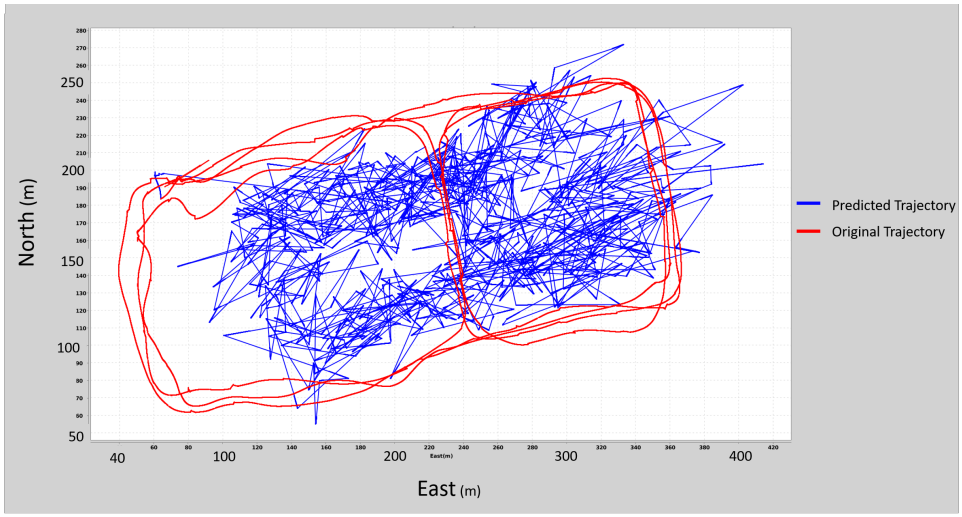

Figure 41: whole state plot of Xception localization model

Results for all VGG16 variations tested are listed in table 3. The best performing VGG16 model results are explored, but all variations were explored the same way. The best VGG16 model is shown in Figure 42. This model has the last three fully connected layers and the softmax layer from Figure 13 section 2.2.11.1 removed and a global average pooling and two dense layers added. The first dense layer had a ReLU activation and the second dense layer had a linear activation. This model trained for twenty epochs with all VGG16 layer weights frozen and not training and the added layers updating. This allows the bottom two layers to be trained on the urban data set while maintaining feature map information from the ImageNet data set. During this initial training RMSProp was used for the optimizer with a learning rate of 0.0001, batch size of thirty-two, and 200 batches per epoch. After this initial training block3 convolution layer three through the dense layers weights were released for training. These additional layers were unfrozen to allow the model to fine tune to the urban data set as there may be less generalization needed. These additional layers were trained for 1000 epochs with a batch size of 32, 600 batches per epoch. Additionally, a stochastic gradient decent (SGD) optimizer was used to

tune the weights with a learning weight starting at 0.0001 and a momentum factor of 0.9 based on the validation MSE. The learning rate had a reduction on it if the validation loss plateaued for ten epochs by a factor of 0.8. All other factors within the keras layers remained unchanged from their preset values.

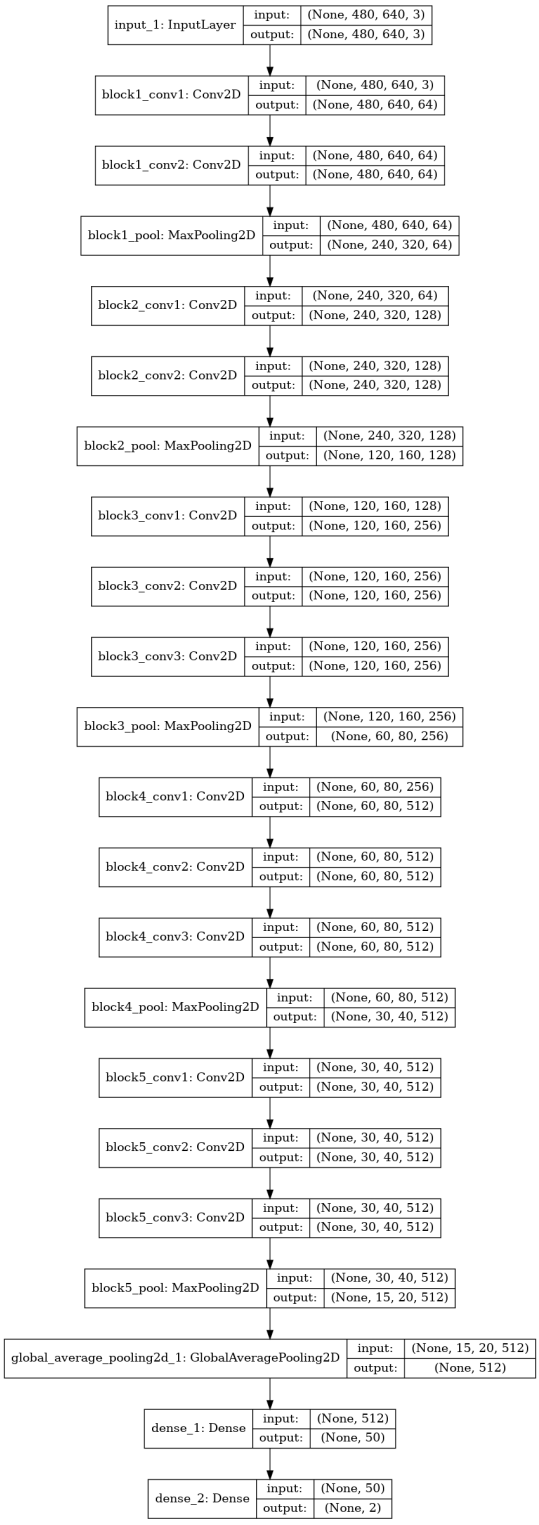

Figure 42: VGG16 model with final three fully connected and softmax layer removed. Global average and two dense layers added

During this retraining the MSE of the training loss decreased for the first 300 epochs and then leveled off as shown in Figure 43. However, the validation loss shown in Figure 44 has a minimal decreasing trajectory that is unnoticeable after epoch fifty. The validation loss values are also extremely noisy. The lowest epoch result was chosen for testing and obtaining the final results. Three epochs had similar MSE results, but two were before epoch 300 and hcosen not to be used because the training loss was still decreasing and the validation stayed relatively flat. Epoch 577 was determined to be the lowest validation loss and chosen for further testing.

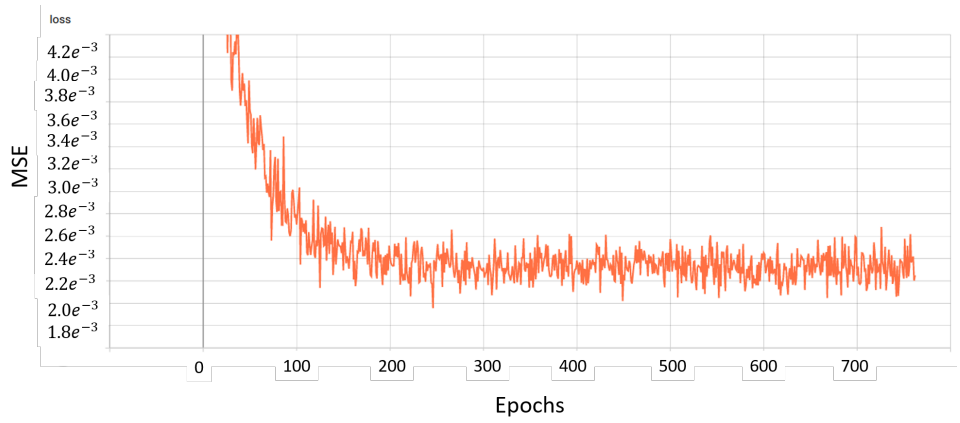

Figure 43: MSE training loss for the VGG16 model during 1000 epochs of training on the highest performing model.

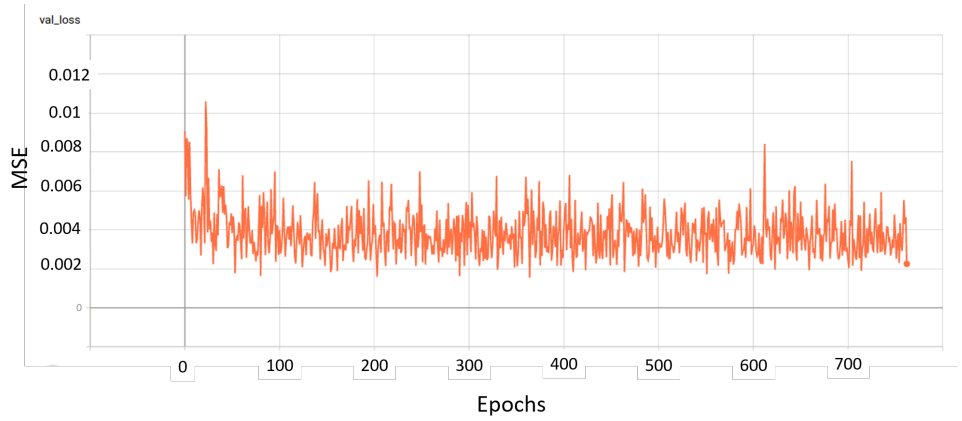

Figure 44: MSE Validation loss for the VGG16 model during 1000 epochs of training on the highest performing model.

The neural network epoch chosen was able to obtain an RMSE error of 20.6m for north positions and 29.2m for east positions. The error plots can be seen in Figure 47. The error stays bounded with an average error close to zero. The error has small oscillation in it where the network biased toward the center values when reaching the minimum and maximum points in the north and east trajectories. The network may have been more center biased due to finding an average value in the beginning of training and then trying to push the predictions closer to the actual values to lower the loss. This results in a more closely matched solution to the positions as shown in whole state plots of Figure 48. The standard deviation of the error equates to $20.6m$ for the north position and $28.4m$ for the east position.

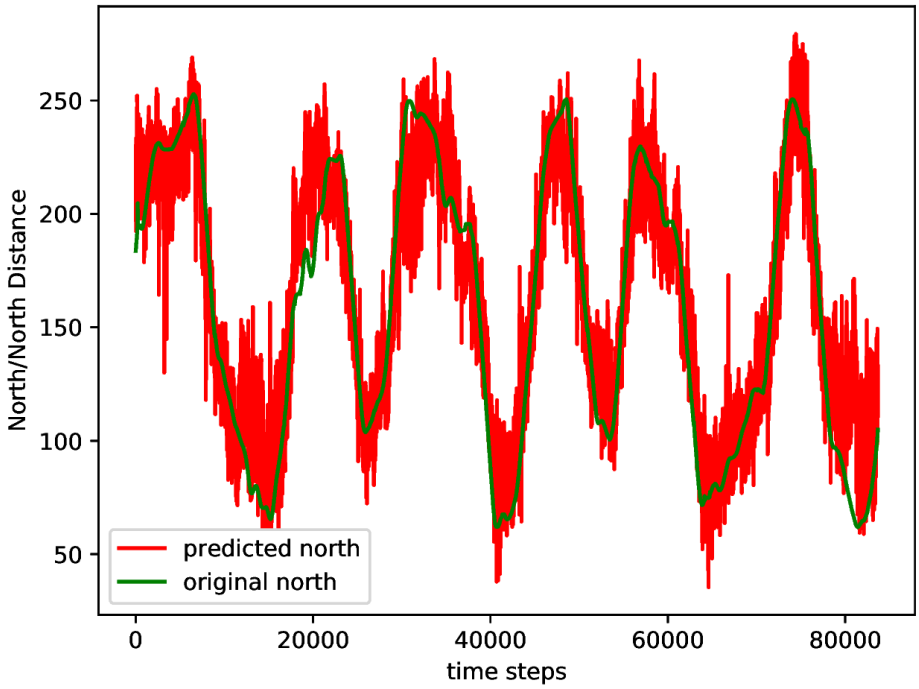

Figure 45: VGG16 model north position truth vs predicted

position.png position.png

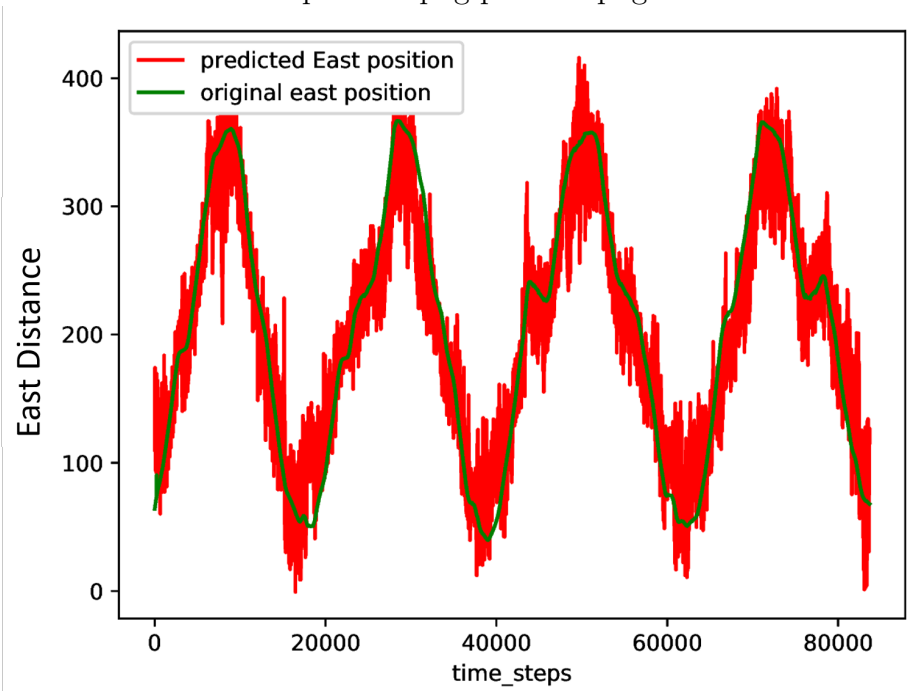

Figure 46: VGG16 model east position truth vs predicted

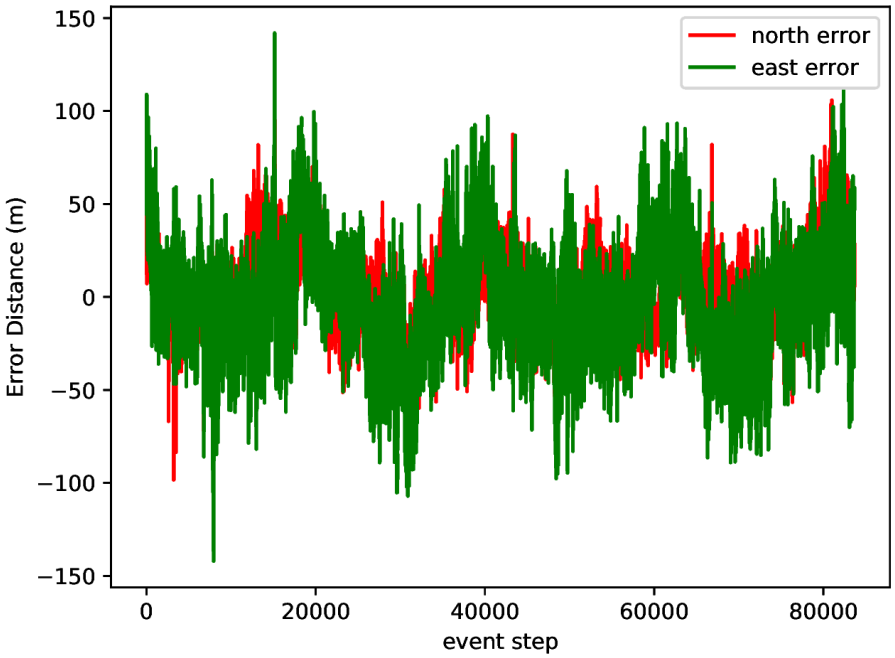

Figure 47: VGG16 position variance for North and East over time

The whole state plot is shown in figure 48. From this whole state plot the navigation solution is hard to track exactly where the solution is going.

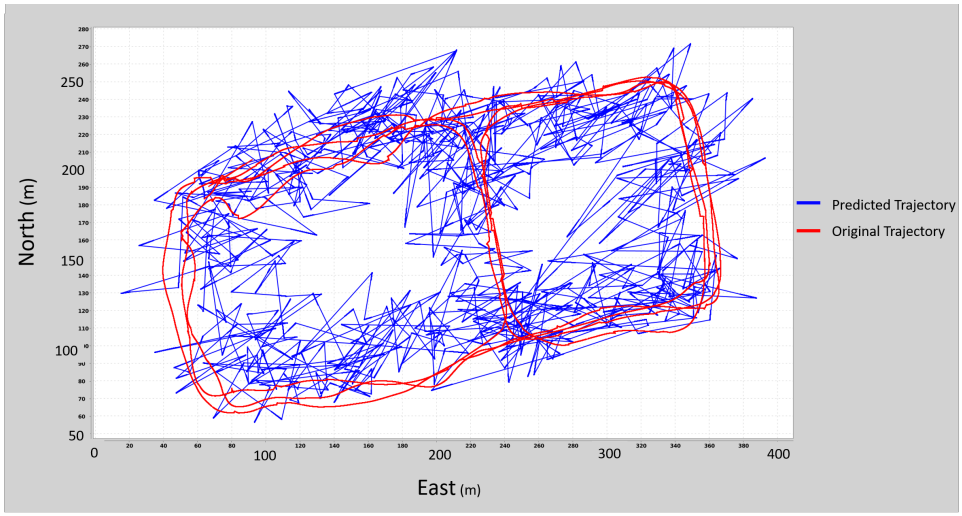

Figure 48: Whole state plot for VGG16 model compared to the true trajectory

Clustering Data Results After testing the transfer learning based techniques with

regression the idea was brought up to try to cluster the data points into discreet locations. This would eliminate any center bias the regression problem had as well as test the transfer learning networks against a classification problem they were originally developed to handle. Error is intrinsically placed in the system due to the cluster center position, but this error could potentially be much smaller than the regression problem if it has a high accuracy rate. The data clusters were tested ranging from 100 to 300 different clusters. Using 100 clusters was chosen to test the data against because the higher the cluster amount the more data points would be given to points effected by the multipath. The clustering of the map can be seen in Figure 50. The bins did not all end up being evenly distributed. To account for this in the testing each value was given a weight that would affect how much it changed the results. The histogram of the clustered data can be seen in Figure **??**.

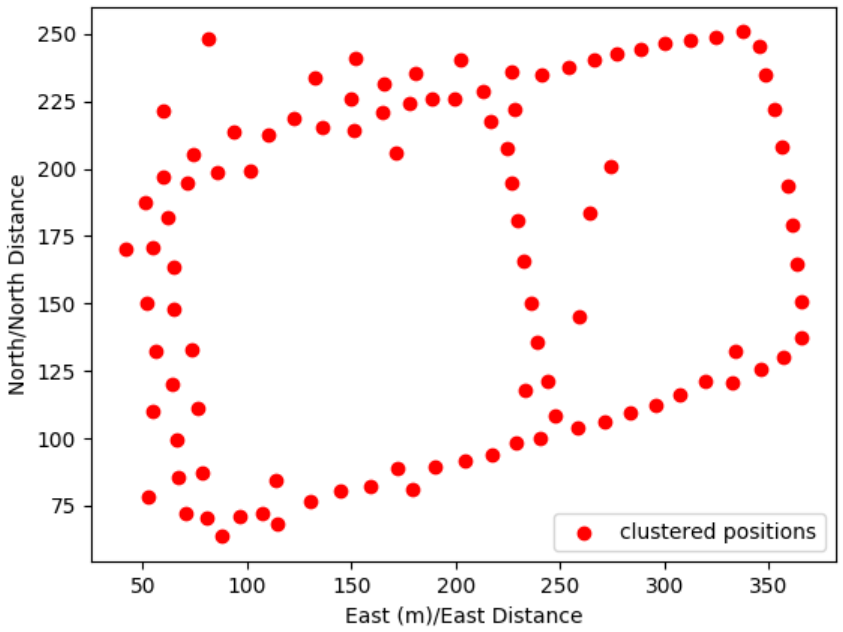

Figure 49: Trajectory points clustered into 100 discreet locations.

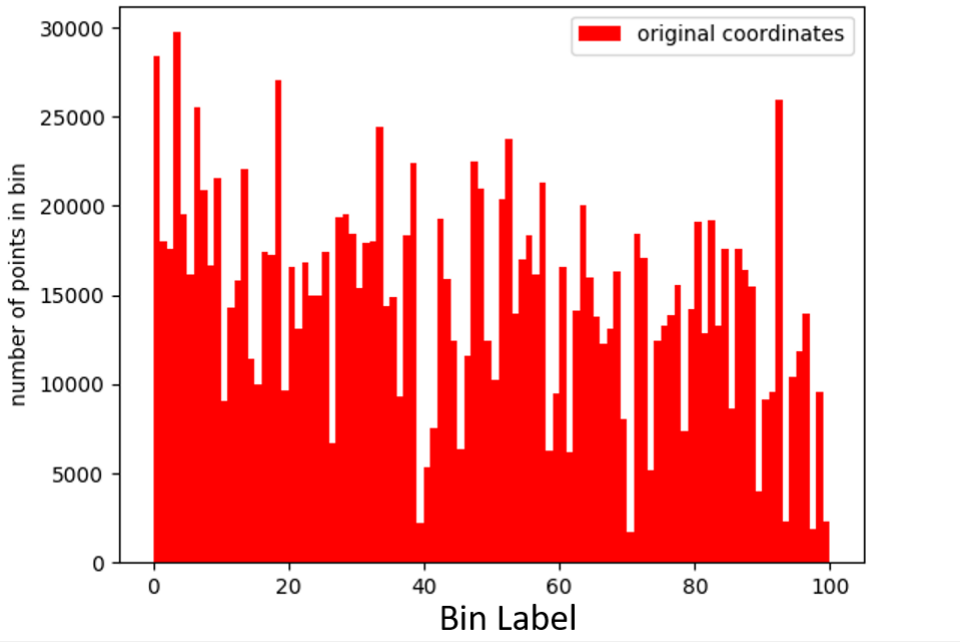

Figure 50: Histogram of the different clustered points

F With the data clustered both the transfer learning best performing models were retrained using a classification based approach. The VGG16 model was the highest performing regression results, but under performed during training. The Accuracy for the training never achieved greater than thirty percent accuracy on the training set and the validation set never got higher than ten percent. The WideNet model tested achieved a higher classification success rate of fifty percent on training but never got above twenty percent on the training. The only model that moved forward to determining the RMSE of the trajectory was the Xception model. The training loss for the Xception model is shown in Figure 51

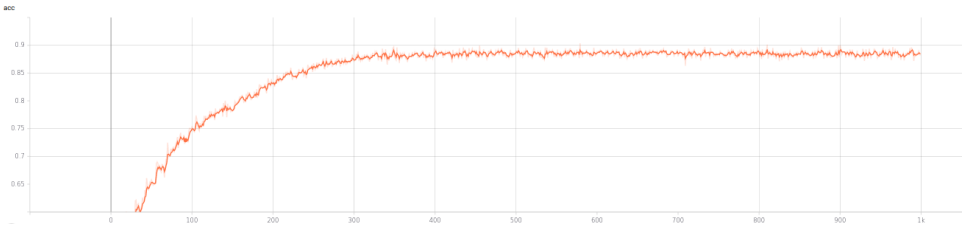

Figure 51: Accuracy curve for training set with cluster based localization Xception model

The training accuracy continued to improve for the first 400 epochs before stabilizing around 89% accuracy. Although the training accuracy showed improvement the validation accuracy jumped around 30 percent. This could be due to images being placed in nearby clusters as the results are only based off of the highest probability model. The validation accuracy can be seen in Figure 52

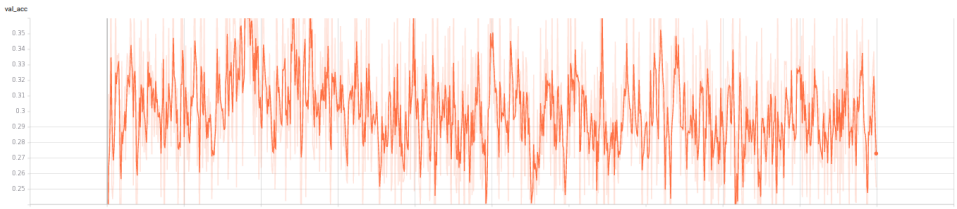

Figure 52: Accuracy curve for validation set with cluster based localization Xception model

With the highest performing accuracy the model was tested for RMSE values to compare to the regression model. Only the highest probability classification was used in determining the error and the labels were converted to their corresponding center points. The model was able to obtain a total RMSE of 61.05m shown in table 3. This result didn't perform better than five of the six regression models. If the probability of the classifications were taken into account and the top two or three label center points were combined to create an average it might improve the results. However, the results seem consistent with the regression error on the Xception model. One

thing to point out though is that the WideNet neural network was able to train to a classification problem with no prior transfer learning. This shows the ANNs had a hard time learning a regression problem with only the urban data set.

4.2.3 Localization Result Summary

Results varied for the ANNs tested for localization. None of the Glorot initialized models were able to obtain a navigation solution. However, both of the transfer learning cases were able to obtain a localization solution on all variations. All results are listed in table 3 for comparison. The VGG16 model performed better on all counts when compared to the Xception model even though the Xception has better metrics on the ImageNet data set classification problem. This could potentially be due to the Xceptions network depth and having trouble readjusting to the new data set or the Xception model may have performed better on classifications that were not seen in the urban data set. The VGG16 model explored was chosen to be used in the EKF for position measurement updates based on its RMSE. With both transfer learning models and their variations obtaining a solution it gives more credence to the possibility the urban set isn't adequate for training on its own. With a localization solution determined the results of a PDR solution needs to be explored next.

Table 3: localization results for transfer learning based models

Models	Middle flow unfrozen start layer	North RMSE	East RMSE	Complete RMSE	Error Standard deviation
Xception	Block4 Sep1	32.01m	47.80m	57.53m	55.62m
Xception	Block5 Sep1	32.67m	49.90m	59.64m	55.07m
‘ Xception	Block6 Sep1	39.85m	52.61m	66.00m	62.85m
Xception Clustered	Block4 Sep1	31.33m	52.47m	61.05m	
VGG16	7-15	20.83m	28.56m	35.35m	35.09m
VGG16	11-15	22.35m	28.74m	36.41m	36.23m
VGG16	15	23.96m	28.79m	37.46m	36.42m

4.3 Pedestrian Dead Reckoning

Initially models were tested against the Oxford dataset as a benchmark for comparison as well as dialing in known working models. The highest performing model architectures were then used in training on the urban data set. In this initial testing only some of the parameters discussed in section 3.3.2 were tested. This includes the windowing effect on the input data and the single final output step for the entire input as shown in output one of Figure 22. Additionally, various hyperparameters were tested including the activation function, types of layers, number of layers, how long the input lookback would be, number of units, and drop out rate. After this initial testing two models were chosen to initially test on the urban data set. The first one is a two layer Gated Recurrent Unit (GRU) architecture shown in Figure 53. The results of this model had a bit of drifting in the beginning but stabilized and followed similar paths to the original trajectory as shown in Figure 54. The second model that performed better during testing used CuDNN Long Short-Term Memory (LSTM) layers. These layers are similar to normal LSTM layers except they only allow a hyperbolic tangent function (tanh) activation function and no drop out. The other important hyper-parameters in this model were a RMSProp optimization with a learning rate of 0.001, 200 input lookback, and 256 units. The architecture for this model is shown in Figure 55 and the results are shown in Figure 56

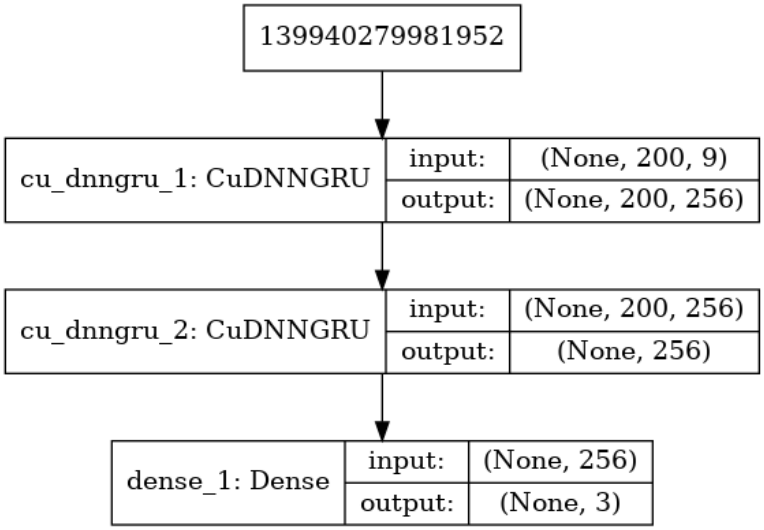

Figure 53: Neural network results for best GRU network on the Oxford PDR data set

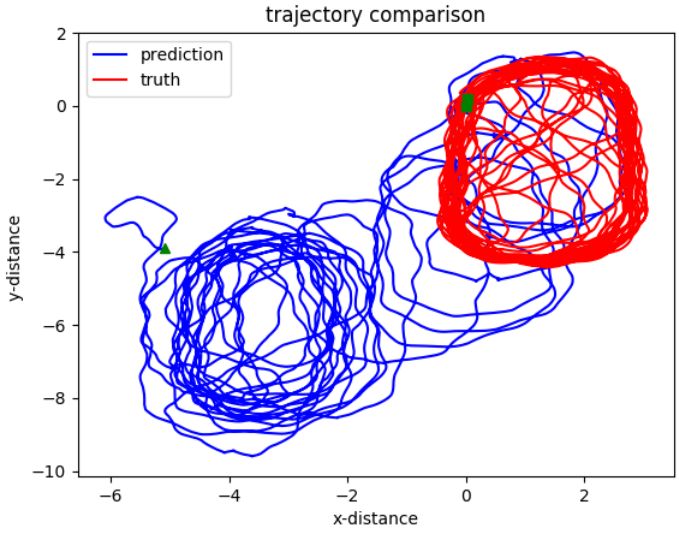

Figure 54: Trajectory for GRU neural network architecture trained on Oxford PDR data set.

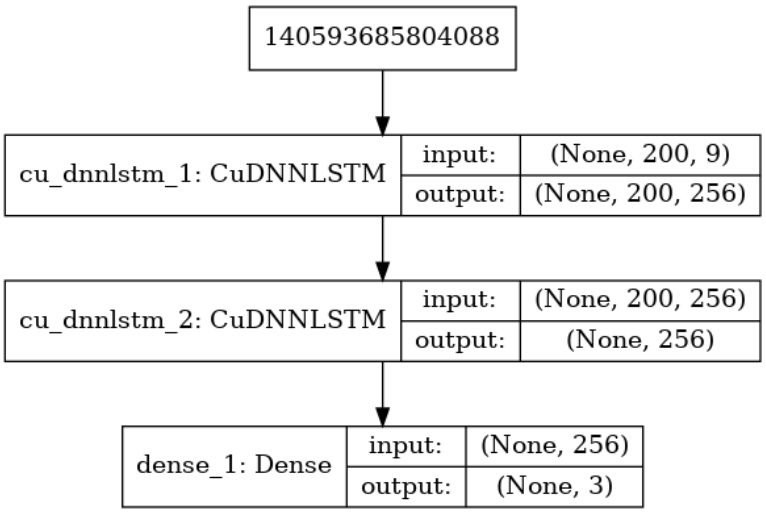

Figure 55: Neural network results for best overall and best LSTM network on the Oxford PDR data set

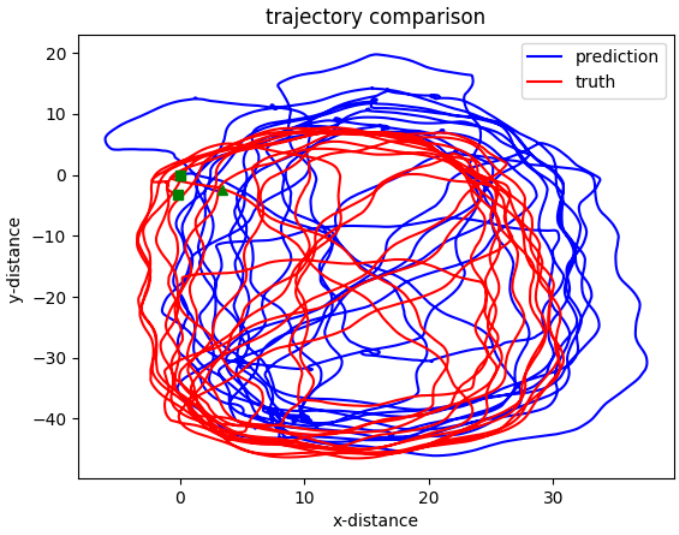

Figure 56: Trajectory for LSTM neural network architecture trained on Oxford PDR data set.

After initial testing showed results with the ability track track the test sample on the Oxford data set the neural network architectures were tested on the urban data set. The models were trained from initialization on the urban data set and tested

on the urban data test set. The two models that performed the best on the Oxford data set showed no success in learning on the urban data set. The predicted outputs averaged around zero and never deviated. Since neither of these models worked, testing began solely on the urban data set again. These new tests included testing the windowing and non windowing inputs, all three output types in Figure 22, LSTM, GRU, and Temporal Convolutional Network (TCN) layer types, and various hyper parameters. Over 500 variations of models were tested using hyper parameter sweeps and training for 25 epochs. This initial test was to get model baselines that could be improved upon to learn basic distance and angle changes. The various hyper parameters tested are shown in table 4. Unlike the localization not every combination was tested due to timing, but a large enough number to be able to eliminate various parameters. By holding certain parameters constant under multiple variations model parameters could be eliminated. Due to the large number of models trained initial performance to eliminate parameters was based on a qualitative visual inspection of the test trajectory prediction performance compared to the true trajectory. The PDR neural networks had similar problems to the Glorot initialized localization neural networks in that they kept learning constant values for all predictions and were clearly evident in the trajectory results. During inspection of the results various hyperparameters were eliminated to include LSTM layers and Bi-directional LSTM layers, sigmoid and ReLU activation, adaptive moment estimation (ADAM) and SGD optimizer, and models trained with no dropout. None of the models trained with these hyperparameters achieved predicted results other than a constant average value of the outputs for all inputs received. In addition to these hyperparameter values two of the output types were not able to obtain a valid prediction. These were second and third model outputs shown in Figure 22 in section 3.3.2 that only output a value for the remaining time step of the inputs and the one that summed the outputs for

the entire input time sequence into a single output. With the parameter window narrowed better models could be pushed forward for additional training.

Table 4: PDR hyper-parameter variations tested

layer types	LSTM Bi-directional LSTM	GRU Bi-direction GRU		
Activation Functions	ReLU	tanh	sigmoid	
Optimizer	RMSProp	ADAM	SGD	
number of stacked layers	2	3	4	
lookback	50	100	150	200
units	32	64	128	256
dropout	0%	30%	40%	50%

The TCN models had various parameters that were taken into account as well that are listed in table 5. The TCNs networks did not perform better than the higher rated GRU layers. This model had large errors in the distance and angle change predicted measurements. This model was tested on both input types, but was only tested on two of the output types. The two output types tested were the first and second outputs shown in section 3.3.2 Figure 22. The learning rate was similar to the others in that it used a starting value of 10^{-4} with a reduction on plateau. This reduction occurred with a 0.8 multiplier and a patience of ten epochs.

Table 5: PDR TCN hyperparameter variations tested

Activation Functions	ReLU	tanh	sigmoid	
Optimizer	RMSProp			
Kernel size	2	3	4	
dilation	[1,2,4,8]	[1,2,4,8,16]	[1,2,4,8,16,32]	
number of stacked blocks	2	3		
dropout	0%	30%	40%	50%

From visual inspection alone the remaining parameters in table 4 could not be further eliminated. The remaining models performance was measured off of the RMSE of the distance, angle change, and position trajectory. Eight models were chosen to to retrain for additional training epochs to try to achieve a lower error rate. These sixteen models were trained for 100 epochs and the best epoch weights were chosen to compare the predicted values from the original. The epoch for testing was chosen based off the lowest validation loss of the neural network during training. Additional parameters common to all of these models is their RMSProp optimizer, hyperbolic tangent activation function, learning rate starting at 10^{-4}, 256 batches, 100 epochs, and the data was windowed. These sixteen models were then compared based off of their RMSE values and displayed in table 6.

Table 6: PDR hyperparameter results. All models trained with Three RNN layers, two time-distributed dense layers, RMSProp optimization, tanh activation, learning rate10^{-4},256 batches,100 epochs, and the data was windowed

Layer type	Input	lookback	units	RMSE distance(m)	RMSE Angle(rad)	RMSE trajectory(m)
GRU	IMU	50	64	0.0925	0.0599	180.3
GRU	IMU	50	128	0.0986	0.0569	174.02
GRU	IMU	100	64	0.0790	0.0794	190.39
GRU	IMU	100	128	0.0797	0.0588	198.52
GRU	IMU and Magnitometer	50	64	0.098	0.0564	171.2
GRU	IMU and Magnitometer	50	128	0.0878	0.0643	223.6
GRU	IMU and Magnitometer	100	64	0.09793	0.0600	195.9
GRU	IMU and Magnitometer	100	128	0.0862	0.0583	188.9
Bi-Directional GRU	IMU	50	64	0.079	0.045	183.06
Bi-Directional GRU	IMU	50	128	0.0883	0.0527	199.6
Bi-Directional GRU	IMU	100	64	0.0847	0.0588	215.6
Bi-Directional GRU	IMU	100	128	0.0923	0.0455	197.5
Bi-Directional GRU	IMU and Magnitometer	50	64	0.0780	0.0499	197.2
Bi-Directional GRU	IMU and Magnitometer	50	128	0.0838	0.0581	205.6
Bi-Directional GRU	IMU and Magnitometer	100	64	0.095	0.0412	178.12
Bi-Directional GRU	IMU and Magnitometer	100	128	0.0741	0.0479	195.68

The model that performed the best under the various tests was a three layer GRU network shown in Figure 57. This model had all three input sensors including the accelerometer, gyroscope, and magnetometer data. The output training target was when an output was predicted for each time step of the input sequence showcased in section 3.3.2 Figure 22. This model had a data input size of fifty time steps of data and slides once every time step. Only the last output is taken from each output input pairing as it slides through the data once the first fifty predictions occured. Moreover, the model had sixty-four units for each layer and trained on batches of 256. This model utilized a hyperbolic tangent activation for all layers except a linear final activation. The model was trained with a RMSProp optimizer with a learning

rate starting at $10^{-}4$ with a 10^{-1} reduction whenever the data would plateau for twenty epochs.

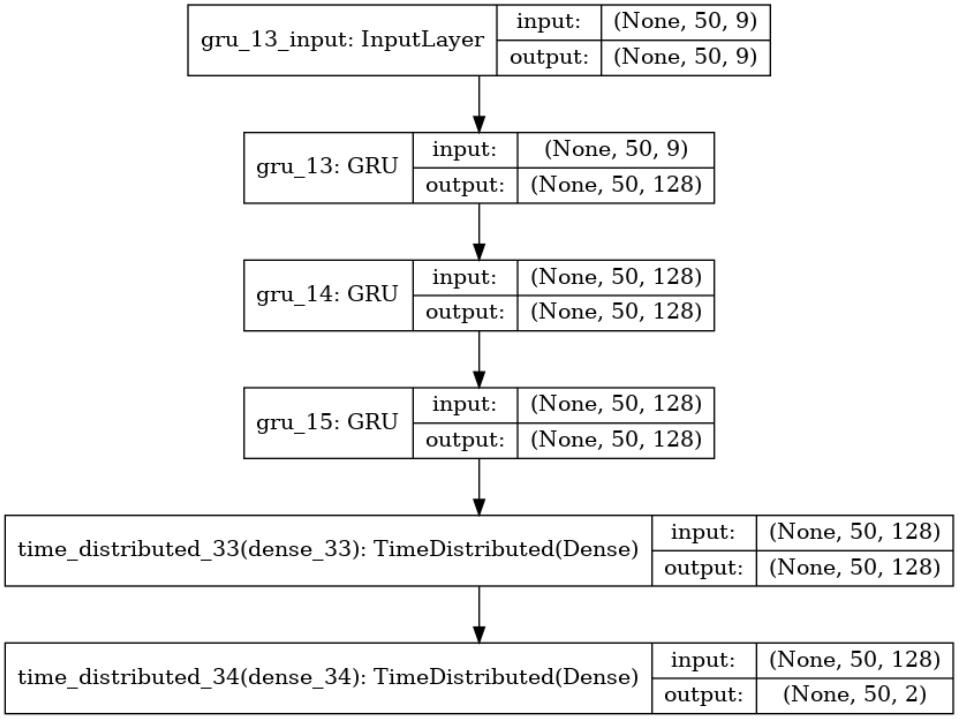

Figure 57: velocity GRU neural network architecture obtaining the lowest MSE for distance and angle measurement

The results for the urban data set were not able to match the results shown in Chen et-all [43] or reproduced on their data set. This could potentially be due to the type of data being analyzed or the difference in labeling the data. His work had much higher resolution for tracking positional changes per frame. This allows the network to get precise changes every step of the input instead of interpolated answers which provide similar responses for large sequences of the inputs. Another difference is the trajectory continually went in circles in a small room. This is not representative of the urban environment that it was retrained on. The urban environment had large straight paths with minimal turns. Figure 58 showcases the predicted distance compared to the true distance of the test collect. When the windowing effect is taken

into consideration the distance RMSE is 0.0098m and the change in angle RMSE is 0.056rad. Due to the activation function, negative numbers seep into the model for the distance which does not happen in the true model. However, negative numbers happen infrequently and don't provide a large portion of the error. Changes in angle seemed to be a lot harder to determine for the the neural network models on this data set. This is partly due to the sparsity of changes in angles throughout the training data.

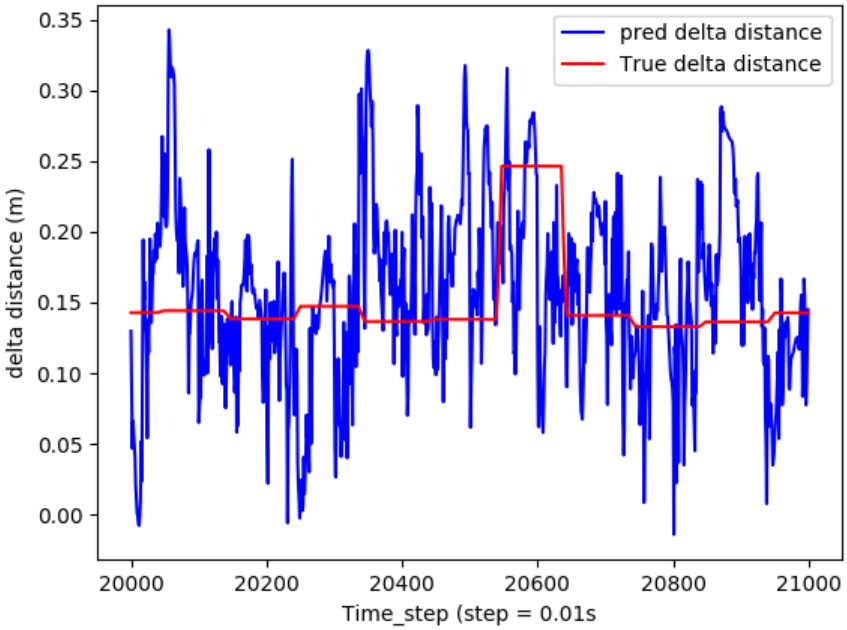

Figure 58: GRU truth vs predicted distance changes for 1000 time steps

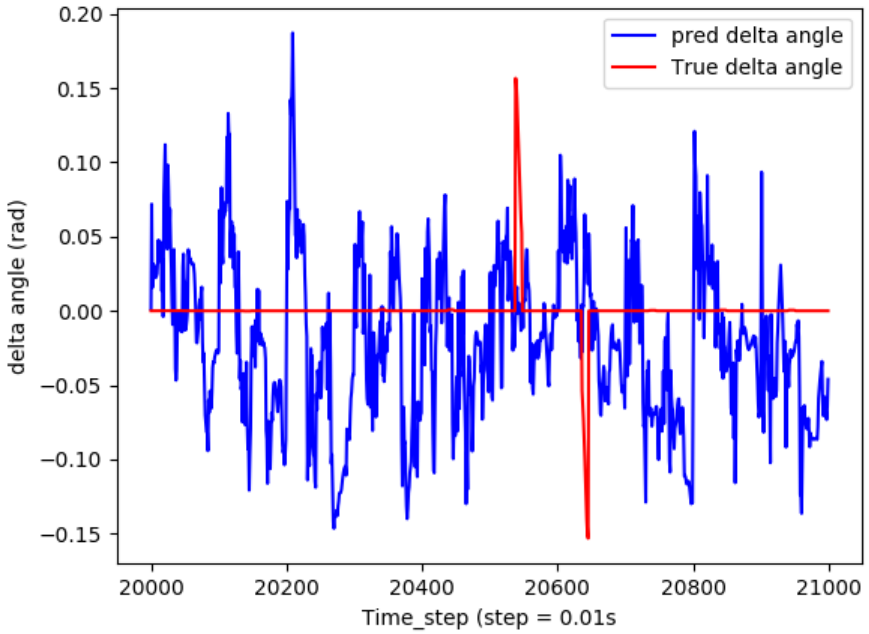

Figure 59: GRU truth vs predicted angle changes for 1000 time steps

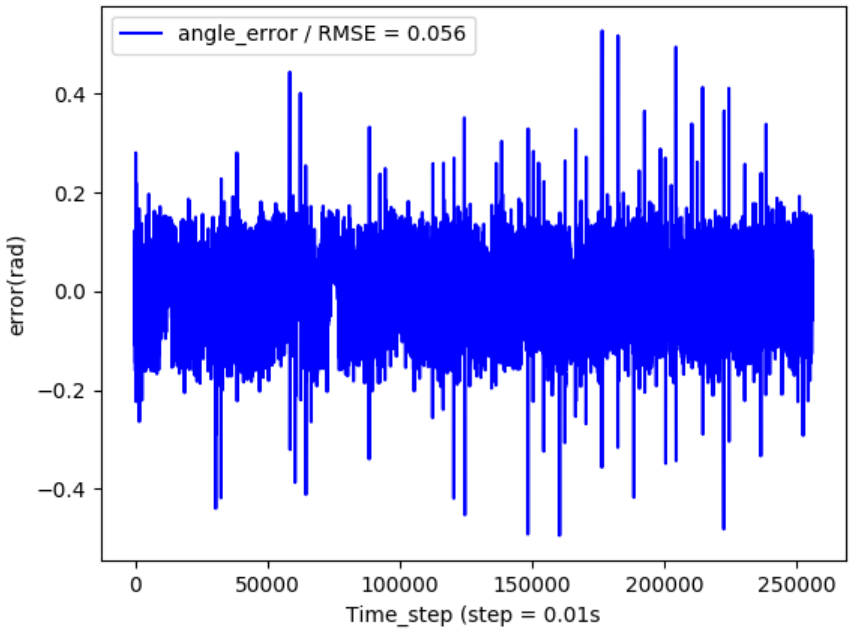

Figure 60: GRU predicted angle change error over the course of the test collection

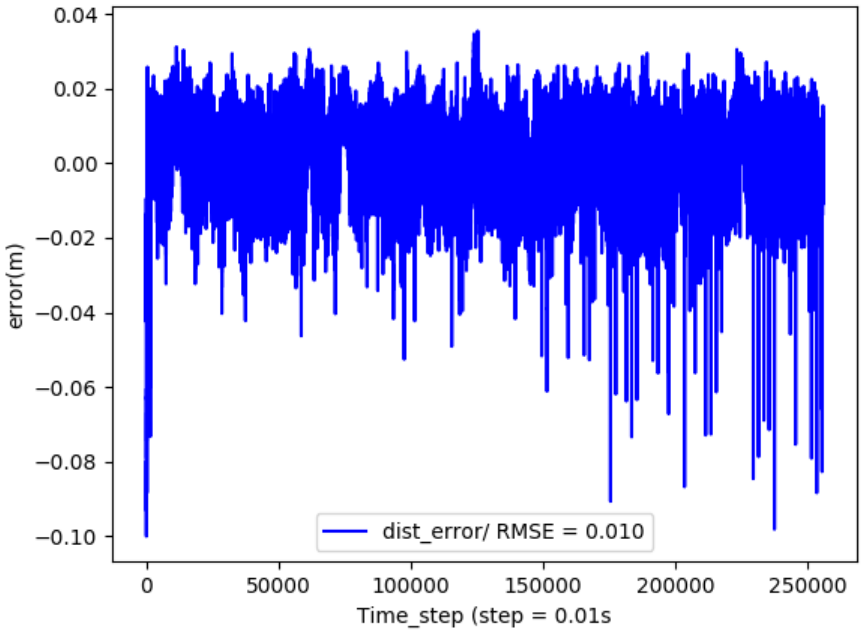

Figure 61: GRU distance error over the course of the test collection

While these errors are small they compound on each other as the model progresses through the 240,000 time steps in the data set. This results in a large deviation from the true data set and an RMSE of 171m and a whole state plot shown in Figure 62. This whole state plot was generated with an initial true angle of orientation and then propagated completely on change in distances and angles to that original orientation.

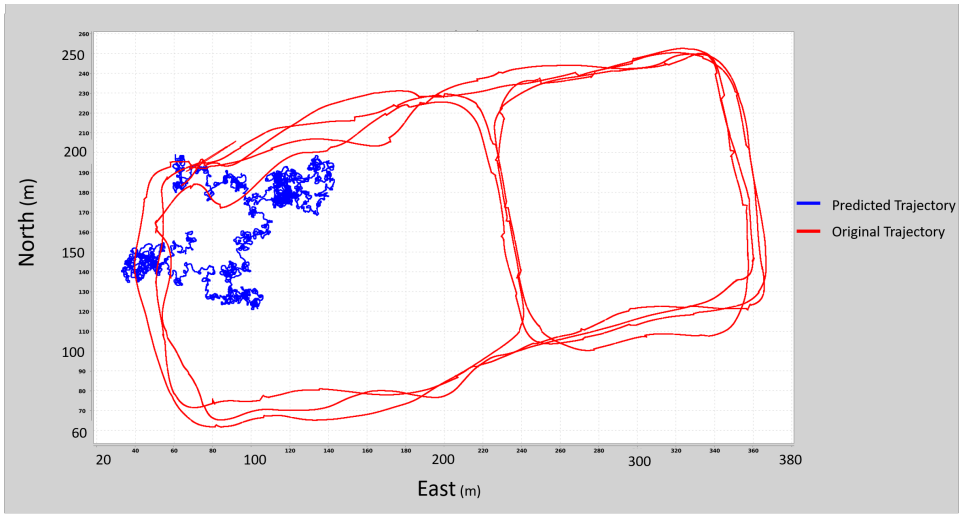

Figure 62: PDR whole state solution for highest performing GRU based model.

4.4 Extended Kalman Filter

Neither of the two neural network results were able to create a complete solution on their own that tracked the trajectory within a small enough margin of error. The localization had very noisy and jumpy result and the PDR solution wasn't able to follow the trajectory at all. To determine if classical filtering could improve either solution the ANNs were combined together within an EKF to potentially reduce errors and smooth results. As mentioned the EKF utilized four state state-space representation with a first order Gaussian Markov (FOGM) dynamics block to monitor the north and east velocities. The FOGM variance and time constants were approximated and tuned to work well based off of the true data set. The dynamics Q noise matrix is showcased in equation 32 where the sigma values are set to 0.1 and time constant(τ)

is set to one second.

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 e^{\frac{-|t|}{\tau}} & 0 \\ 0 & 0 & 0 & \sigma^2 e^{\frac{-|t|}{\tau}} \end{bmatrix} \tag{32}$$

The measurement noise model was a four by four cross correlation matrix with variances based off of results from the single test set. It was assumed there would be no correlation between any of the states and everything except the diagonal was set to zero. The position variances were determined by taking the average error variance across the entire test collect for the VGG16 ANN highest performing results. The PDR variances were estimated based off of the error variance and tuned. A direct relationship between the PDR ANN and the EKF can't be made because the EKF is calculated in absolute velocity and the ANN is relative distance and angle changes per time step. The localization variance and tuned velocities for the measurement noise matrix R is highlighted in equation 33. This matrix showcases the noise for north position, east position, north velocity, and east velocity providing the highest accuracy results.

$$\mathbf{R} = \begin{bmatrix} 425 & 0 & 0 & 00 \\ 0 & 806 & 0 & 0 \\ 0 & 0 & .01 & 0 \\ 0 & 0 & 0 & .01 \end{bmatrix} \tag{33}$$

Two EKF solutions are shown in Figures 63 and 64. Figure 63 showcases the EKF results with a shorter period between position updates. Position is updated every three seconds and velocity updates are given every one hundredth of a second.

This solution was able to obtain an RMSE value of $28.5m$ from the true position for the complete trajectory. Figure 64 showcases an EKF result that has a larger time between position updates. This model updates position every thirty seconds with velocity updates still occurring every one-hundredth of a second. This solution was able to obtain an RMSE value of $48.2m$. Both of these solutions have a center bias that has propagated into them from the localization neural network.

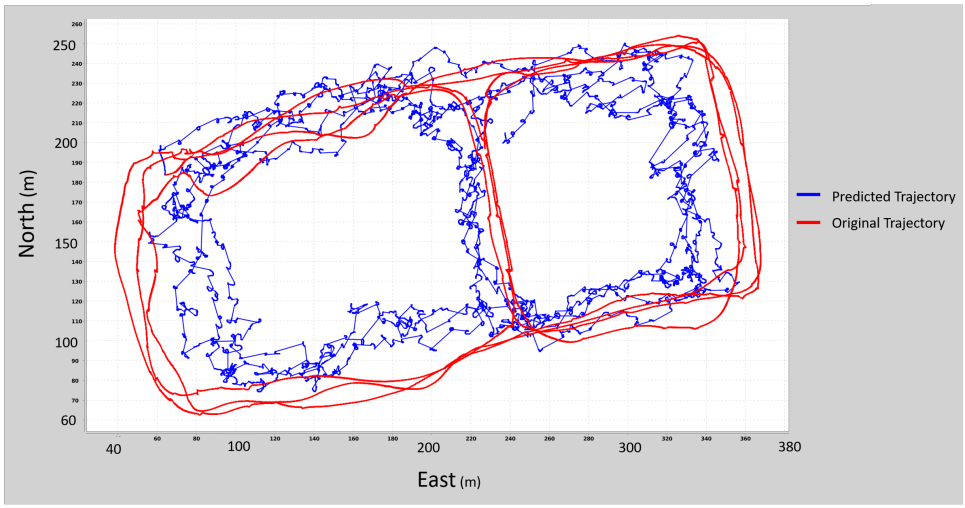

Figure 63: EKF solution with position updates every 3 seconds velocity updates one hundredth of a second

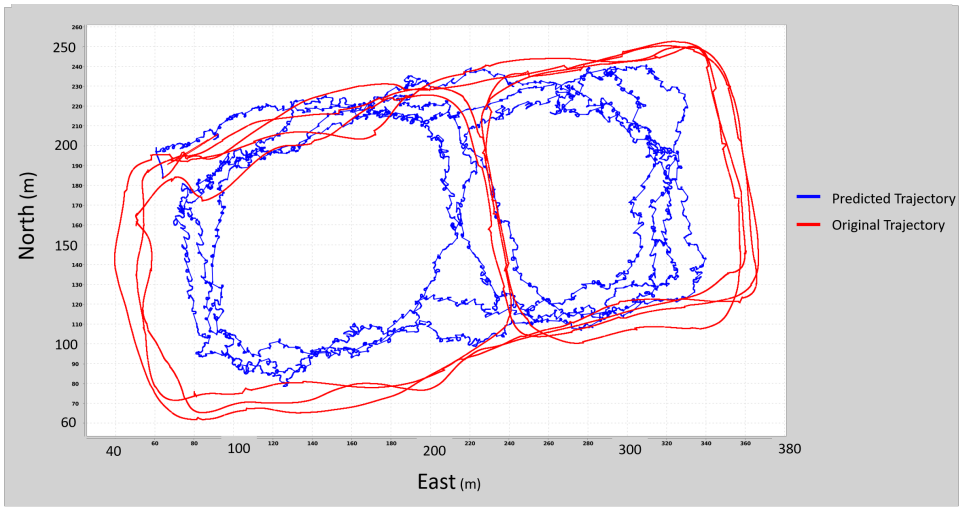

Figure 64: EKF solution with position updates every 30 seconds velocity updates one hundredth of a second

The localization solution appeared to introduced less error into the system as the results got remarkably better for increasing the position measurement updates. This effect only lasted up until the three second update mark and then the error started to grow again as it became more frequent. This appears to be due to the noisy nature of the data and increasing the update rate begins to override the beneficial aspects of the velocity updates when faster than three seconds. Since the results from the PDR solution are less than desirable tests were run without any velocity updates to the system to ensure the EKF was getting some useful information from the PDR solution. This resulted in RMSE values similar to the localization results with a complete RMSE value of 34.65m on the best performing VGG16 model. While this is an improvement it is much smaller than the improvements made by having the PDR solution combined.

4.5 Chapter Summary

The solutions provided by the localization ANN were able to obtain a navigation solution with and RMSE of 28m. Although this is not groundbreaking results it does showcase the increased benefit of employing transfer learning practices on navigation solutions. Additionally, there isn't a guaranteed ANN model that will perform best for all data sets. The need still exists to test out various models on new data sets. The PDR solution was not able to obtain an accurate solution based on the given data set. The type of data seems seems extremely important when solving these solutions as previous work on different data sets was able to obtain much higher accuracy results. Finally, from the RMSE results and figures 63 and 64 the EKF solution had better performance than either ANN solutions on their own. Further closing thoughts are discussed in the next chapter.

V. Conclusions

5.1 Conclusion

As the increase in urbanization and Global Positioning System (GPS) degraded and denied areas continues to grow solutions are needed to quickly and reliably navigate areas for pedestrians. These pedestrian navigation methods require every day technologies with quick and efficient algorithms. Cellular phones are an integral technology used by most people will continue to provide service for at least the near future. Additionally, Artificial Neural Network (ANN) algorithms are continually improving and have the potential to continue to provide better results in both image and sequential based methods. This research's purpose was to explore the viability of ANN based pedestrian navigation for localization and Pedestrian Dead Reckoning (PDR). The ANN utilized a set of 30 collects with over 1.5 million images and 6 million data points for the accelerometer, gyroscope, and magnetometer. The results from this training were also combined with classical filters to determine the combined results.

An ANN was used to approximate a localization function based on images. A variety of Methods were explored using both transfer based learning as well as traditional learning. The method using transfer based learning with pre-trained weights on the ImageNet database proved to be the most successful. All models trained using this method were able to obtain a navigation solution with various results. Of these results the model network called VGG16 obtained the lowest root mean square error (RMSE). This model had half it's weights retrained on the urban data set and was able to obtain an RMSE of 35m. Additional models were trained without the use of transfer learning including variations that employed residual, widenet, and basic sequential type convolutional neural network (CNN) models. None of these models

were able to obtain a navigation solution.

Another ANN method was explored to approximate a PDR function that used sequential type accelerometer, gyroscope, and magnetometer based information to obtain a change in distance and heading angle. A variety of models were tested with with varying inputs, outputs, and neural network architectures. Models that obtained the lowest RMSE for these two values used all input information available including the accelerometer, gyroscop, and magnitometer. Data outputs were created for every input step. The model architecture were able to obtain closer results when using hyperbolic tangent functions as an activation function, RMSProp optimizer, and Gated Recurrent Unit (GRU) based neural network layers. The best model selected employed a three layer GRU layer and was able to obtain a 171m position accuracy over the test collect.

Methods using the classical extended Kalman Filter (EKF) were able to obtain the best results by combining the solutions to both the localization and PDR ANN. This method used the localization as a measurement update step and a the PDR as a state prediction update based off the models predicted orientation. Multiple models were tested with various measurement update rates and covariance and noise matrices. Of these the model with localation based updates of three seconds and velocity updates every one-hundredth of a second obtained an RMSE of 28.5m. These errors show combining ANN with classical filters provides a complete navigation solution with minimized errors.

5.2 Future Work

This research demonstrated that using nerual networks with classical filters can be used to provide improved results over neural networks alone for pedestrian navigation. This research is not comprehensive and additional work could be done to potentially

improve results. These methods include but are not limited to the following: 1. Testing additional methods and obtaining a PDR solution would potentially reduce overall RMSE for both the PDR and EKF solution. One potential method would be to obtain additional data sets with larger number of turns so the ANN has a greater opportunity to learn turning. 2. Include measurement variance per time step into the neural network to try to create a variance model that adjusts to the data being seen. These added variance values would provide increased information for the EKF solution and create a more precise variance of the solution. 3. One emerging method for potentially increasing results for the localization would be to utilize work in Neural Architecture Search(NAS) network models [44]. This uses a neural network to create the best block of information for a given type of data set and currently has the highest accuracy for classifying the ImageNet Database. 4. Finally, the last potential would be to determine application solutions viability in indoor environments as GPS signals are not usable.

Bibliography

1. Francois Chollet. *Deep Learning with Python & Keras*, volume 80. Manning Publications Co., 2018.

2. Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. pages 1–15, 2016.

3. Yuan Zhuang, Jun Yang, You Li, Longning Qi, and Naser El-Sheimy. Smartphone-based indoor localization with bluetooth low energy beacons. *Sensors (Switzerland)*, 16(5):1–20, 2016.

4. Wilfred E. Noel. *Signals Of Opportunity Navigation Using Wi-Fi Signals*. PhD thesis, Air Force Institute of Technology, 2011.

5. J Raquet and R K Martin. Non-GNSS radio frequency navigation. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 5308–5311, mar 2008.

6. Hyunho Lee, Jaehun Kim, Chulki Kim, Minah Seo, Seok Lee, Soojung Hur, and Taikjin Lee. Object recognition for vision-based navigation in indoor environments without using image database. In *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, pages 1–2. IEEE, jun 2014.

7. Mart\'\in~Abadi, Ashish~Agarwal, Paul~Barham, Eugene~Brevdo, Zhifeng~Chen, Craig~Citro, Greg~S.~Corrado, Andy~Davis, Jeffrey~Dean, Matthieu~Devin, Sanjay~Ghemawat, Ian~Goodfellow, Andrew~Harp, Geoffrey~Irving, Michael~Isard, Yangqing Jia, Rafal~Jozefowicz, Lukasz~Kaiser, Manjunath~Kudlur, Josh~Levenberg, Dan~Mané, Rajat~Monga, Sherry~Moore,

Derek˜Murray, Chris˜Olah, Mike˜Schuster, Jonathon˜Shlens, Benoit˜Steiner, Ilya˜Sutskever, Kunal˜Talwar, Paul˜Tucker, Vincent˜Vanhoucke, Vijay˜Vasudevan, Fernanda˜Viégas, Oriol˜Vinyals, Pete˜Warden, Martin˜Wattenberg, Martin˜Wicke, Yuan˜Yu, and Xiaoqiang˜Zheng. {TensorFlow}: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

8. François Chollet and Others. Keras. \url{https://keras.io}, 2015.

9. U.S.A Department Of Defense. Global Positioning System Standard Positioning Service. *Www.Gps.Gov*, (September):1 – 160, 2008.

10. B Hofmann-Wellenhof, H Lichtenegger, and J Collins. *GPS - Global Positioning System. Theory and practice.* 1997.

11. Peter Maybeck. *Stochastic Models, Estimation and Control Volume 1.* Academic Press, New York, NY, 1979.

12. Robert Grover Brown and Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering, Third.* John Wiley and Sons, 3rd edition, 1996.

13. David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004.

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

15. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.

16. Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

17. Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

18. Lon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2016.

19. Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Overview of mini-batch gradient descent. `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

20. Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. dec 2014.

21. Peter Bhlmann and Torsten Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, 22(4):477505, Nov 2007.

22. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

23. Nitish Srivastava Salakhutdinov, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

24. yarin Gal and Zoubin ghahramani. A Theoretically Grounded Appliccation of Dropout in Recurrent Neural Networks. 2015.

25. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, 9:249–256, 2010.

26. Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A Survey on Deep Transfer Learning. aug 2018.

27. Jurgen Hochreiter, Sepp and Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

28. Bart Van Merri. Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation. 2014.

29. J Deng, W Dong, R Socher, L.-J. Li, K Li, and L Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

30. Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, sep 2014.

31. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. dec 2015.

32. Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *British Machine Vision Conference 2016, BMVC 2016*, 2016-Septe:87.1–87.12, 2016.

33. Chollet Francois. Xception: Deep Learning with Depthwise Separable Convolutions. 2016.

34. Christian Szegedy, Vincent Vanhoucke, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015.

35. Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

36. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, jun 2008.

37. Martin a Fischler and Robert C Bolles. Paradigm for Model. *Communications of the ACM*, 24(6):381–395, 1981.

38. Michael B. del Rosario, Stephen J. Redmond, and Nigel H. Lovell. Tracking the evolution of smartphone sensing for monitoring human movement. *Sensors (Switzerland)*, 15(8):18901–18933, 2015.

39. Alex Kendall, Matthew Koichi Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2938–2946, 2015.

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

41. Joseph Curro. Navigation with artificial neural networks. *Theses and Dissertations)*, 2018.

42. Changhao Chen, Peijun Zhao, Chris Xiaoxuan Lu, Wei Wang, Andrew Markham, and Niki Trigoni. OxIOD: The Dataset for Deep Inertial Odometry. 2018.

43. Changhao Chen, Xiaoxuan Lu, Andrew Markham, and Niki Trigoni. IoNet: Learning to cure the curse of drift in inertial odometry. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 6468–6476, 2018.

44. Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. *Proceedings of the*

IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 8697–8710, 2018.

Acronyms

1D one dimension. 15, 27

2D two dimension. 15, 21, 22, 45

ADAM adaptive moment estimation. 19, 89

ANN Artificial Neural Network. iv, ix, 1, 2, 3, 4, 5, 14, 15, 17, 19, 20, 21, 33, 36, 37, 38, 40, 42, 44, 47, 49, 51, 52, 53, 59, 60, 85, 86, 98, 99, 101, 103, 104

AWGN additive Gaussian white noise. 10, 11

BLE Bluetooth Low Energy. 1, 33

CNN convolutional neural network. 21, 22, 24, 36, 45, 49, 50, 103

DCM direct cosine matrices. 8, 9

ECEF earth centered earth fixed. viii, 5, 7, 8

EKF extended Kalman Filter. iv, ix, xii, 3, 4, 38, 49, 50, 51, 52, 53, 86, 98, 99, 100, 101, 104

FOGM first order Gaussian Markov. 50, 51, 98

GPS Global Positioning System. iv, ix, x, 1, 5, 40, 41, 42, 53, 54, 55, 56, 103, 104

GRU Gated Recurrent Unit. xi, xii, 24, 26, 47, 86, 87, 89, 93, 94, 95, 96, 97, 104

IMU inertial measurement unit. 2, 33, 37, 39, 40, 42, 44, 58

LSTM Long Short-Term Memory. xi, 24, 26, 37, 47, 86, 88, 89

MEMS Micro-Electro-Mechanical Systems. 39

MIMU magnetic and inertial measuring unit. 35

MSE mean squared error. xi, xii, 17, 18, 44, 69, 71, 76, 78, 79, 94

NED north, east, down. viii, 7, 8, 9, 44, 68

ORB Oriented FAST and rotated BRIEF. 33

PDR Pedestrian Dead Reckoning. iv, ix, xi, xii, xiii, 3, 33, 35, 37, 38, 41, 42, 43, 44, 47, 48, 49, 50, 52, 53, 86, 87, 88, 89, 91, 92, 97, 98, 99, 101, 103, 104

PPS Precise Positioning Service. 5

RANSAC random sample consensus. 33

ReLU rectified linear unit. viii, x, 15, 16, 32, 46, 60, 62, 64, 76, 89

ResNet residual network. ix, 30, 31, 45

RF radio frequency. 1

RGB red, green, and blue. 13, 40

RMSE root mean square error. iv, 53, 69, 72, 80, 84, 85, 86, 92, 94, 97, 99, 101, 103, 104

RMSProp root mean squared propagation. x, 19, 60, 62, 64, 76, 86, 93

RNN Recurrent Neural Network. viii, 24, 25, 26, 47, 49

SGD stochastic gradient decent. 19, 76, 89

SIFT scale invariant feature transform. 13, 33

SoOP signals of opportunities. 5, 33

SPS Standard Positioning Service. 5

SURF speeded up robust features. 33

tanh hyperbolic tangent function. 15, 86

TCN Temporal Convolutional Network. viii, xiii, 27, 28, 47, 89, 91

WIFI Wireless Fidelity. 1, 33

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Sept 2018 — Mar 2020

4. TITLE AND SUBTITLE

Pedestrian Navigation using Artificial Neural Networks and Classical Filtering Techniques

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

David J. Ellis

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-20-M-018

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISA - Information Management Technologies Branch
26 Electronics Parkway Rome, NY 13441-4514
COMM 315-313-5302
Email: james.metzler@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RISA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The objective of this thesis is to explore the improvements achieved through using classical filtering methods with Artificial Neural Network (ANN) for pedestrian navigation techniques. A novel urban data set is created for testing various localization and Pedestrian Dead Reckoning (PDR) based pedestrian navigation solutions. Cell phone data is collected including images, accelerometer, gyroscope, and magnetometer data to train the ANN. The ANN methods are explored first trying to achieve a low Root Mean Squared Error (RMSE) of localization and PDR solutions. After analyzing the localization and PDR solutions they are combined into an Extended Kalman Filter to achieve a 20% reduction in the RMSE.

15. SUBJECT TERMS

Artificial Intelligence, Pedestrian Navigation, Localization, Pedestrian Dead Reckoning, Kalman Filtering

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Captain Joseph A. Curro, AFIT/ENG
U	U	U	UU	115	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext:4620 joseph.curro@afit.edu