UNIVERSITY OF OKLAHOMA
GRADUATE COLLEGE

AUTOMATED DETECTION OF BIRD ROOSTS USING NEXRAD RADAR DATA AND CONVOLUTIONAL NEURAL NETWORKS

A THESIS
SUBMITTED TO THE GRADUATE FACULTY
in partial fulfillment of the requirements for the
Degree of
MASTER OF SCIENCE

By

CARMEN CHILSON
Norman, Oklahoma
2017

AUTOMATED DETECTION OF BIRD ROOSTS USING NEXRAD RADAR DATA AND CONVOLUTIONAL NEURAL NETWORKS

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Amy McGovern, Chair

Dr. Eli Bridge

Dr. Andrew Fagg

Acknowledgements

I would like to thank Dr. Amy McGovern for guiding me throughout my research, helping me find an interesting dataset to work on, and being my advisor and committee chair. I would also like to thank Dr. Jeff Kelly and the Oklahoma Biological Survey for sharing their data with me. The data took a long time to collect and organize and I appreciate them sharing their research with me. I also wanted to thank the other professors at OU who helped with with my research project. Dr. Eli Bridge was very helpful in answering all my biological questions and Dr. Phillip Chilson answered all of my radar related questions. I also want to say thank you to my whole committee for your time and help.

Kate Avery, an undergraduate at the University of Oklahoma, decided to take an interest in my research. She started working with me early on in my research and helped organize my data. She also build her own machine learning model that I use as a comparison against my results. She showed a lot of initiative and was very helpful even as a Freshman. Thank you for your for time and help.

Table of Contents

List Of Tables

List Of Figures

Abstract

NEXRAD radars have proven to be an effective tool for detecting bird roosts for several species or birds, however manually locating these roosts in radar images is a time consuming process. We introduce a Convolutional Neural Network trained to automatically determine whether each individual radar image contains at least one Purple Martin or Tree Swallow roost. Radars give us a continental-scale snapshot of an entire vertebrate population. Many fields within ecology conservation could benefit from automated detection of bird roosts, and we are able to find bird roosts for species that are visible in radar imagery with 90 percent accuracy. We use a dataset of radar images that contain Purple Martin roosts and Tree Swallow roosts in the Eastern half of the United States. We show that Convolutional Neural Networks (CNNs) are an effective method for automating the bird roost detection. CNNs have recently revolutionized image classification largely because CNNs capture spatial components of images. We hypothesized that these same principles can be applied to radar data. To further improve the accuracy of bird roost detection, machine learning techniques such as batch normalization and transfer learning are applied to the CNN. Our results show that CNNs are a promising approach for bird roost detection for legacy radar data and dual polarization radar data.

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are the most state-of-the-art methods for computer vision (Szegedy et al. 2016). In 2012, Deep CNNs achieved record breaking results for classifying ImageNet data (Krizhevsky et al. 2012). Since then, many strides have been made in image processing using varying architectures for CNNs (Srivastava et al. 2014; Szegedy et al. 2015; Ioffe and Szegedy 2015). We use Convolutional Neural Networks to automate bird roost detection in level 2 NEXRAD radar data formatted as 2D images.

Although CNNs have achieved amazing results, they require large amounts of training data to be effective since the network has to learn millions of weights (Oquab et al. 2014). Since our dataset contains approximately 30,000 labeled images, we turn to transfer learning. Transfer learning allows us to transfer knowledge learned on one dataset to a new dataset by tweaking the weights of the model (Oquab et al. 2014). A CNN trained on ImageNet can learn to categorize the 1000+ classes of images, but it will also learn general image features that can be applied to other image data (Shin et al. 2016).

Although deep CNNs have been shown to be quite effective for many computer vision problems, we decided to test how well convolution performs in shallower neural networks since this reduces the number of weights that the network has to learn, and therefore, the required number of labeled data samples. Part of what makes images difficult to classify is the number of background objects, the

size of the object, and the large number of types of objects that the user wishes to classify (Tudor Ionescu et al. 2016). Radar images contain simpler patterns than photographs, and our dataset only has two classification categories. We believe this may make it possible to build a network with fewer convolution layers. Smaller networks can be effective for image processing, for example an automated facial recognition system was built using only 5 convolutional layers (Lawrence et al. 1997).

This thesis evaluates how well machine learning methods such as Artificial Neural Networks and Convolutional Neural Networks can learn to identify bird roosts in NEXRAD radar images using techniques such as transfer learning and batch normalization. We use radar data formatted as 2D images as inputs to the networks and output whether the radar contains a roost. We compare several different network architectures and explain which network architecture works best for this problem. Our approach is the first attempt that we are aware of to automate the roost detection using legacy and dual polarization NEXRAD radar data.

Chapter 2

Background

2.1 Bird Roosts

Studying bird movements is an important aspect of ecological conservation (Shipley et al. 2017). We outline a few examples in this section. Migrating birds transport nutrients, transport organisms, forage, and become prey, all of which impact the local ecosystems. Local crop yields can benefit from migrating birds eating insects (Bauer and Hoye 2014). Approximately 3.5 million birds migrate over wind farms and it's estimated that over 100 million birds collide with man-made obstacles each year in the USA alone (Johnson et al. 2002). There is a long list of reasons a variety of different stakeholders could benefit from studying bird roosts and bird migration: wind turbine collision, habitat deterioration, nature conservation, pest control, crop damage, pollination, dispersal of pathogens, citizen science, and research (Bauer et al. 2017). Many researchers have shown that NEXRAD radars are a useful tool for locating several species of bird roosts and studying bird migration (Bauer et al. 2017; Chilson et al. 2012a; Gauthreaux Jr and Belser 2003; Kelly et al. 2012; RoyChowdhury et al. 2016; Stepanian and Horton 2015).

Aeroecology is a scientific discipline that integrates aspects of atmospheric science, ecology, earth science, computer science, computational biology, and

engineering to further study biology in the atmosphere (Kunz et al. 2008; Chilson et al. 2012b). Currently, most approaches to using radars to study bird roosts are local and small scale (Bauer et al. 2017). One significant factor that currently hinders radar based Aeroecology research is that "accessing and processing the data require significant computational skills and time investment" (Chilson et al. 2012a). A few species of bird roosts can be spotted in radar data since they often form distinct patterns within the radar know as "roost rings" (Kelly and Pletschet 2017). Although these pattens are easy for the human eye to detect, only a few people within the biology community utilize this data (Chilson et al. 2012a). Even those who process the radar data into images are still faced with the time consuming task of sifting through the data to find the bird roosts. Machine learning can assist in detecting bird roosts in radar data by providing automated models trained on data that biology researchers have previously hand labeled.

Several research teams have developed techniques for automatically detecting birds at a close range using specialized radar. One such approach uses a radar adapted specifically for bird detection to identify single small and medium sized birds flying across a fixed position radar beam (Zaugg et al. 2008). The radar continuously monitors a bird over several seconds and the patterns produced by wing flapping allow the bird to be detected within 8 km of the radar (Zaugg et al. 2008). Another approach uses Airport Surveillance Radar (ASR-9) to automatically detect small groups of birds within 10 km of the airport by completing a volume scan every 5 seconds (Troxel et al. 2001). The high scan rate of the ASR-9 allowed them to detect moving flocks of birds and prevent bird strikes on airplanes (Troxel et al. 2001). Another proposed solution is the Avian Radar Sensor Design (Weber et al. 2005). The WSR-88D radar could be

configured to meet short-range avian radar requirements, however this modification would modify the radar design specifically for bird-strike advisory rather than for weather (Weber et al. 2005). These papers clearly show that birds can automatically be detected within 10 km with specialized radars. NEXRAD radars, however, can detect clusters of "biological targets" up to 240 km away (DeVault et al. 2013, p. 142). Using NEXRAD radars to detect several species of birds roosts allows us to use the existing radar infrastructure without the cost of installing and maintaining more radars. This approach allows us to find bird roosts in archived NEXRAD data as far back as 1991 and from 2013 for upgraded dual-polarization radar data. We hypothesize that our method for locating bird roosts in dual polarization will be more effective, however it is worth developing a method for legacy radar data because it enables us to examine years of past data.

2.2 NEXRAD Radar

The radar data used for this research came from the level 2 data from the nationwide network of NEXRAD radars. This data set consists of a mix of single-polarization Doppler radar (which we refer to as legacy radar in this thesis) and dual-polarization Doppler radar. The radars were incrementally upgraded from 2012 to 2013, and with this process, came benefits for biological applications as well (Stepanian et al. 2016). Different radar products are created from the horizontal and vertical radar polarizations. Reflectivity, Doppler Radial Velocity, and Spectrum Width radar fields are available to us in the legacy and dual-polarimetric radar data, however differential Reflectivity (Z_{DR}), Differential Phase (ϕ_{DP}), and Correlation Coefficient (ρ_{HV}) are only available in the

dual-pol radars. We go into more detail about each radar product and which are useful for detecting bird roosts in Chapter 3.

The NEXRAD radars scan the lower 10 km of the atmosphere and have been collecting data approximately every 5 minutes since the early 1990s (Chilson et al. 2012a). Using radar, the peak roost visibility starts 20 min before sunrise and ends 40 minutes after sunrise. The majority of bird roosts can be found using this 60 minute window (approximately 12 radar scans). Furthermore, the roosts have primarily been detected over the summer months and and the roost locations have been confirmed in 64 different Eastern US (east of 100° W) radars (Kelly and Pletschet 2017). Although limiting the time of day, year, and the number of radars that need to be searched helps reduce the size of the dataset, that still leaves approximately 70,000 radar images a year to look through. Kelly and Pletschet start searching for roosts one hour before local sunrise until 30 minutes after local sunrise from June 1 to September 30 (2017). This is a broader and more thorough search of the roost data and requires researchers to search through 140,000 images a year. Both search windows still require researchers to manually look through large amounts of data. Our work will reduce the number of radar images researchers need to search through by automatically searching through the 140,000 images and finding images that likely contain roosts.

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) consist of multiple layers of connected artificial "neurons". They are inspired by biological neural networks (Mitchell 1997).

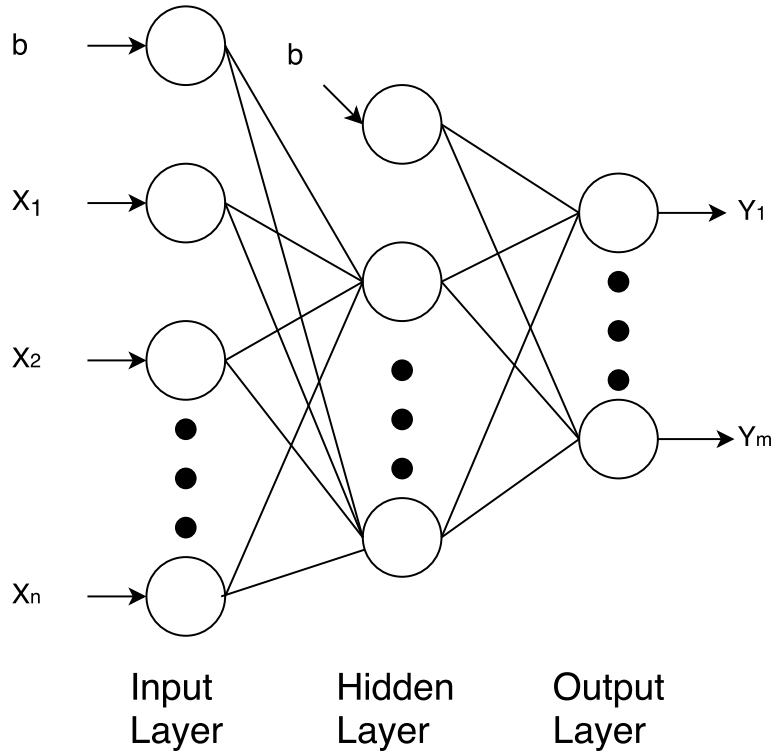


Figure 2.1: An example of a typical feed-forward neural network with 1 hidden layer. $x_1, x_2, ..., x_n$ represent the network input and $y_1, ..., y_m$ represent the network outputs.

ANNs are mathematical models that can represent complex nonlinear relationships between inputs and outputs (Dayhoff and DeLeo 2001). The architecture of a typical fully connected ANN containing one input, hidden, and output layer is depicted in Figure 2.1. In this section we will start by explaining how the inputs are passed forward through the network and modified to produce the outputs values.

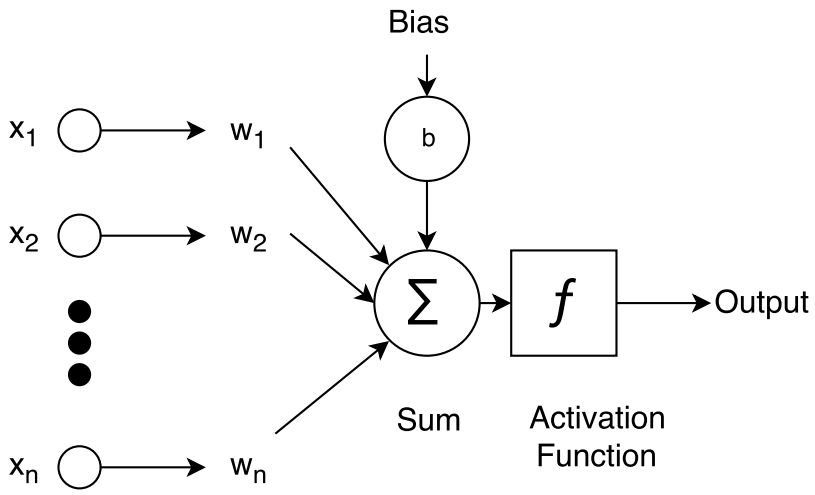


Figure 2.2: The structure of an artificial neural network neuron. The inputs $x_1, x_2, ..., x_n$ are multiplied with weights $w_1, w_2, ..., w_n$ and then summed with the bias node. This value is then passed through an activation function to produce the neuron output.

ANNs are built by connecting nodes called neurons. See Figure 2.2 for a visual representation of a neuron. A neuron takes in inputs, x, and processes them into an output value, z:

$$z = b + \sum_{i=1}^{n} w_i x_i. \tag{2.1}$$

In a neuron the inputs $x_1, x_2, ..., x_n$ are multiplied with weights $w_1, w_2, ..., w_n$ and then summed with the bias node b (Raudys 1998) to produce z. The neuron then passes z through an activation function, which is then output from the neuron.

2.3.1 Activation Functions

There are several different commonly used activation functions that neural network neurons use. The sigmoid function (Equation 2.2) is probably the most

common activation function used in classical feed-forward artificial neural networks and produces an output between 0 and 1 (Jain et al. 1996). Here is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.2}$$

Training time using sigmoid nodes is much slower than training time using Rectified Linear Units (ReLUs) (Krizhevsky et al. 2012). ReLUs still provide a nonlinear activation function (Equation 2.3), however it is much faster for gradient descent to optimize because the gradient is constant (Krizhevsky et al. 2012). The ReLU function is define as:

$$\sigma(z) = max(z, 0). \tag{2.3}$$

ReLUs were used in the deep convolutional neural network that achieved record breaking results on classifying the ImageNet dataset (Krizhevsky et al. 2012). ImageNet is a giant image dataset containing 3.2 million annotated images spreading over 5247 different categories (Deng et al. 2009). It is useful to have activation functions that allow for faster training when training computationally expensive networks such as deep convolutional neural networks.

For classification problems, the last layer of neural network outputs labels for the input data. The labels are commonly encoded using one-hot vectors of size n, where n is the number of classification categories. The one-hot vector contains all zero values except for the index of the corresponding label that is marked with a one. The neural network generally has an output node for each classification category. A binary classification model would have two output nodes and a dataset with 50 different categories would need 50 outputs nodes. The advantage of having an output node for each category is that each output node can be interpreted as a probability that the input belongs to any given

category. In order to ensure that all of the probabilities in the output layer add up to one we use the softmax activation function. Softmax is different than sigmoid because it takes all of the outputs of the layer into consideration and normalizes them to sum to one:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{m} e^{z_k}}. \tag{2.4}$$

Equation 2.4 is the softmax equation for node j of a network that has m output nodes.

2.3.2 Backpropagation Through Multiple Layers

In the previous section we explained how the network input gets passed forward through the network and output in the final layer. In this section we explain how the neural network learns to update the weight to minimize the loss of the output using stochastic gradient decent and back-propagation. Back-propagation for neural network neurons was first introduced in 1986 (Rumelhart et al. 1986; LeCun et al. 1989; Werbos 1990). Backpropagation, short for backward propagation of errors, is an algorithm for training artificial and convolutional neural networks using gradient descent to minimize the loss function or the error of the network.

Before discussing backpropagation, we need to discuss the difference between training on an epoch, a single example, or a mini-batch of data. In batch gradient descent training, the weight changes are calculated for every sample in the dataset (an epoch) before being applied to the network (Wilson and Martinez 2003). Traditional stochastic gradient descent processes only one example per iteration before updating the weights (Li et al. 2014). Another approach calculates the gradient on a mini-batch or subset of the training data

before applying a weight update to the neural network (Li et al. 2014). Both traditional stochastic gradient descent and mini-batch gradient decent are considered on-line learning techniques (Wilson and Martinez 2003). While it is true that batch training calculates the true gradient for the weight updates rather than the approximation generated by the instance or mini-batch training, it almost always learns slower in practice, especially for large datasets (Wilson and Martinez 2003). Even though on-line training can have noisy gradients that contradict each other, on average they will move in the direction of the true gradient (Wilson and Martinez 2003). Stochastic gradient descent trains faster, batch gradient descent is more stable, and mini-batch gradient descent allows us to find a batch size that maintains a balance between them.

We will give some background for the stochastic gradient decent method. Note that the x and y used here are not related to the neural network but refer only to a generic function with an input x and an output y. Gradient based optimization tries to minimize the function $f(x)$ by altering x (Goodfellow et al. 2016). A derivative is useful for minimizing y because it tell us how to modify x in order to make a small adjustment in y (Goodfellow et al. 2016). Gradient descent is the process of reducing $f(x)$ by moving in small steps with the opposite sign of the derivative. A neural network has multiple inputs, therefor a partial derivative is computed for each individual weight with respect to loss (Goodfellow et al. 2016).

Supervised neural networks are trained on n samples (called a mini-batch) to minimize the loss or cost of the network. Classification networks with softmax use a cross-entropy cost function:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\big[y_i log(a_i) + (1-y_i) log(1-a_i)\big], \tag{2.5}$$

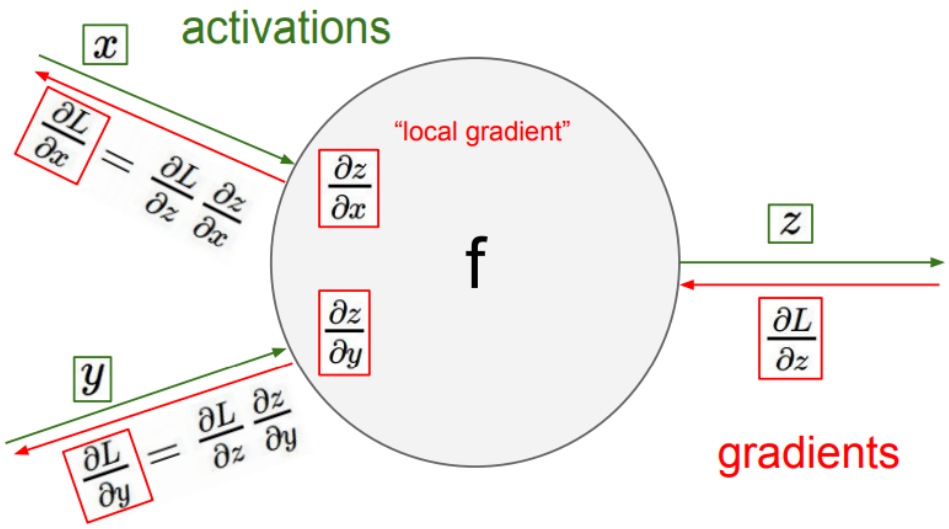


Figure 2.3: Chain rule backpropagation on a single neuron. Image from Li et al. (2016).

where a is the output value of a neuron $\sigma(z)$, and z is the weighted sum of the input values. Loss is computed given a set of inputs $x_1, x_2, ..., x_n$ propagated forward through the neural network with an output of $a_1, ..., a_m$ and a target of $y_1, ..., y_m$.

The purpose of backpropagation is to figure out the partial derivatives of the neural network's loss function with respect to each individual weight of the network. We can use the chain rule to compute the gradient of the error with respect to each weight. Figure 2.3 shows a good visualization of the backpropagation step for a single neuron (Li et al. 2016). This figure is from Stanford lecture slides[1].

[1]http://cs231n.stanford.edu/slides/2016/winter1516_lecture4.pdf

We give an example how to update the weight of a node w_{jk} connecting a node from layer j to layer k with an input vector x and an output y. L_k is the loss of the output node and α is the learning rate.

$$\Delta_k = L_k \times \sigma'(x_k)$$

$$w_{jk} \leftarrow w_{jk} + \alpha \times y_j \times \Delta_k$$

$$\Delta_j = \sigma'(x_k) \sum_k w_{jk} \times \Delta_k$$

There is a variant on stochastic gradient descent called Adaptive Moment Estimation (Adam) (Kingma and Ba 2014). We apply this to one of the networks we describe below. Adam computes adaptive learning rates for the different parameters by estimating the first and second moment of the gradients (Kingma and Ba 2014). Adam uses moments to converge on a solution faster and slowly decays the learning rate (Kingma and Ba 2014). Adam has been shown to produce a faster learning curve for large scale problems in terms of the number of learning iterations required (Kingma and Ba 2014). The full Adam algorithm introduced by Kingma and Ba is shown in Figure 2.4 (2014).

2.4 Convolutional Neural Networks

Convolutional Neural Networks (often referred to as CNNs or ConvNet) are a type of feed-forward artificial neural network that is commonly applied to computer vision problems. Convolutional Neural Networks share many of the properties of Artificial Neural Networks. In this section, we describe CNN properties such as local receptive fields, shared weights, and pooling (Goodfellow et al. 2016).

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2.4: Adaptive Moment Estimation algorithm from Kingma and Ba (2014).

ANNs expect a 1 dimensional vector as the input to the network, but convolutional neural networks can read in 1D, 2D, or 3D data. Since we use CNNs for 2D image classification we will only discuss the 2D case. A 2D convolution layer takes an image with width, height, and channel. Technically, this is 3 dimensional input, but the channel is handled differently so this is still referred to as 2D convolution. Typically, these images will have 3 RBG channels or 1 gray-scale channel.

2.4.1 Convolution Filters

One key component of convolutional neural networks is the use of shared weights. In traditional ANNs, each node in layer n is connected to each node in layer $n+1$ with a corresponding weight. Convolutional neural networks, on the other hand, share weights using filters/kernels. Throughout this paper we will use the words filters and convolution kernel interchangeably. First, we will talk about how a filter is used in the convolutional neural network and then we will describe how the weights of the filter are initialized and then updated during learning.

A convolutional filter has a size of $k \times k \times depth$, where k is a small integer and depth is set by the number of feature maps in the previous layer. The filter slides over the image, spatially convolving the image as seen in Figure 2.5. This figure shows an example of a 3×3 filter being applied to a 7×7 input image with a depth of one. The amount the convolution kernel moves at each step is called the stride length. For the example in Figure 2.5, the convolution filter moves across the matrix with a stride of 1. The region of the input that the kernel is affecting at a single step is called the local receptive field. The local receptive field will always have the same dimension as the convolution filter. The convolution step consists of multiplying the weights of the filter by

the local receptive field. A feature map is computed by repeating this step at every local receptive field. This uses the same idea as the operation visualized in Figure 2.2, only we compute the dot product of the shared filter weights and the local receptive field instead of a vector of inputs and their weights. A feature map is created for each convolution filter applied to a layer as seen in figure 2.6. We apply this process to each convolutional layer of the neural network. In the input layer, we apply a filter size of $k \times k \times channel$, and in the hidden layers we apply a $k \times k \times depth$, where depth is the number of feature maps.

Each filter applied to layer n of the convolutional neural network produces a feature map in layer $n + 1$. In order to produce different feature maps when performing convolution on the same layer, we need filters with different weights. We use the Glorot Uniform Initializer (Glorot and Bengio 2010), as implemented by Keras:

$$W \sim U = \left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]. \tag{2.6}$$

We use equation 2.6 to initialize each W with a uniform distribution within the given range (Glorot and Bengio 2010). n_j is the number of neurons feeding into a node and n_{j+1} is the number of neurons is the number of neurons the output feeds to (Glorot and Bengio 2010).

2.4.2 Pooling

Pooling layers are commonly placed between convolution layers in order to down-sample the number of weights in a convolutional neural network. One commonly used type of pooling is called max pooling (Zhou and Chellappa 1988). In Figure 2.7 we can see an example of a 2×2 max pool using a stride length of 2. The figure shows 64 different 224×224 features maps pooled into

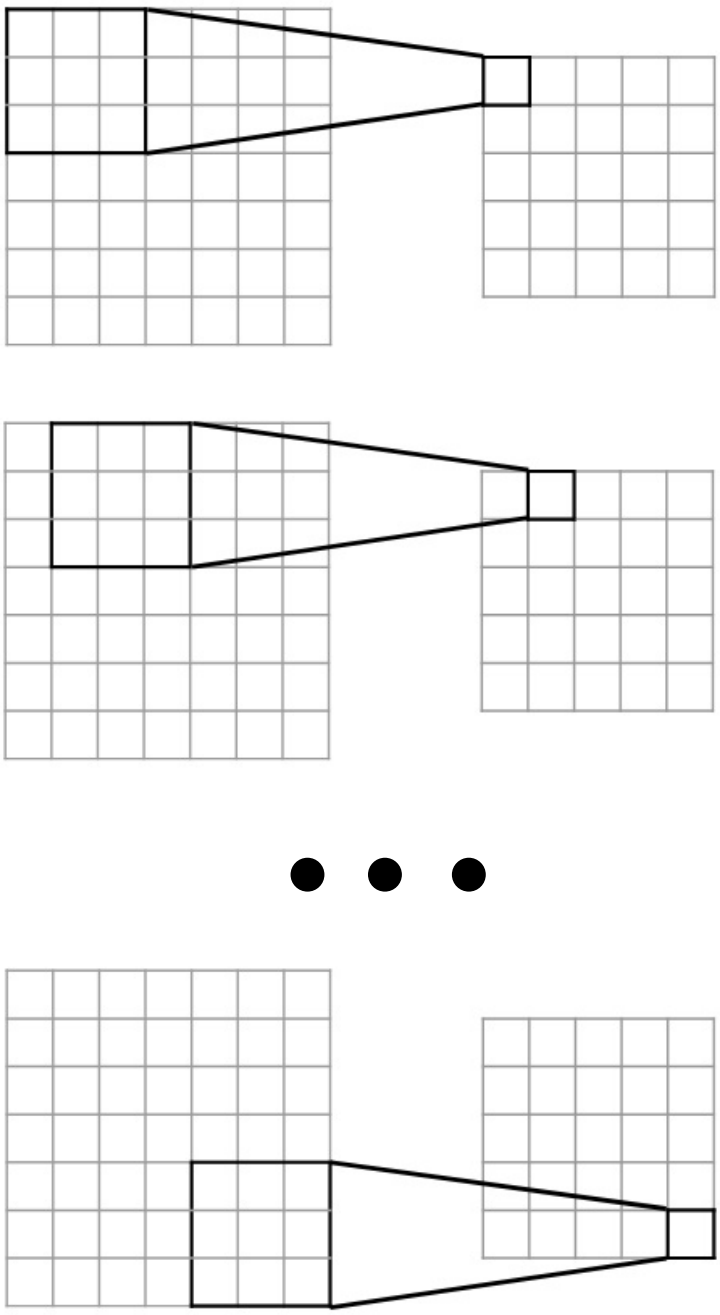


Figure 2.5: Visualization of a filter (or kernel) of size 3×3 being applied to each local receptive field in a 7×7 matrix to create a 5×5 output.

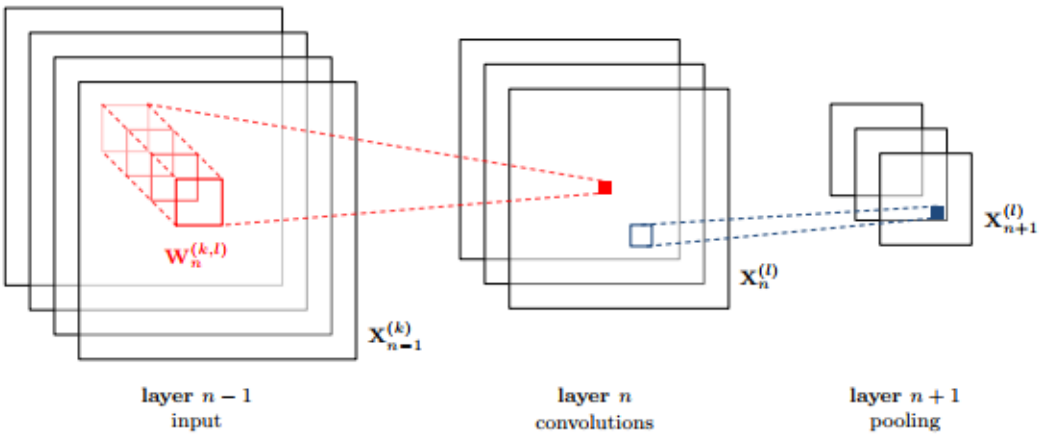


Figure 2.6: Image from (Dieleman et al. 2015).

64 different 122×122 feature maps. We will always have the same number of feature maps before and after a pooling layer, pooling will only reduce the width and height of the image. It is common to apply 3×3 pool with a stride of 2 (Szegedy et al. 2015), or a 2×2 pooling layer with a stride of 2 (Simonyan and Zisserman 2014). Since pooling does decrease the spatial dimensions of the feature maps, it can only be applied a limited number of times. For shallower neural networks, a pooling layer is sometimes applied between every convolution layer, but for deeper convolution layers, a pooling layer is typically only applied once every two to three layers.

2.5 Inception Network

As computation power grows, convolutional neural networks have become deeper and deeper. However, Szegedy et al. has a slightly different approach to deep CNNs other than just stacking additional convolution and pooling layers to the network (2015). The inception network was inspired by Lin et al., who

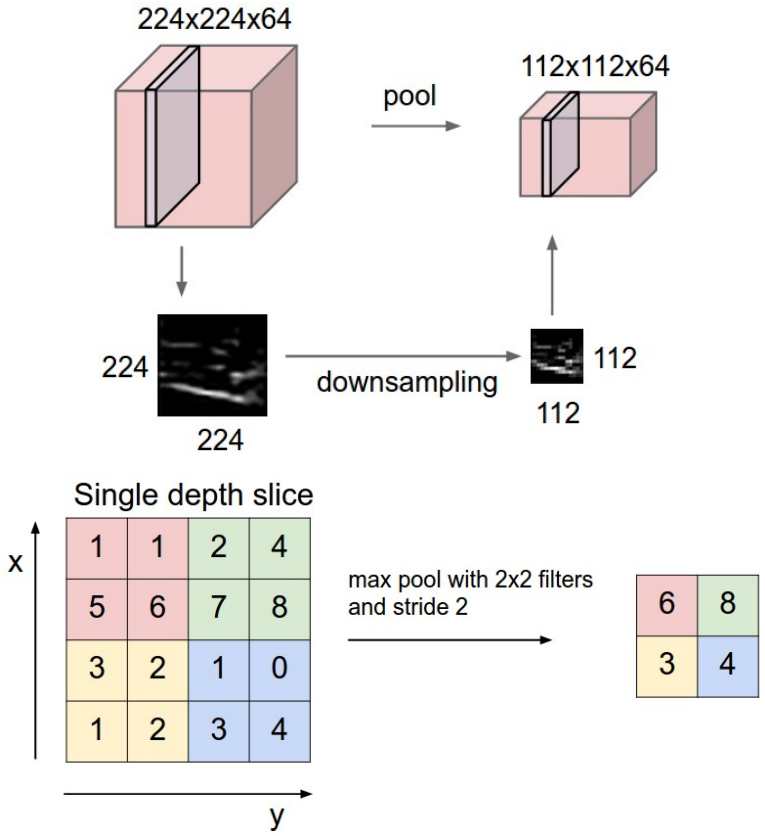


Figure 2.7: Example of max pooling using a pool size of 2×2 and a stride of 2. Image from (Karpathy 2016)

introduced the concept of a network within a network (2013). Pooling, 1×1 convolution, 3×3 convolution, and 5×5 convolution layers all come with their own advantages. A core concept of the inception network is not choosing a single filter size or pooling for each layer linearly, but rather applying them in parallel and then concatenating the results in an inception module (Szegedy et al. 2015). The inception network is created by stacking several inception modules together.

In practice, repeatedly applying 3×3 and 5×5 convolutions can be quite computationally expensive. The inception module often uses a 1×1 convolution with a low filter count in order to reduce the dimensionality before applying

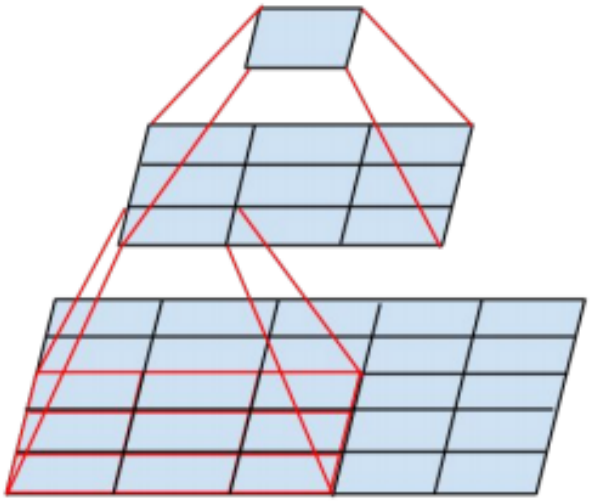
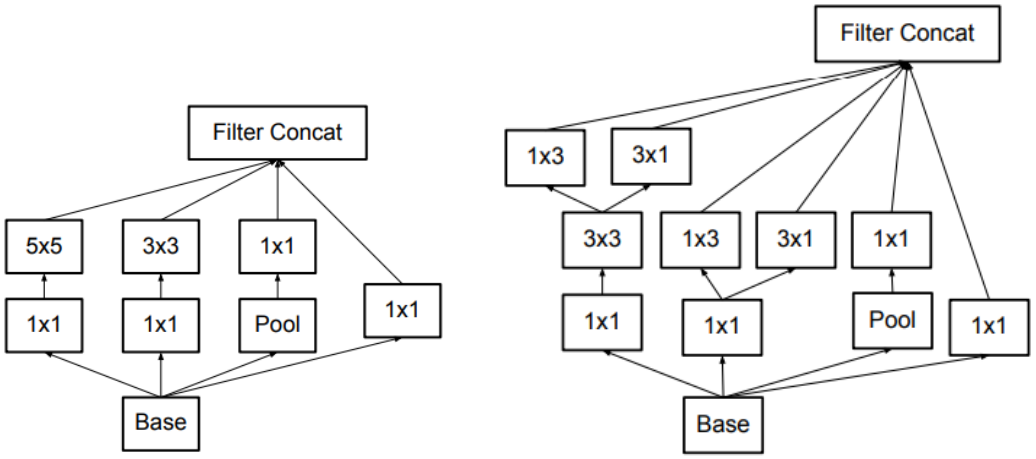


Figure 2.8: A mini-network replacing a 5×5 convolution with a 3×3 convolution. Image from Szegedy et al. (2016)

large expensive convolutions (Szegedy et al. 2015). For example, applying a 1×1 convolution with 20 filters on an image of size 28×28 with 50 filters results in an output size of $28 \times 28 \times 20$.

One core idea behind the inception-v3 network design is to replace larger convolutions filters with smaller ones (Szegedy et al. 2016). Larger filter sizes such as 5×5 and 7×7 can be "disproportionally expensive" when it comes to computation time compared to improved accuracy (Szegedy et al. 2016). Figure 2.8 shows a 5 x 5 filter being replaced by two layers of 3×3 convolution (Szegedy et al. 2016). Furthermore, a 3×3 convolution can be replaced by a 3×1 convolution followed by a 1×3 convolution.

Szegedy et al. combines the idea of applying a pooling, 1×1 convolution, 2×2 convolution, and 3×3 convolution simultaneously with splitting larger convolutions into smaller ones for the building block of the inception-v3 network.



(a) Original inception module design. (b) Updated inception module design for inception-v3 network

Figure 2.9: Image from Szegedy et al. (2016)

In Figure 2.9, we can see the original design of the inception module and the updated design for faster computation. Decreasing the computation time and bottlenecks of each layer allows for deeper convolutional neural networks. The inception-v3 network shown in Figure 2.11 is one of the models we train to classify bird roosts.

The inception-v3 network includes three different types of inception modules, as show in Figure 2.10. The inception-v3 network also contains an auxiliary classifier, a concept introduced in Szegedy et al. (2015). The auxiliary classifier computes the loss at earlier stages of the network to encourage learning in the lower layers and to increase the gradient signal by propagating error back earlier in the network (Szegedy et al. 2015). This is an important step in the very deep networks such as the 27 layer deep GoogLeNet (Szegedy et al. 2015). The inception-v3 network only includes one auxiliary classifier towards the top

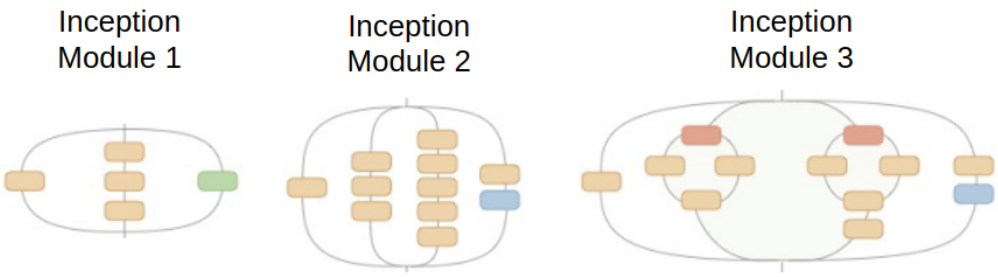


Figure 2.10: The three different inception modules used in the inception-v3 network. Image modified from Szegedy et al. (2016)

part of the network because adding the auxiliary network lower down did not improve the network convergence (Szegedy et al. 2016).

2.6 Batch Normalization

Batch normalization is a new and important technique that accelerates learning in deep neural networks and makes the network more robust to varying learning rates and parameter initializations. Batch normalization was first introduced in 2015 and it improved the accuracy of ImageNet classification while simultaneously speeding up learning 14 times (Ioffe and Szegedy 2015). During training, the weight of each layer in a deep learning network are updated at every learning step. This can be a problem since each network layer is affected by the parameters of all the previous layers so the later layers have to learn new weights as the output from earlier layers changes (Ioffe and Szegedy 2015). This phenomenon slows down learning and is referred to as internal covariate shift (Ioffe and Szegedy 2015).

Batch normalization comes from a concept called whitening, which linearly transforms the inputs x to have zero means and unit variances (Ioffe and Szegedy

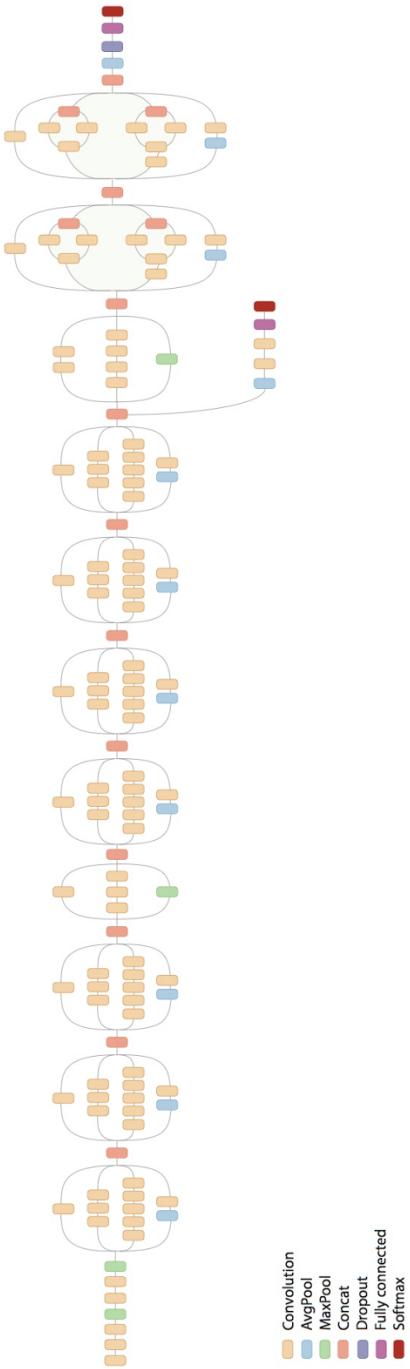


Figure 2.11: This is a visualization of the Inception-v3 network described in Szegedy et al. (2016). Each shape in the diagram represents a layer in the network (such as convolution or pooling layer). Image from Shlens (2016).

2015). It has been shown that this process helps training converge faster (Ioffe and Szegedy 2015). Whitening the inputs x of each layer is expensive and not always differentiable (Ioffe and Szegedy 2015). Instead, Ioffe and Szegedy normalize each scalar feature independently over the training data set (2015). The math for this step can be seen in Equation 2.7:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{Var\left[x^{(k)}\right]}}. \tag{2.7}$$

Always normalizing the layer input changes the layer. The variables γ and β are used to scale and shift the normalized values which "restores the representational power of the network" (Ioffe and Szegedy 2015). Both γ and β from Algorithm 2.1 are updated during network training.

Algorithm 2.1: Batch Normalization applied to activation x over a mini-batch. Algorithm from (Ioffe and Szegedy 2015)

Input : Values of x over a mini-batch $B = \{x_{1...m}\}$; Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu B)^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{(x_i - \mu B)^2}{\sqrt{\sigma_B^2 + \epsilon}}$ // normalize

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$ // scale and shift

---

When adding batch normalization to a network, first perform a dot product on the weights in layer inputs to compute a scalar value, then perform batch

normalization on a mini-batch, and finally apply the activation function. γ and β are used to learn whether to apply normalization to the input or not.

2.7 Transfer Learning

As we mentioned earlier, transfer learning allows us to train convolutional neural networks using a smaller number of labels. We transfer knowledge gained from training on one dataset to another similar dataset. For example, the weights of the CNN filters are learned when training on ImageNet and then fine tuned when training the new dataset (Shin et al. 2016). Transfer learning has affectively been applied to the inception network on image datasets. Transfer learning allows us to train deeper Convolutional Neural Network with fewer training labels since there are fewer weights and parameters that our network has to optimize. Convolutional Neural Networks can automatically learn complex features needed for object recognition given enough training data (Yue-Hei Ng et al. 2015). The inception network learned to classify ImageNet data with a 6.67% error without using handcrafted features or an external dataset (Szegedy et al. 2015). We hypothesize that we can reuse these same general image features to classify radar data.

Chapter 3

Dataset

3.1 Bird Roost Dataset

All of the input data for our machine learning models comes from publicly available level 2 NEXRAD radar data hosted on Amazon Web Services[1]. We obtained a list of radar datum that are known to contain a snapshot of birds leaving their roosts. We identify the datum by the radar name and time stamp. Our dataset consists of the ground truth roost locations collected by research groups at two different universities. The coordinates of 365 roost sites that were identified as purple martins were collected by the Purple Martin Conservation Association (Kelly et al. 2012), however only a subset of these were individually documented with a time stamp of when the bids are visible. The UNQC_CREF radar mosaic products were manually searched at known roost locations in order to identify roost in radars (Bridge et al. 2016). For more details, see the 'Compilation of roost data' section in Bridge et al. (2016). The remaining labels were collected using a crowd sourced approach by asking citizen scientists to label radar data online[2] (Laughlin et al. 2014).

Roosts form a distinct visual pattern in the radar data images. Roosts can be detected as they leave their roost every day around sunrise. Figure 3.1 shows

[1]https://aws.amazon.com/public-datasets/nexrad/

[2]http://radar.cs.umass.edu/roost-label

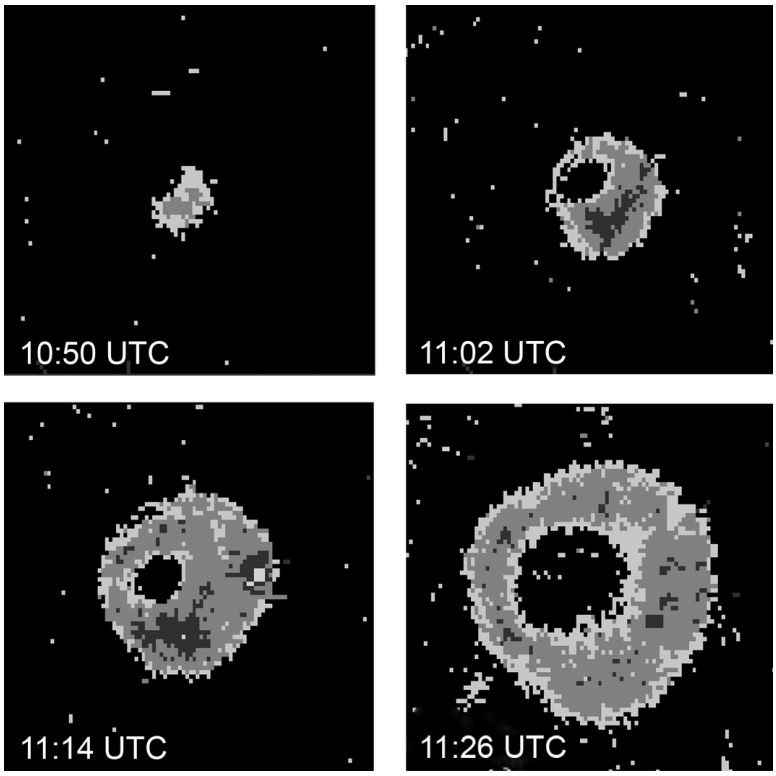


Figure 3.1: Example of birds leaving their roost. This particular image shows a roost in radar KHTX from 5:50 AM to 6:26 AM CDT on August 4 2015. Image from (Kelly and Pletschet 2017)

an example of what a birds leaving their roost looks like at different snapshots in time. This visual pattern was used to identify the bird roosts in our dataset. Reflectivity can be accurately represented with gray-scale as seen in 3.1, however the inputs to the convolutional neural networks use RGB values so that transfer learning can be applied.

3.1.1 Radar Products

Reflectivity, Doppler Radial Velocity, and Spectrum Width radar products are available to us in the non-polarimetric radar data. The dual-polarization radar

data contains all products of the legacy radars, in addition to Differential Reflectivity (Z_{DR}), Differential Phase (ϕ_{DP}), and Correlation Coefficient (ρ_{HV}). Reflectivity, Radial Velocity, Differential Reflectivity (Z_{DR}), and Correlation Coefficient (ρ_{HV}) are considered useful radar products for detecting bird roosts (Muller et al. 2015), so the other two parameters (Spectrum Width and Differential Phase (ϕ_{DP})) were not included in our machine learning input. Reflectivity refers to the intensity of the echo caused by the radar beam bouncing off its target and it has been show to be useful for detecting bird roosts, as well as calculating the density of birds (Diehl and Larkin 2005). Velocity is a useful radar product when studying birds because it helps determine which direction the birds are flying in (Gauthreaux Jr and Belser 1998). Weather and wind tend to move in a single direction, whereas birds leaving their roost fly in multiple directions creating a unique pattern that can be used to help locate birds roosts in radar data. Differential Reflectivity (Z_{DR}) is most explored polarimetric product for biology, which many applications successfully applied to determine the orientation of birds (Stepanian and Horton 2015). Correlation Coefficient (ρ_{HV}) has also been an important metric for detecting bird roosts since birds typically register a lower ρ_{HV} than meteorological echoes (Van Den Broeke 2013). Figure 3.2 shows a visualization of the four radar products that we use as the input data to our machine learning model.

3.2 Data Formatted for Machine Learning Input

The roost labels come from 10 different radars: KAMX, KBRO, KDOX, KGRK, KJAX, KHGX, KLCH, KLIX, KMLB, and KMOB. A distribution of labels by dataset can be seen in Figure 3.4, and a distribution of which labels came

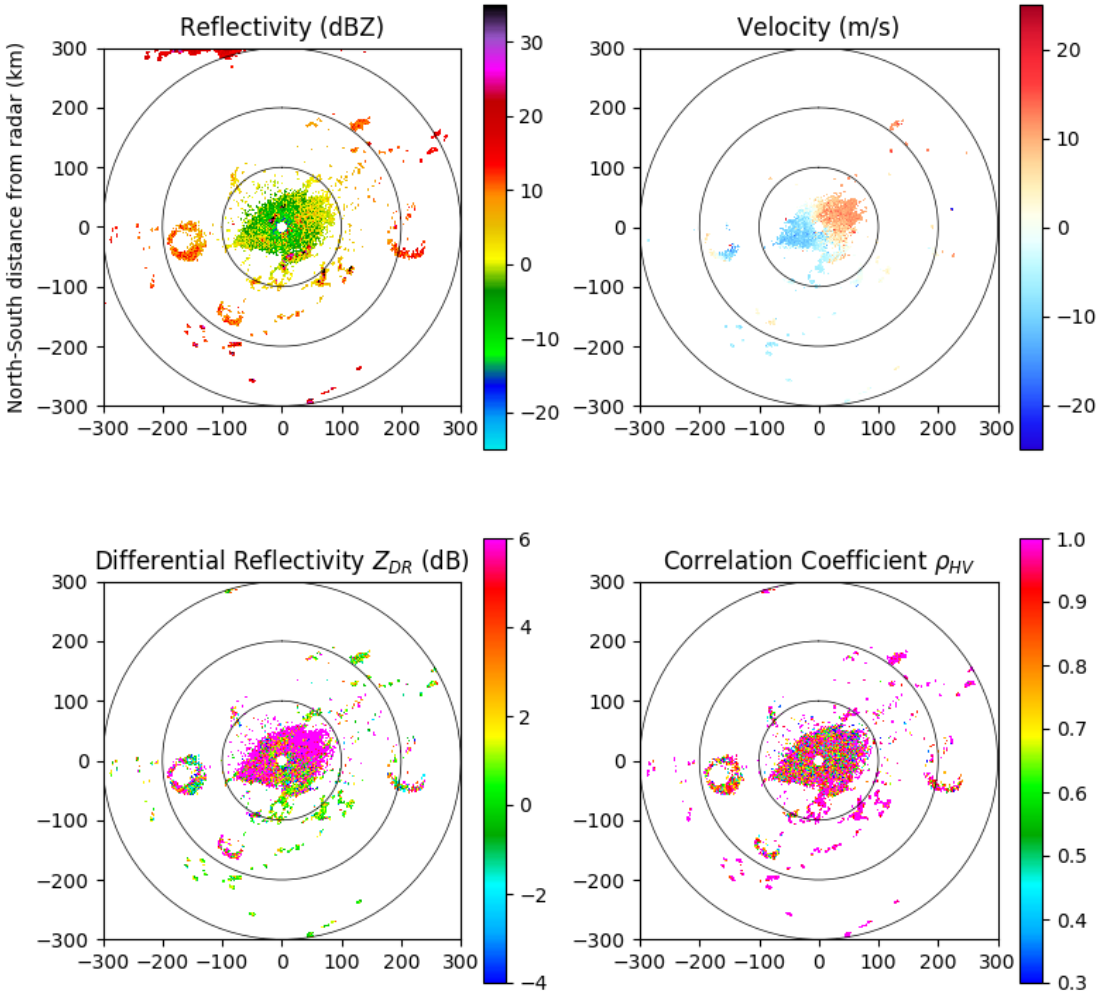


Figure 3.2: Machine Learning input. This is an example of a radar image that contains a roost. This is from the KMOB radar from July 4th 2015, 11:19 UTC. Image created using the Py-ART library (Helmus and Collis 2016).

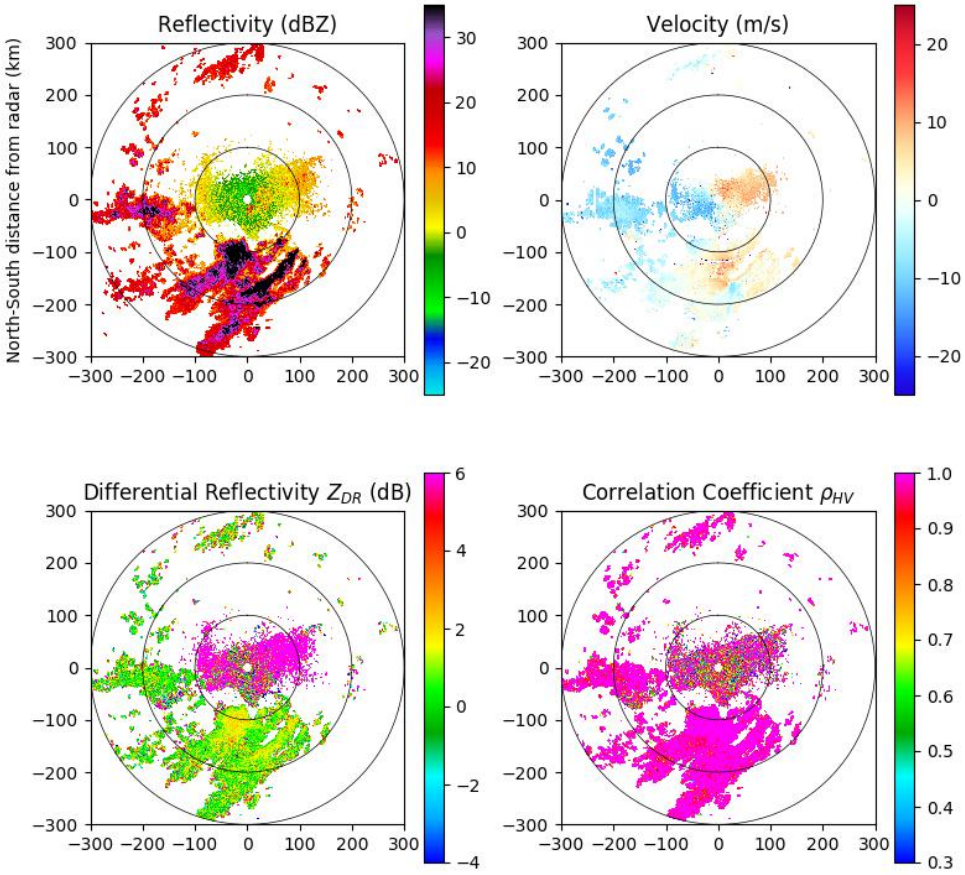


Figure 3.3: Machine Learning input. This is an example of a radar image that does not contain a roost. This is from the KMOB radar from July 2nd 2015, 11:27 UTC. Image created using the Py-ART library (Helmus and Collis 2016).

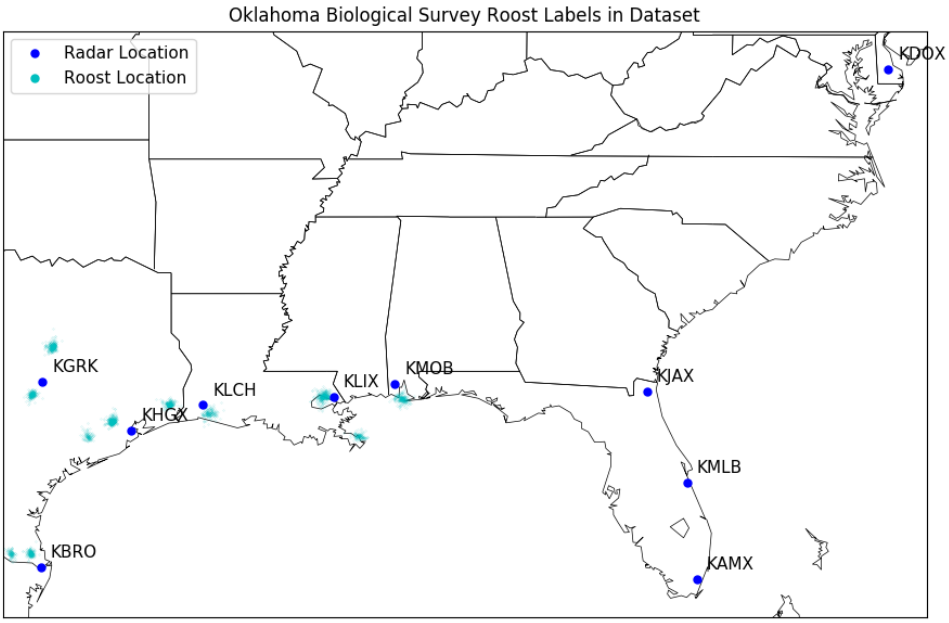
from legacy and dual-pol radar can be seen in Figure 3.5. Both of the datasets contained primarily positive labels of roosts found in radar since this is the data that the researchers were studying. As a result, only a few no roost radar data were recorded. In order to collect more negative labels, we selected radar data before and after the roosts were located. We collected negative labels starting 2 hours before until 1 hour before sunrise and 1 hour after until 2 hours after sunrise, leaving a 2 hour window in between. The majority of our negative labels were selected from this time period and the remaining negative labels came from the datasets described above. The noise in our radar images (dust, weather, sun-streaks, etc.) directly before, during and after the roost is visible in the radar is similar. This forces our machine learning algorithms to detect the roost itself in order to correctly classify our data. Sun-streaks appear in radar images more often close to sunrise, however they appear in both our roost and no roost data.

We converted each labeled radar datum to a 2D image. We detect bird roosts using the lowest level radar scan, since that is where the birds leaving their roost are most commonly detected (Chilson et al. 2012b). We convert the lowest level radar scan (0.5 degree) to Cartesian coordinates and save it as an image. We save the reflectivity, velocity, ρ_{HV}, and Z_{DR} radar products as individual images. In Figures 3.2 and 3.3 we see an example of what each of the lowest level radar scans saved as an image looks like. A color is assigned to each radar product using standard meteorological values. Figure 3.2 shows an example of a roost input and Figure 3.3 shows an example of no roost input. The roost images contain the roost ring pattern as seen in Figure 3.2. The no roost images contain many different types of noise including weather, dust,

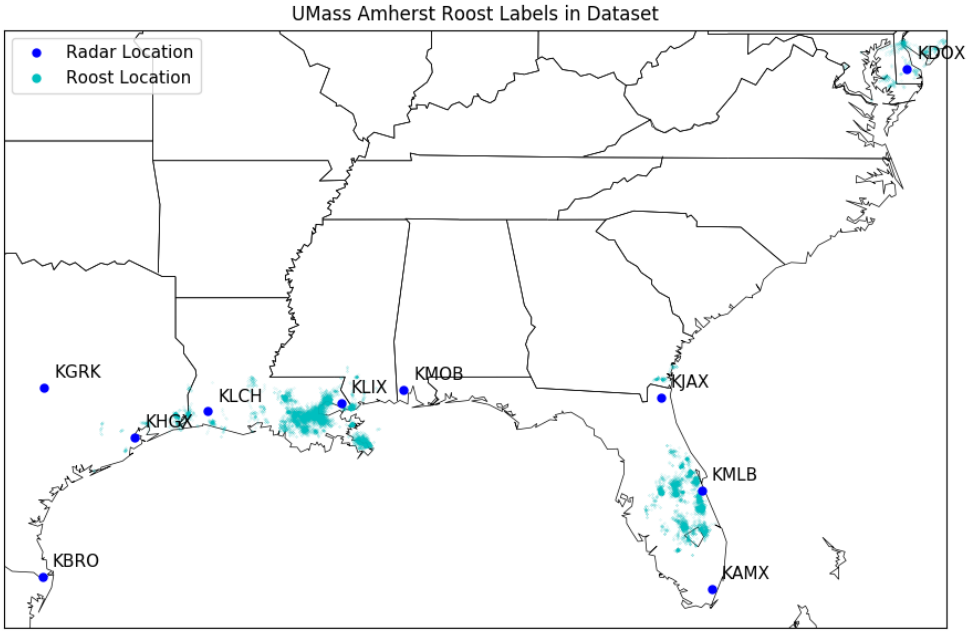
	Roost	No Roost
Legacy	11,112	19,939
Dual-Pol	1,346	10,806

Table 3.1: The distribution of labels. This table lists how many dual-pol radar labels exist within the data.

insects, etc. These images serve as the input to our machine learning models. Table 3.1 shows how many training labels we have as inputs to our model.

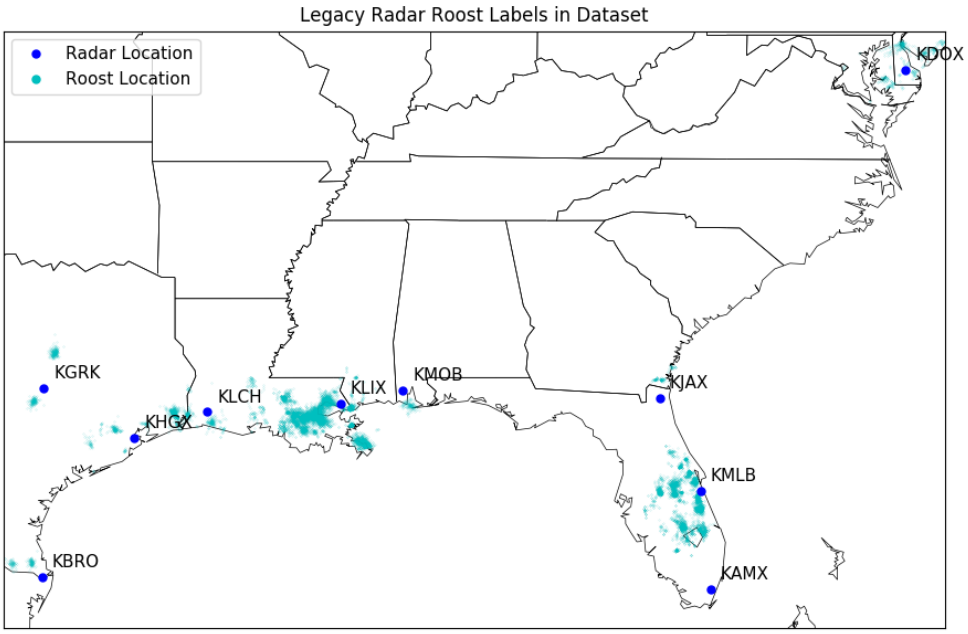


(a) Oklahoma Biological Survey roost data

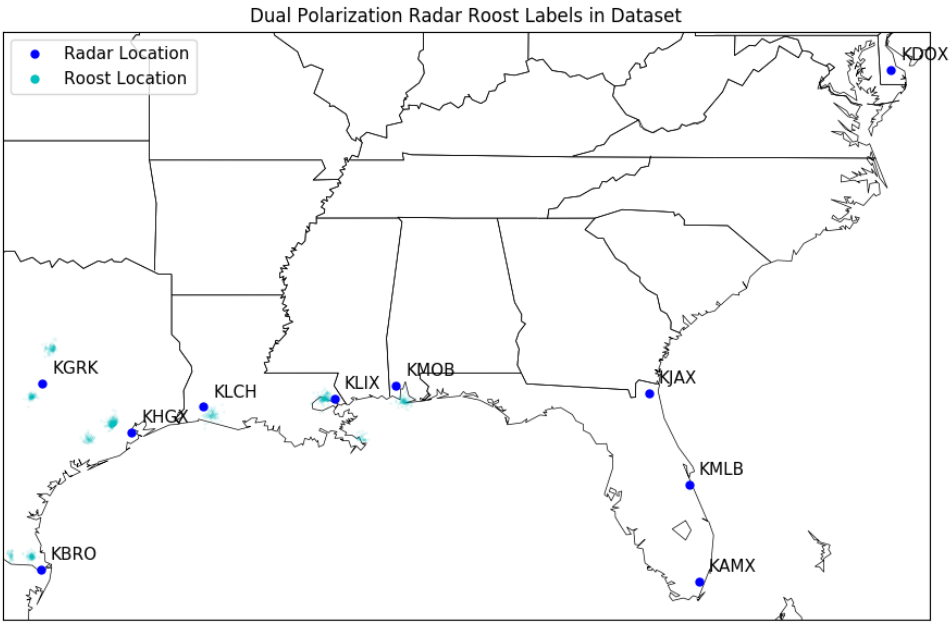


(b) UMass Amhert citizen science roost data

Figure 3.4: Visual distribution of roost labels from the Oklahoma Biological Survey and UMass Amhert citizen science labels. This figure shows where each roost was found.



(a) Legacy radar roost data



(b) Dual-Pol radar roost data

Figure 3.5: Visual distribution of roost labels for legacy radar and dual-pol radar data.

Chapter 4

Design: Neural Network Architectures

When creating ANN or CNN network architectures it is important to find the simplest version that stills works effectively. Canziani et al. (2016) compares different network architectures, their complexity, their training time, and their accuracy. Small accuracy increments can often lead to a large computation cost increase as the accuracy approaches its upper bound (Canziani et al. 2016). By incrementally making changes and adding complexity or layers to a simple network, it's possible to find the point where added complexity results in diminishing returns. We compare artificial neural network performance to convolutional neural network performance. We also compare how well deep CNNs trained with transfer learning perform compared with shallow CNNs trained from scratch.

4.1 High Level Design

In order to have a fair comparison for the ANN, shallow CNN, and deep CNN models, we create an overall design that is similar for each of the three models. Image classification is generally done on a single image, but we have four separate radar product images that we use to give a single classification. We also want to compare how well machine learning models can perform on each individual radar product.

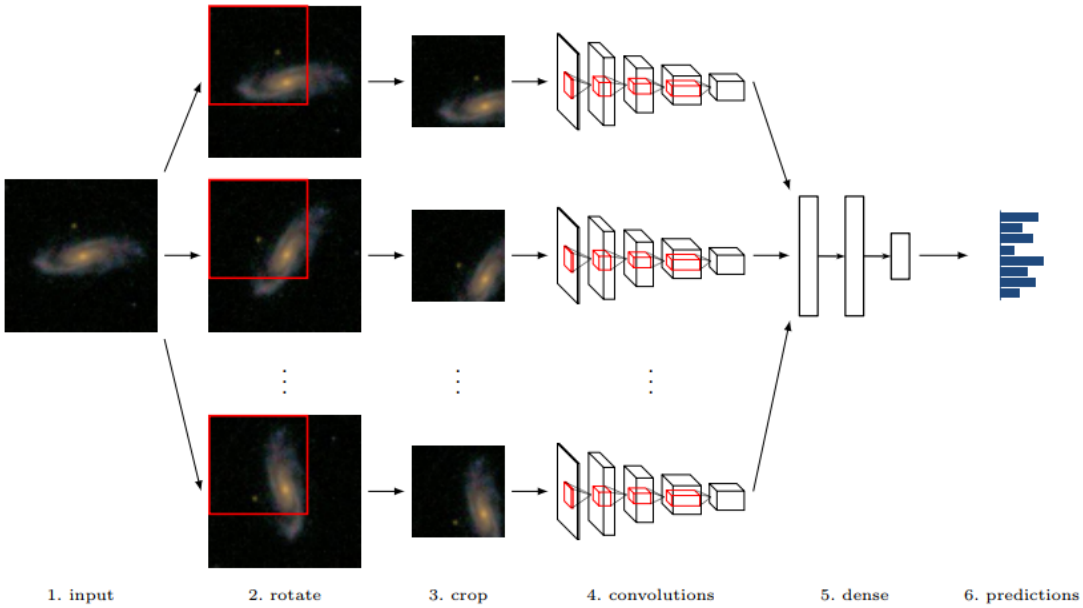


Figure 4.1: Overview design of a convolutional neural network for classifying rotational-invariant galaxy images. Image from (Dieleman et al. 2015).

We use a layered machine learning design inspired by a rotational-invariant convolutional neural network galaxy classification architecture as seen in Figure 4.1 (Dieleman et al. 2015). The design of their network took a single image and rotated it in three different ways. Each rotated image was cropped and then input into three separate convolutional neural networks. The output from each convolutional neural network was then combined in dense neural network layers to obtain the aggregate prediction (Dieleman et al. 2015). Instead of using rotated images as inputs to separate CNNs we use individual radar field images as the inputs to separate CNNs. The overall architecture for the roost classification model for both dual-pol and legacy radar data can bee seen in Figures 4.2 and 4.3, respectively. Unlike the Dieleman et al. paper, our model does not crop or shift the images. Radars are stationary, so it would not make

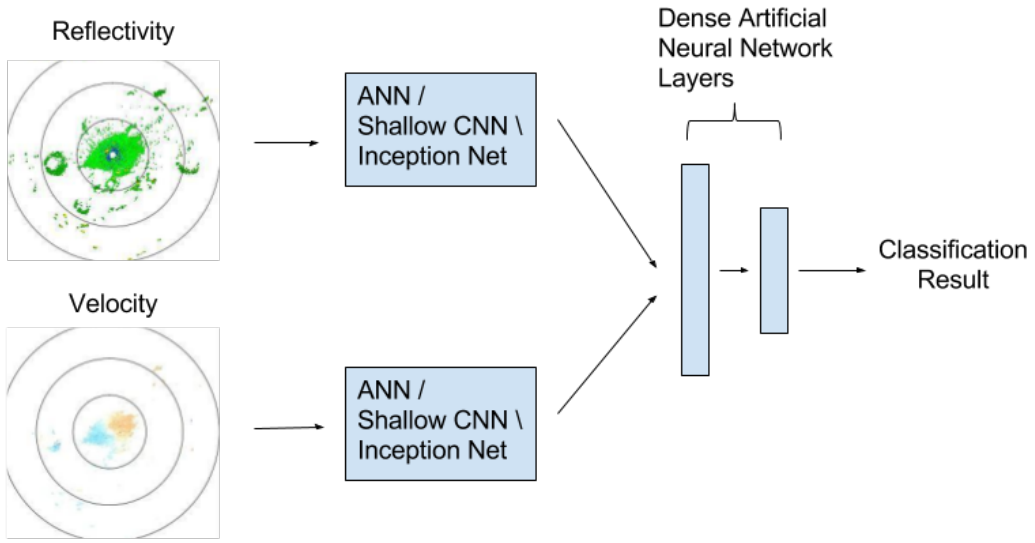


Figure 4.2: Design of the machine learning classification system for legacy radar data.

sense to learn different orientations of the radar imagery. As one can see from Figure 3.2, there is a lot more noise in the center of the image. The radar scans closest to the radar are also much closer to the ground, and thus more clutter, such as dust and insects is detected in the center of the radar scan. The machine learning model needs to learn to ignore the noise at the center of the image.

Another key difference between our network design and the galaxy classification network is that the output from our convolutional neural networks is not directly connected to the fully connected (or dense) layers. As previously mentioned, we want to be able to compare the accuracy of classification using only legacy radar fields, using all of the radar fields, and using each radar field individually. To avoid retraining the exact same component of the model multiple times for different networks, we train the convolutional neural networks independently. Another reason to train the networks on each radar field separately

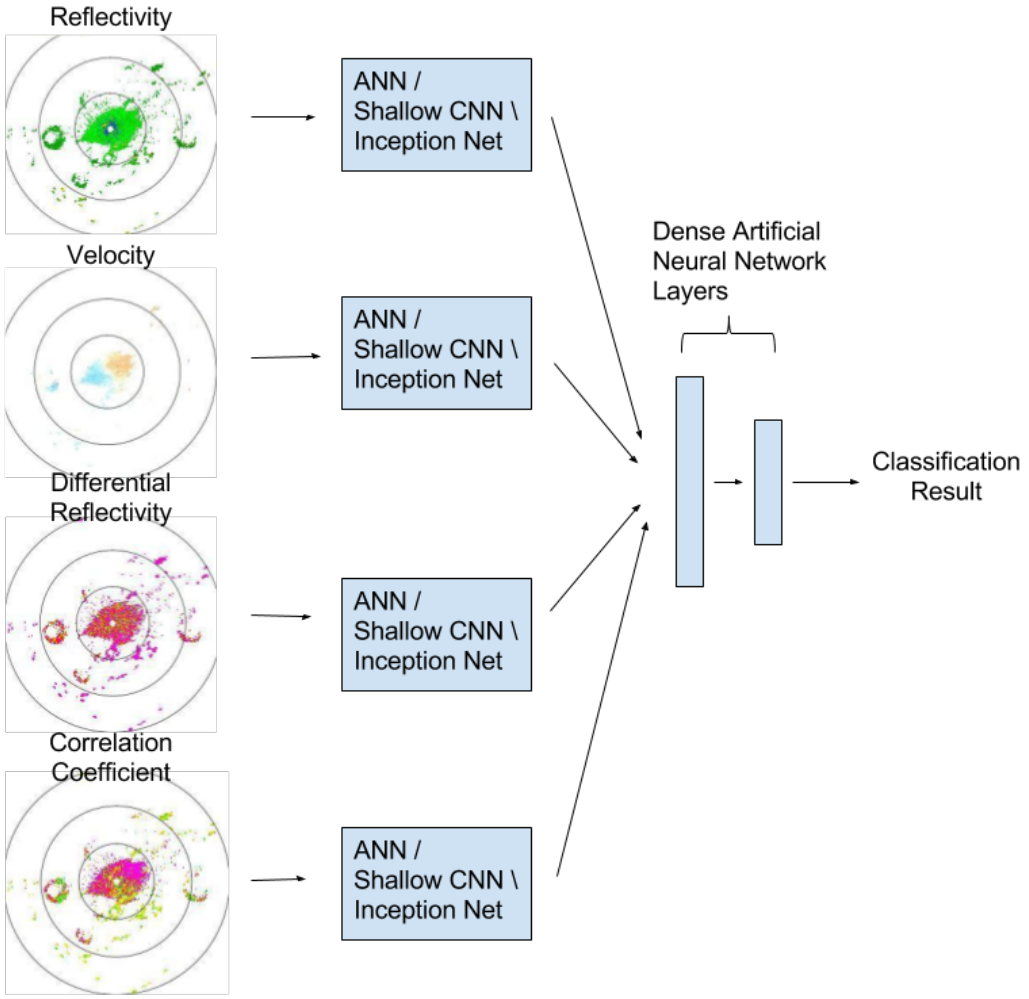


Figure 4.3: Design of the machine learning classification system for dual polarization data.

type	BN	nodes	activation
Input Layer		2 or 4	
Hidden Layer	Y	16	ReLU
Output Layer	N	2	softmax

Table 4.1: The design of the aggregate classifier. This table gives a full description of the design of each layer of the network. BN stands for batch normalization.

is to minimize the number of networks weights that need to be learned simultaneously. A neural network trained to classify bird roosts using four images as input would need to learn four times the number of network weights from 4 times as many input values. We train each of our CNNs on the individual radar parameters: Reflectivity, Radial Velocity, Differential Reflectivity (Z_{DR}), and Correlation Coefficient (ρ_{HV}). The networks learn to optimize classification results using a single radar field as part one of our layered machine learning design.

In the second part of our design, we trained a feed-forward artificial neural network to output a roost or no roost classification given four input values. The input values are the four (two for the legacy radar data) classification probabilities output from the individual radar field networks. This ANN only had a single hidden layer with 16 nodes as described in Table 4.1. For the rest of the paper we will refer to this network as the aggregate classification ANN. The aggregate classification ANN relies on accurate input probabilities in order to perform well since it doesn't see the radar data directly.

4.2 Neural Network Architectures

We compare several different neural network architectures to see how well they can classify radar field images. In this section we will describe the network structure of these three different neural networks. The artificial neural network and the inception-v3 network were trained using back propagation and stochastic gradient decent. The shallow convolutional neural network was trained using the variant on stochastic gradient decent called Adam.

4.2.1 Artificial Neural Network

We use a feed forward artificial neural network as our simplest model for comparison. The results and design from this network come from Katherine Avery's Honors Thesis (Avery 2018). The ANN we used took a 240 by 240 pixel grayscale image as the input. When training on reflectivity the three dense hidden layers consisted of 64, 32, and 8 nodes. When training the Velocity, Z_{DR}, and ρ_{HV}, the first hidden layer consisted of 128 nodes instead of 64. Each of the hidden layers was followed by batch normalization and a ReLU activation function, and the output layer was followed by batch normalization and a softmax activation function. This model was trained with a learning rate of 0.01.

4.2.2 Inception-V3 Network with Transfer Learning

We train a deep convolutional neural network using transfer learning since the bird roost dataset does not have enough labels to fully train a deep convolutional network from scratch. The deep CNN that we chose was the Inception-V3 network since it achieved great results on ImageNet data (Szegedy et al. 2016). We gave a full description of the inception-v3 network in the background section

type	BN	nodes	activation
Input Layer		240×240	
Hidden Layer	Y	128 or 64	ReLU
Hidden Layer	Y	32	ReLU
Hidden Layer	Y	8	ReLU
Output Layer	N	2	softmax

Table 4.2: The design of the artificial neural network. This table gives a full description of the architecture and number of nodes in each layer of the network.

in Chapter 2. The weights of the network were fully trained using the ImageNet dataset. The last layer of the network was then retrained using radar image data to detect bird roosts. Theoretically the network learned all of the useful features and information about images from ImageNet, and then the network was retrained to learn how those features translate to detecting bird roosts in radar images. The network was trained using a learning rate of 1e-04.

4.2.3 Shallow Convolutional Neural Network

Another network we compare will be referred to as the shallow CNN from this point on in this thesis. The shallow CNN used only two convolution layers. Adding more layers in the network did not improve results, but did increase training time. The network took in an image with 240×240 pixels and RGB channel values. The shallow network contains two convolution layers with 5 x 5 kernels. The first convolution layer contains 32 filters, the second contains 64 filters, and each is followed with a max pooling layer. Both pooling layers have a pool size of 2 x 2 and stride of 2. The output of the second pooling layer was the input for the a dense layer with 500 nodes with ReLU activation

type	BN	filters	nodes	kernel	pool	stride	activation
Convolution	Y	32		5 x 5		1	ReLU
Max Pool					2 x 2	2	
Convolution	Y	64		5 x 5		1	ReLU
Max Pool					2 x 2	2	
Fully Connected	Y		500				ReLU
Fully Connected	N		2				softmax

Table 4.3: The design of the shallow convolutional neural network. This table gives a full description of the design of each layer of the network.

and batch normalization. The final layer provided a probability classification using the softmax activation function. We used accuracy as the metric to see how well our model was training and binary cross-entropy as the loss function. This network was trained using a learning rate of 1e-4. Learning rates higher than 1e-04 produced less stable results and learning rates lower than 1e-04 led to longer training with without any improvement in accuracy.

Chapter 5

Results

5.1 Scoring Metrics

When evaluating the classification results of our machine learning models, we use four different metrics. We evaluate the total accuracy (ACC), the true positive rate (TPR), the true negative rate (TNR), and the area under the ROC curve (AUC). ACC, TPR, and TNR can all be calculated by counting the number of true positives (TP), true negatives (TN), false positive (FP), and false negatives (FN).

$$ACC = (TP + TN)/(TP + FN + FP + TN) \tag{5.1}$$

$$TPR = TP/(TP + FN) \tag{5.2}$$

$$TNR = TN/(FP + TN) \tag{5.3}$$

		Actual Label	
		Roost	No Roost
Predicted Label	Roost	TP	FP
	No Roost	FN	TN

Table 5.1: A binary contingency table for whether or not a Roost is found in radar data.

Accuracy allows us to look at how many correct classifications the network produced overall. A high true positive rate should be favored when the goal is to miss as few bird roosts as possible. A high true negative rate should be favored when a few missing roosts is acceptable, but we want to spend minimal time eliminating false positives from the data. AUC is useful because this metric is not biased by an unbalanced dataset. Each metric has a purpose, and the correct metric to chose will vary for different research purposes.

A receiver operating characteristic (ROC) curve can provide a richer classification measure than the scalar metrics used above, as well as creating a visualization of the classification results. When comparing the results from different ROC curves we can use the AUC metric (Fawcett 2006). The AUC values range from 0 to 1 where a perfect classifier will achieve a score of 1 and random guessing will score 0.5 (Fawcett 2006). The AUC value of a classifier is the probability that the model will rank a positive classification higher than a negative classification (Fawcett 2006). An AUC value of .9 or above is considered to be a good result.

We evaluate our results using cross-validation. K-fold cross-validation is the process of partitioning the data into k subsets of the data (folds), training on $k-2$ folds, validating on 1 fold, and testing on the remaining fold (Kohavi et al. 1995). The training is repeated k times, where each fold is used as the testing and validation fold exactly once. K-fold allows us to evaluate every labeled datum. We use 5-fold cross-validation to train and evaluate our models. We chose a small number for k because convolutional neural networks are computationally expensive to train.

For each of our metrics we calculate the confidence interval using the bootstrapping percentile method. The percentile method calculates the chosen metric (for example, loss or accuracy) on randomly selected samples of the data iteratively (Efron and Tibshirani 1986). Then for a 95% confidence interval we take the upper and lower 2.5% points of distribution (Efron and Tibshirani 1986). This is a range that 95% of the bootstrapped samples fall within. The upper and lower bound of the distribution become the confidence interval for the performance metric (Efron and Tibshirani 1986).

Each testing fold was evaluating using its corresponding network. The results from each of the testing folds are then combined. To compute the confidence intervals for ACC and AUC we randomly select one thousand samples with re-sampling from the combined testing results. For TPR we select one thousand samples from the roost data and for TNR we select one thousand samples from the no roost data. We repeat this process for one thousand iterations on each of our metrics in order to compute the confidence intervals. We have more confidence in our legacy radar results since we are able to samples from a larger dataset.

5.2 Classification Results

Of the three different machine learning networks we trained, the shallow CNN and Inception network aggregate classifiers produced the best results with an accuracy of 90 percent. The inception aggregate classifier has the highest true positive rate, and the shallow CNN Dual-Pol aggregate classifier has the highest true negative rate. These results are averages from five runs of each of the

networks. The full results for each network are included in Table 5.2. The box plots of the confidence intervals are found in Figures 5.2 and 5.1.

We hypothesized that both the Inception-v3 network and the shallow CNN would outperform the traditional Artificial Neural Networks since Convolutional Networks are better at learning spacial relations. The results from the ANN are preliminary results so we can only compare accuracy as our metric. The Shallow CNN had a higher accuracy than the traditional ANN in everything except for reflectivity. The traditional ANN's accuracy was in the low eighties for each of the different radar products (see Figure 5.7).

The ANN trained from scratch produced better results than transfer learning on the Inception-v3 network. We expected the Deeper Convolutional Neural Network to perform better since it takes many network layers to fully create custom features. We also expected the Inception-v3 network to outperform the shallow CNN because deep CNNs have performed very well on many image datasets. It's worth noting that the Inception-v3 aggregate networks performed very well and are comparable with the shallow CNN results even though the individual radar products did not perform as well on their own.

We believe that transfer learning would have worked better if the inception network was initially trained on a large set of radar data, for example to predict different severe weather systems from radar data. Natural images are different than radar images and may require different convolutional filters that may not necessary translate to radar data. Natural images contain shadow, light, objects in the foreground and background, lines, edges, etc. It may also have helped if we trained the lower layers of the Inception-v3 network instead of relying on ImageNet to find useful features for radar data. Another reason the Inception-v3 network may not have worked as well as expected is that we did not have enough

	ACC	TPR	TNR	AUC
ANN				
Reflectivity	.829	-	-	-
Velocity	.817	-	-	-
Differential Reflectivity	.821	-	-	-
Correlation Coefficient	.824	-	-	-
Inception				
Reflectivity	.757 – .808	.803 – .848	.728 – .779	.839 – .884
Velocity	.770 – .819	.789 – .837	.757 – .807	.851 – .894
Differential Reflectivity	.746 – .796	.819 – .865	.732 – .787	.847 – .905
Correlation Coefficient	.712 – .766	.774 – .824	.705 – .760	.805 – .874
Legacy Aggregate	.848 – .889	.857 – .896	.841 – .885	.929 – .956
Dual-Pol Aggregate	.906 – .938	.923 – .952	.904 – .937	.971 – .990
Shallow CNN				
Reflectivity	.873 – .912	.785 – .832	.920 – .950	.937 – .964
Velocity	.636 – .692	.000 – .002	.999 – 1.00	.684 – .754
Differential Reflectivity	.882 – .919	.727 – .781	.903 – .936	.912 – .953
Correlation Coefficient	.901 – .935	.697 – .751	.927 – .955	.910 – .956
Legacy Aggregate	.875 – .913	.797 – .844	.915 – .947	.930 – .961
Dual-Pol Aggregate	.905 – .938	.813 – .857	.916 – .948	.931 – .970

Table 5.2: Results for each model and each metric: ACC - Accuracy, TPR - True Positive Rate, AUC - Area Under Curve. These results show the bootstrapped confidence intervals.

radar data, especially polar-metric radar data, to effectively train this network. The inception network learns to utilize a wide range of image properties in the features and it may take more training data to fully utilize this information.

Although the shallow CNN outperformed the Inception-v3 network, the shallow CNN over-fit more on the training data. The training accuracy and loss from the inception network were consistently more similar for the inception network than for the shallow CNN as seen in Figures 5.3, 5.5, 5.4, and 5.6. The shallow CNN had more over-fitting and even approached 100 percent accuracy on the training data for every radar product that we trained on. Once the training network loss is zero, the network will stop learning. This still allowed the shallow CNN to approach 90 percent accuracy on the test data. It is unfortunate that our learning was capped here. It is possible that the network would over-fit less if we had more training data.

In addition to the learning curves in Figures 5.3 and 5.5 we also included the loss of the networks in Figures 5.4 and 5.6. The loss was computed using binary cross entropy. Note that for the inception network the loss continues to go down even after the network accuracy stops improving. This could be because the computed probabilities are getting better, which is not directly reflected in the accuracy. If a roost is classified as a roost with a probability of .7 or .8 then the accuracy for both examples will be the same (correct), but the loss will be lower for the one classified with a .8 probability. We use a threshold of .5 for the classification.

The ROC curves for both the inception network and the shallow CNN are displayed in Figures 5.8 and 5.9. The shallow CNN has a higher AUC for everything except for the velocity network. It is interesting to note that the shallow CNN trained only using velocity classified all of the images as No Roost

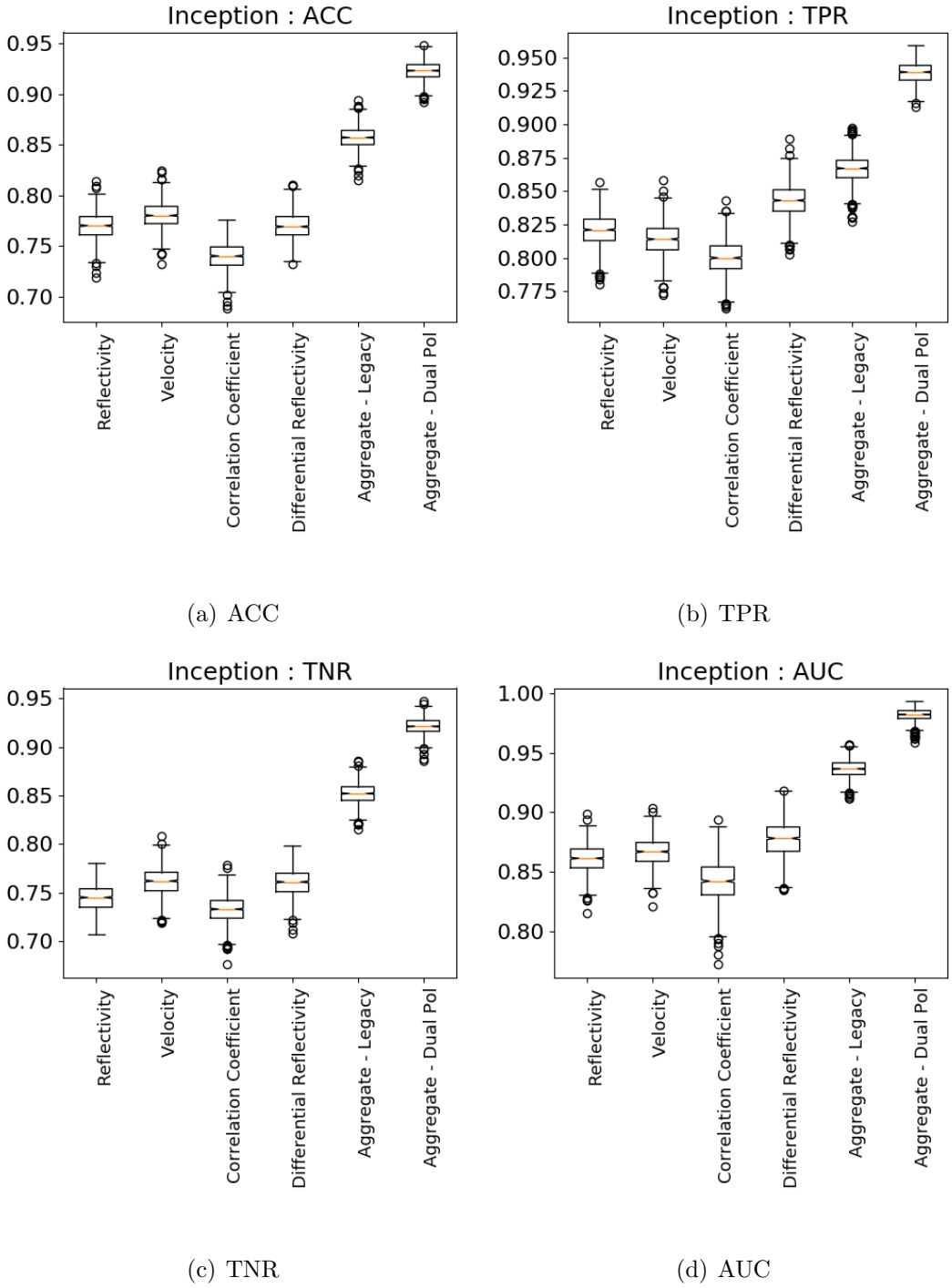
while the inception network was able to learn to classify velocity with a ACC of 78%.

5.2.1 Aggregate Classification Results

Our classification model when using multiple radar products to classify bird roosts is a layered machine learning problem. Our models will assign a classification probability to each radar product in the first part of the problem. These classification probabilities serve as the inputs to our second model. Ideally we would have two separate validation sets for the two different machine learning steps, however we did not have a sufficient number of labels to create two validation sets. Our dataset was split into three different groups that we will refer to as A, B and C. Group A contains 60 percent of the data and the remaining 40 percent is split equally between group B and C. When training the models to detect bird roosts from image data for a single radar product we use A as our training set, B as our validation set, and C as our testing set. For the second stage of learning we input the probabilities of the different radar products into a 1 hidden layer neural network for the aggregate classifier. At this stage of the problem we use B as our training set, A as the validation set, and C as the test set. C is consistently used as the test set throughout. The learning curve in Figure 5.10 and loss curve in Figure 5.11 shows that our new validation data outperforms the training set. This is not surprising since the training set data produced a higher accuracy during the first part of the machine learning so probabilities we input into our machine learning model were more accurate.

The ROC curve for the aggregate classifier can be seen in Figure 5.12. The final classification step improved the ACC, TPR, TNR, and AUC of the inception network. The aggregate classifier didn't improve the shallow CNN results

as much, but the shallow CNN individual radar product did fairly well so it was harder to improve. The Inception-v3 network and shallow CNN Dual-Pol aggregate classifiers have the highest accuracy of all of the results. Of these two networks Inception-v3 Dual-Pol classifier had a higher TPR and AUC, however the shallow CN Dual-Pol aggregate classifier had a higher TNR.



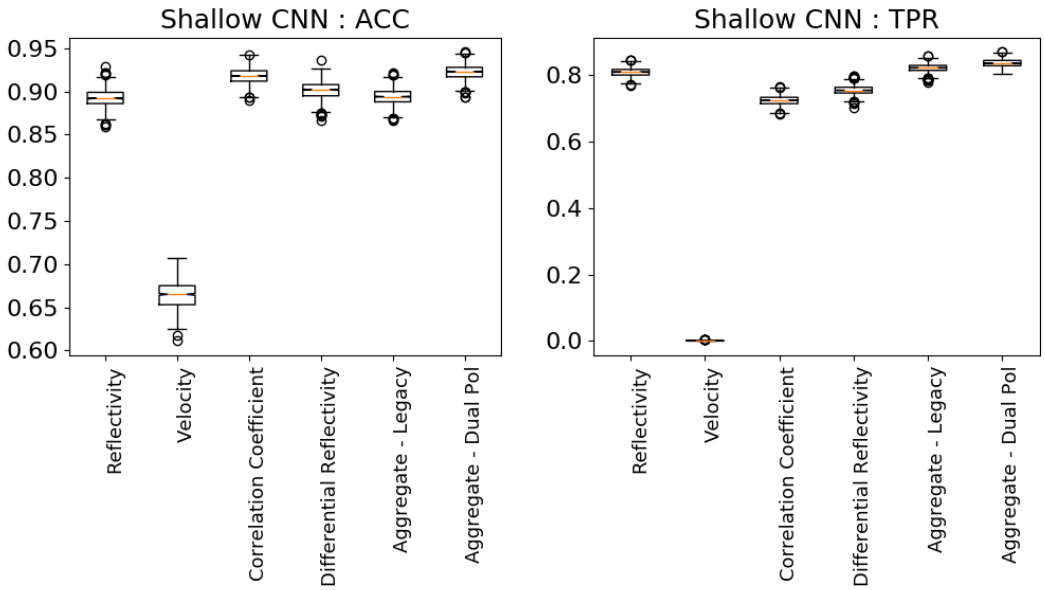
(a) ACC

(b) TPR

(c) TNR

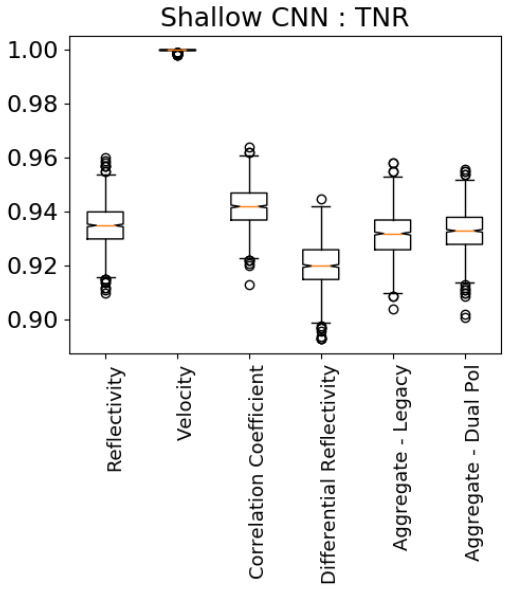
(d) AUC

Figure 5.1: The confidence intervals for each metric evaluated on the Inception-v3 network.

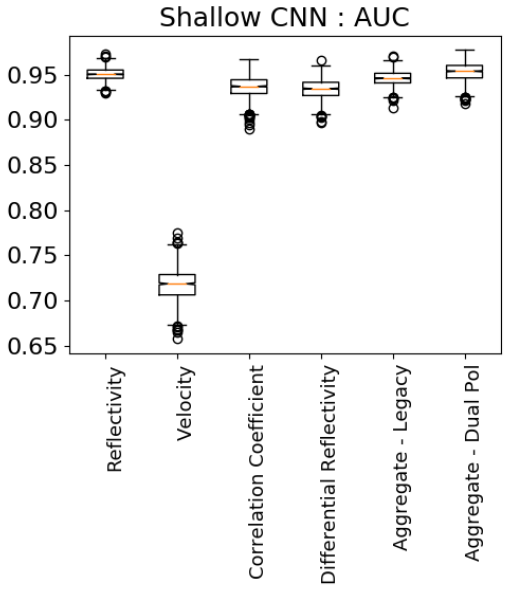


(a) ACC

(b) TPR



(c) TNR



(d) AUC

Figure 5.2: The confidence intervals for each metric evaluated on the shallow CNN.

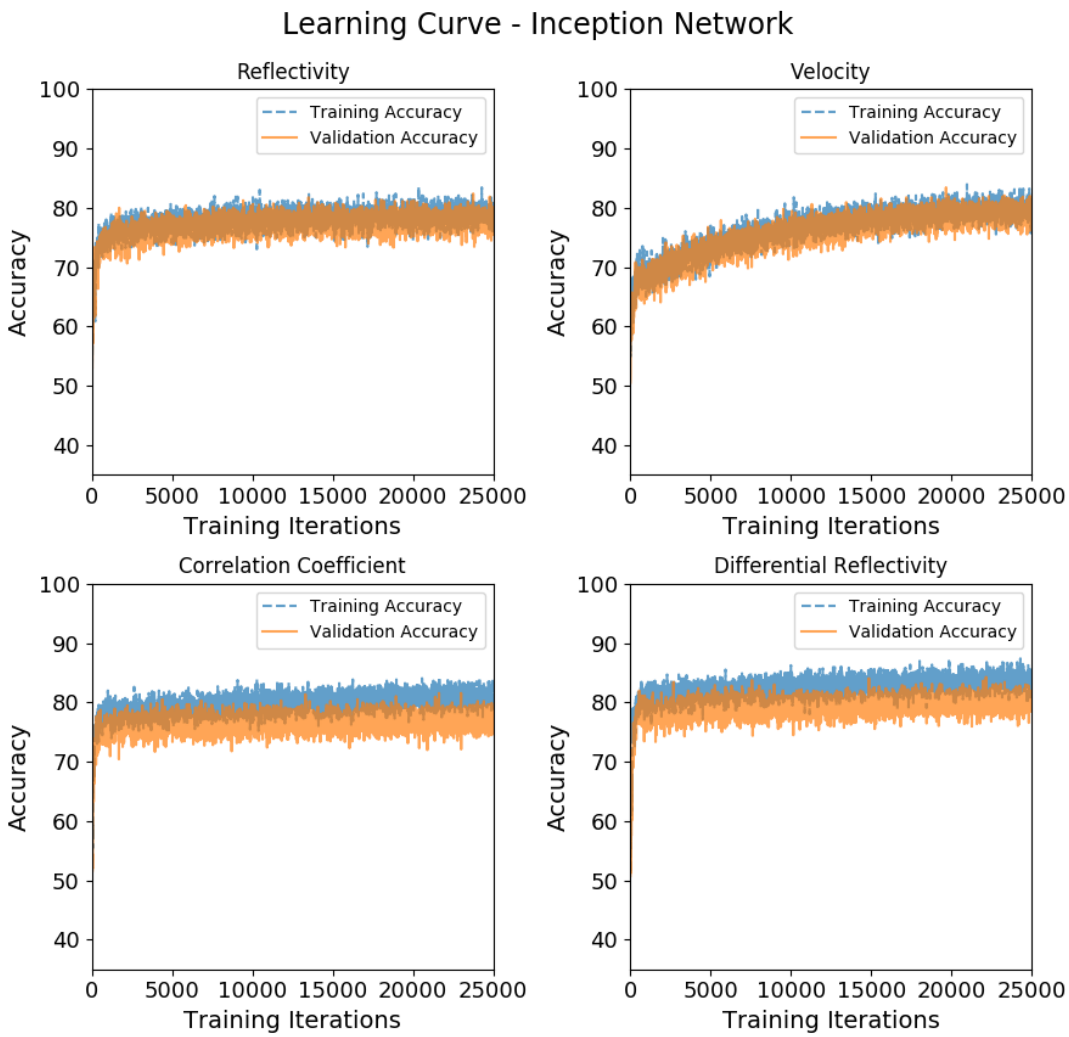


Figure 5.3: Learning Curve for retraining the inception network. Average learning curve from five runs.

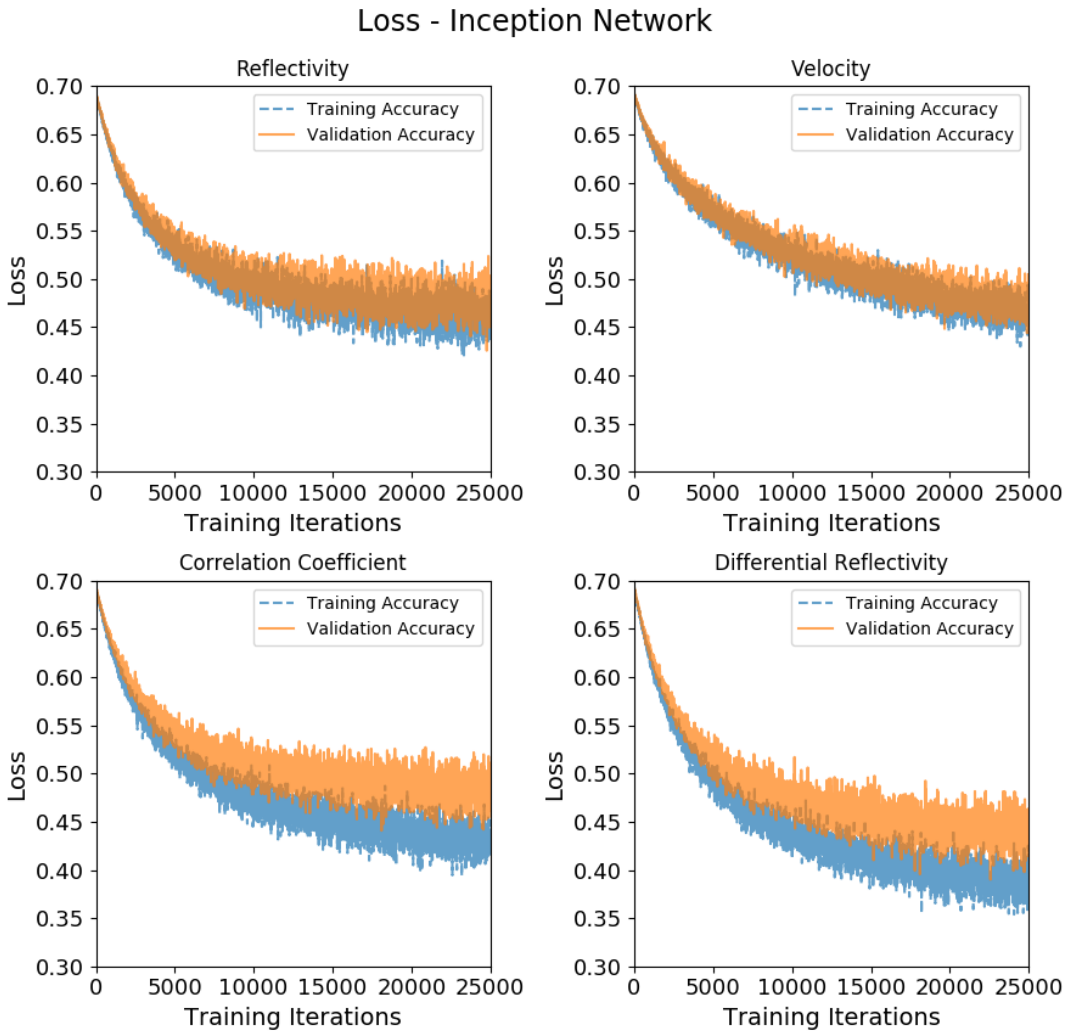


Figure 5.4: Loss for retraining the inception network. Average learning curve from five runs.

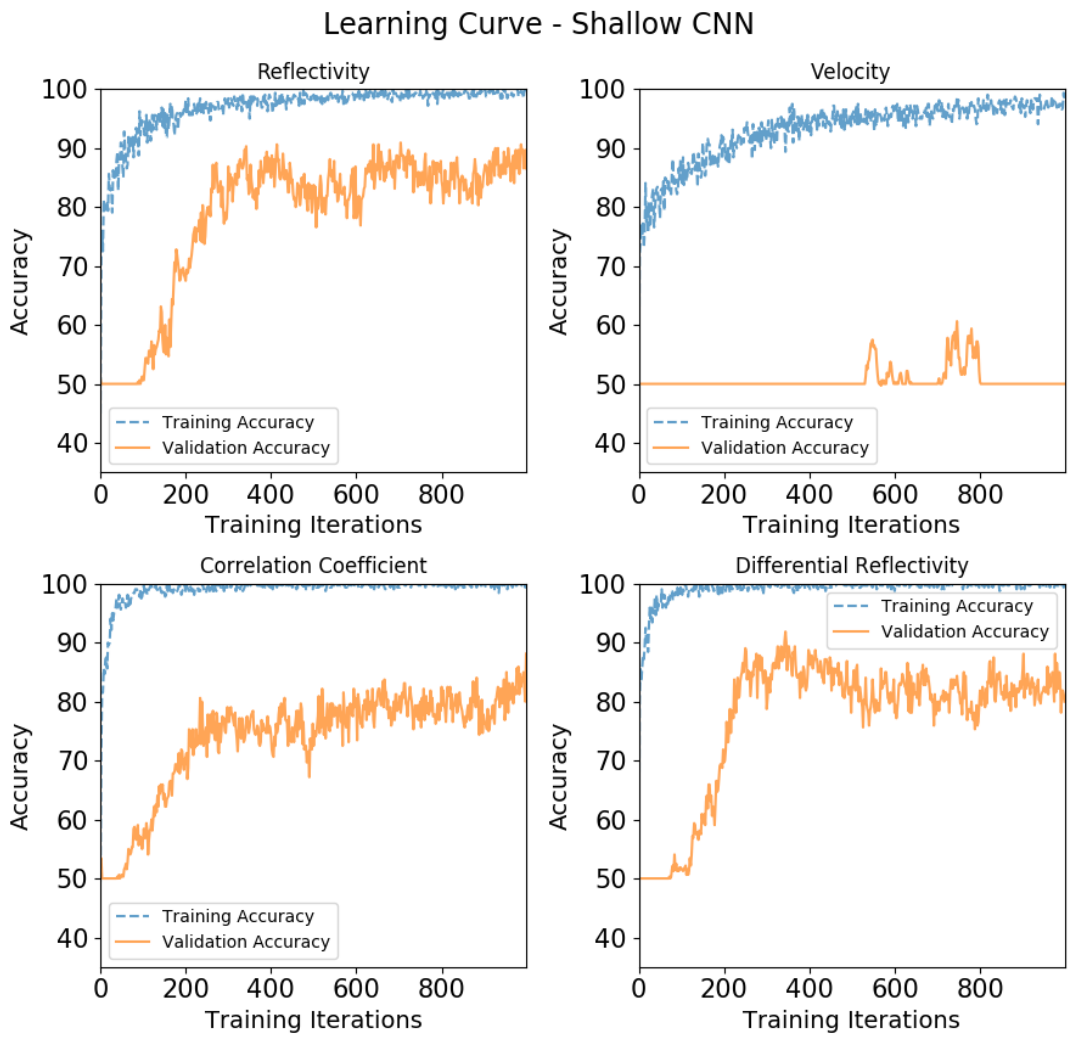


Figure 5.5: Learning Curve for training the shallow CNN network. Average learning curve from five runs.

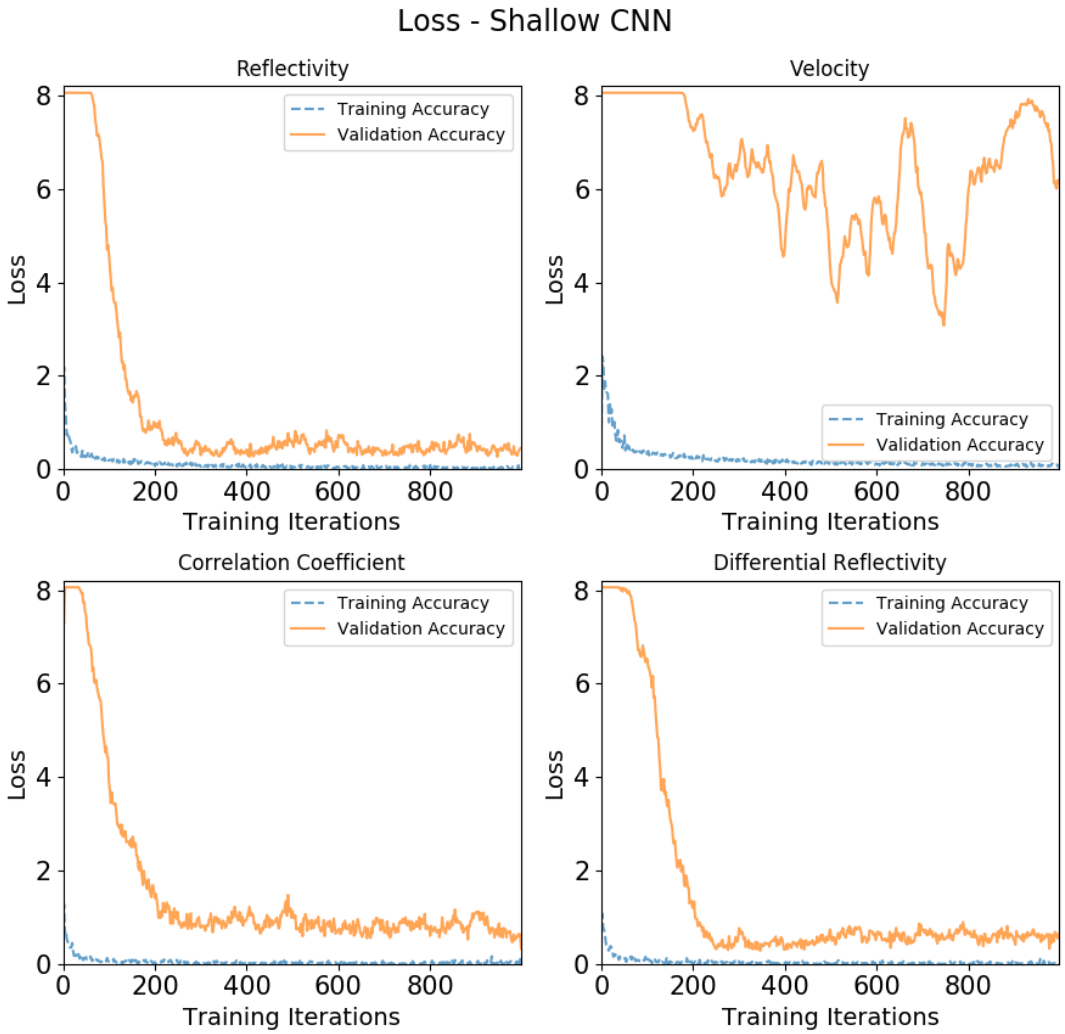
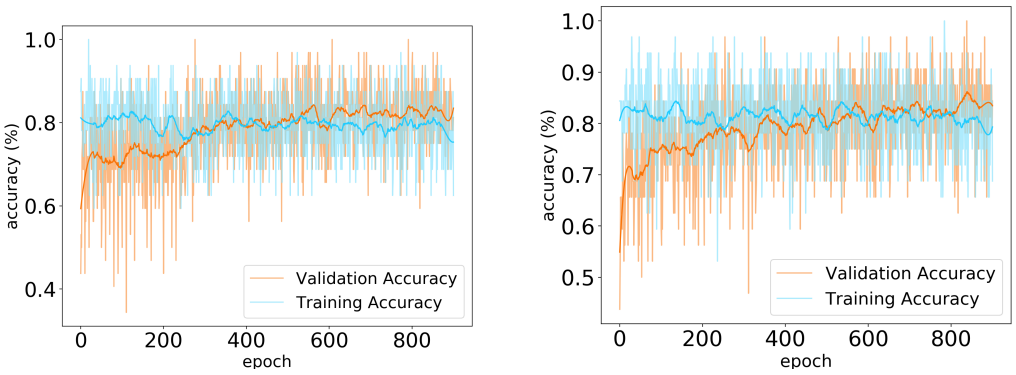
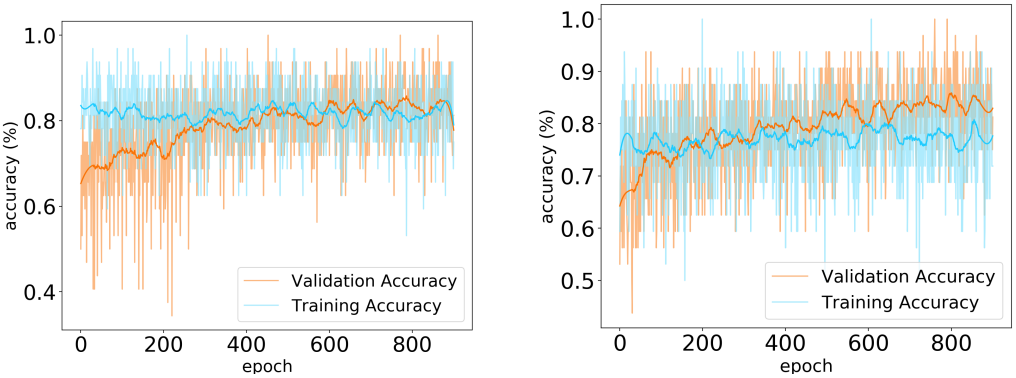


Figure 5.6: Loss for training the shallow CNN network. Average learning curve from five runs.



(a) Reflectivity

(b) Velocity



(c) ρ_{HV}

(d) Z_{DR}

Figure 5.7: Learning curve for the ANN trained on four individual radar products: Reflectivity, Velocity, ρ_{HV}, and Z_{DR}. These learning curves are each from a single run. Results from Avery (2018)

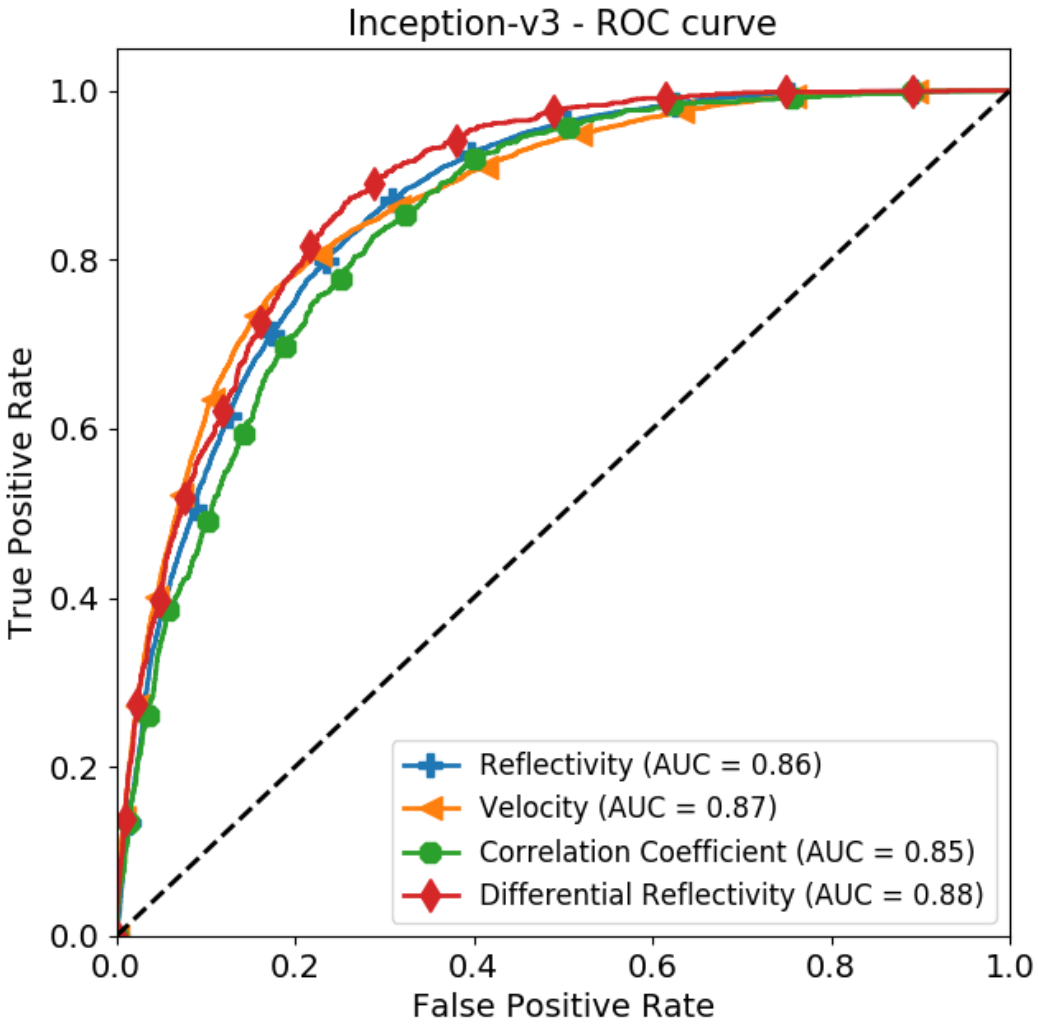


Figure 5.8: ROC curve for the inception net. Four different shallow CNN networks were trained on Reflectivity, Velocity, ρ_{HV}, and Z_{DR} separately.

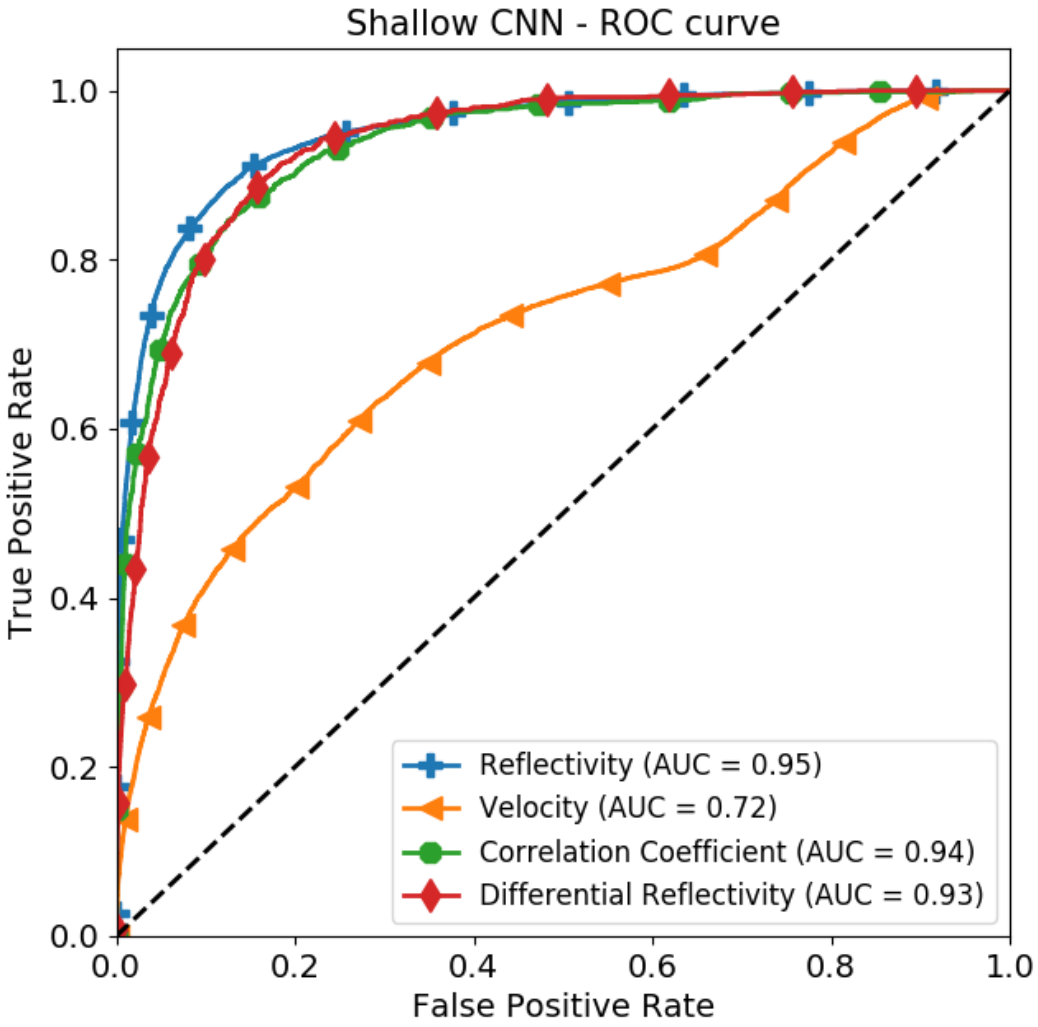


Figure 5.9: ROC curve for the shallow CNN. Four different shallow CNN networks were trained on Reflectivity, Velocity, ρ_{HV}, and Z_{DR} separately.

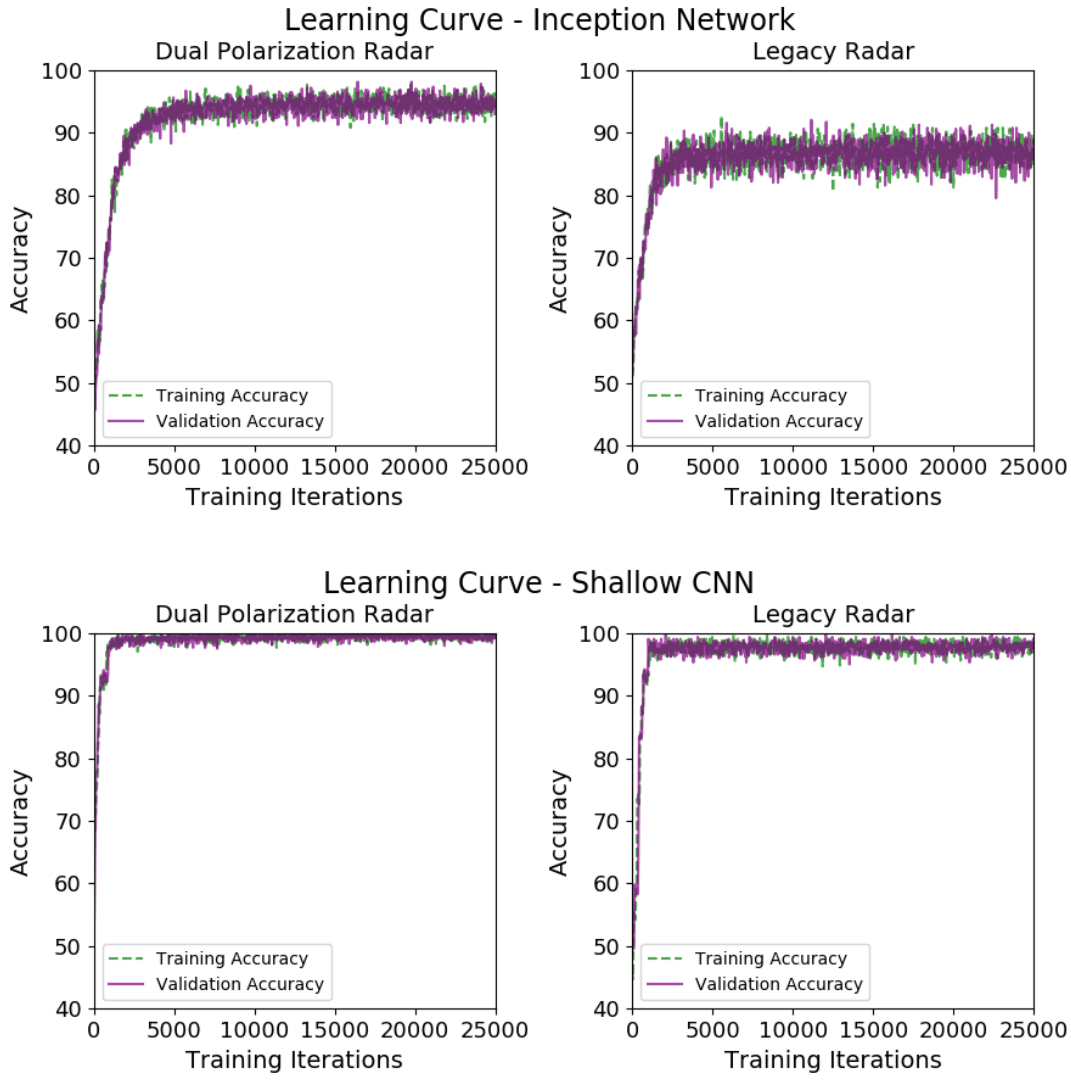


Figure 5.10: Learning curve when training on the inception network and shallow CNN probability that an image contains a roost given four individual radar products: Reflectivity, Velocity, ρ_{HV}, and Z_{DR} .

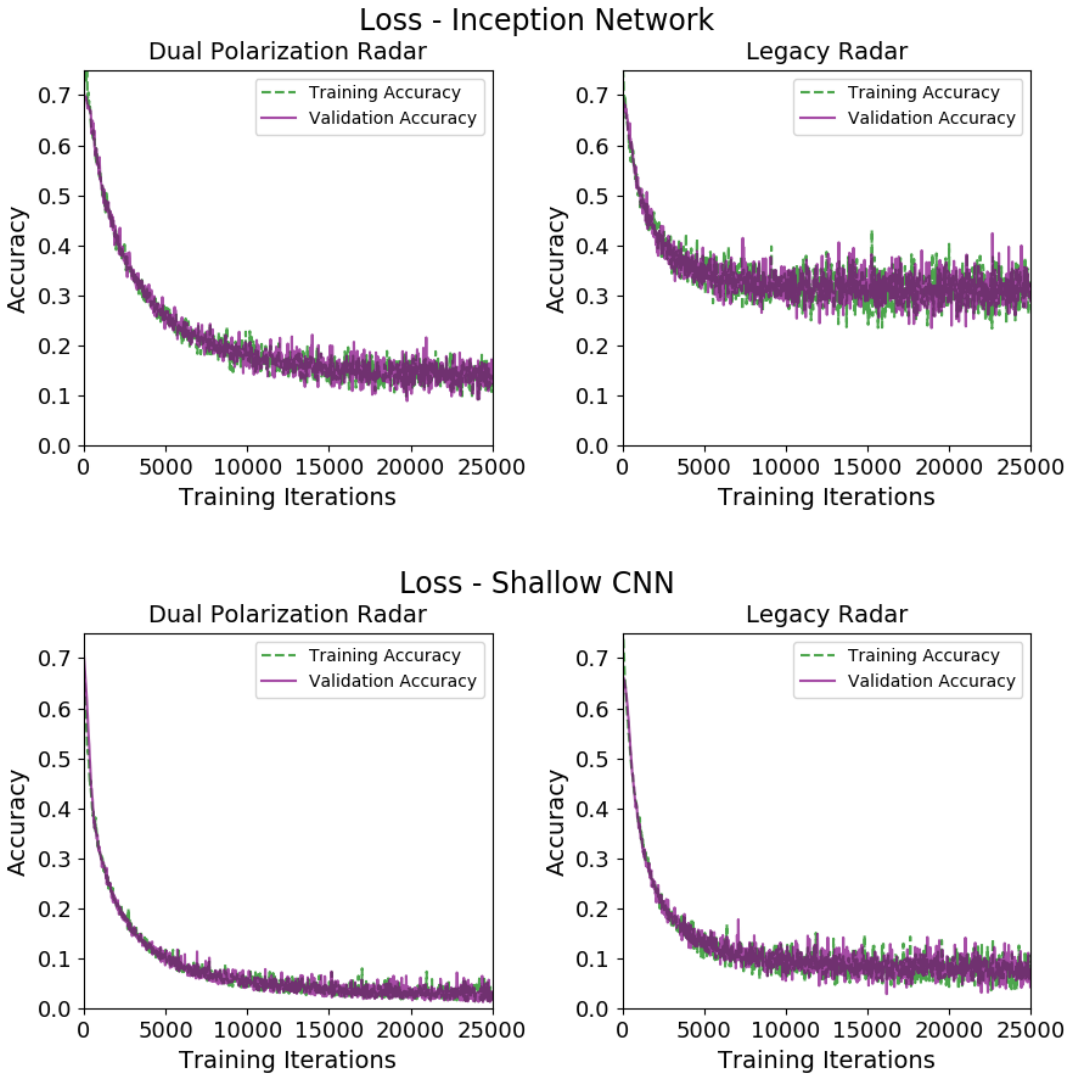


Figure 5.11: Loss when training on the inception network and shallow CNN probability that an image contains a roost given four individual radar products: Reflectivity, Velocity, ρ_{HV}, and Z_{DR} .

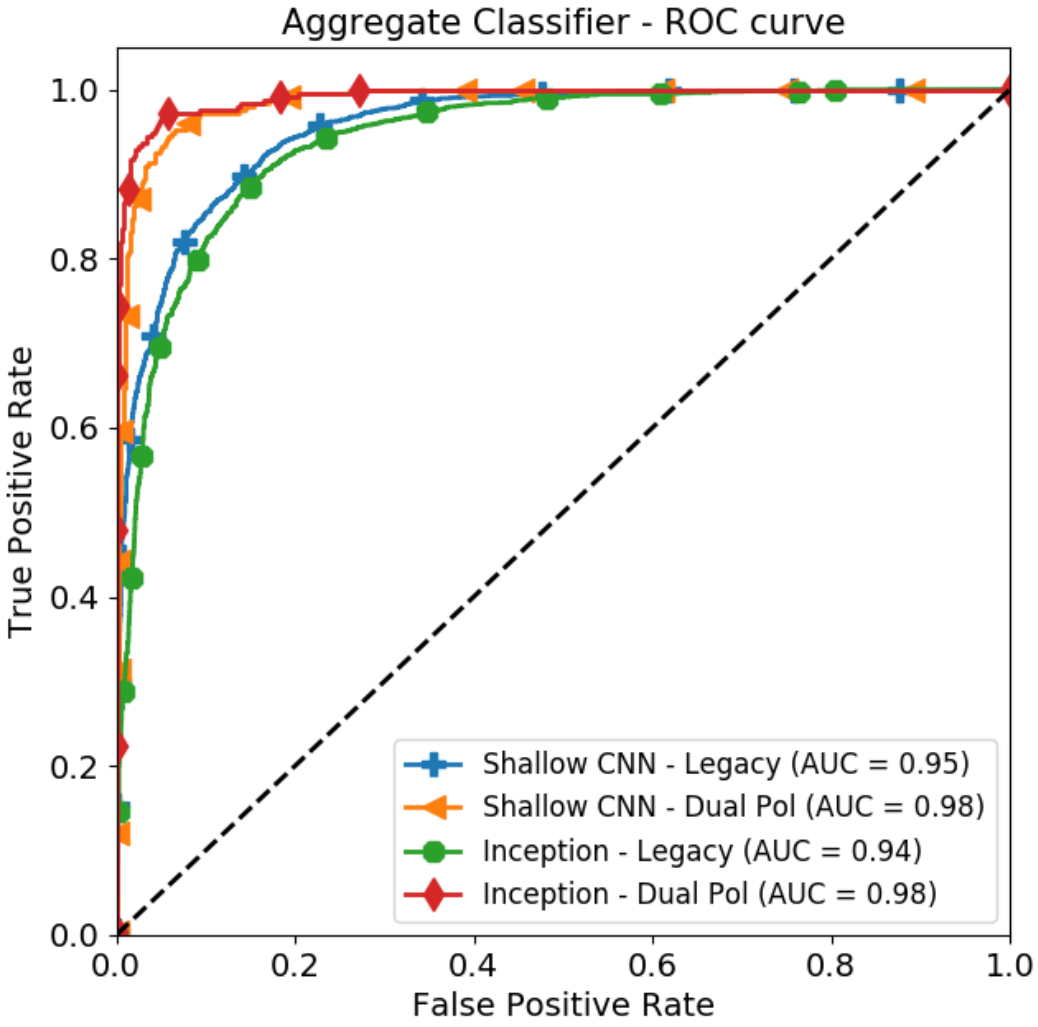


Figure 5.12: Final ROC curve results for the dual-pol and legacy data. Our machine learning model was trained on probability outputs from Reflectivity, Velocity, ρ_{HV}, and Z_{DR}

Chapter 6

Conclusions and Future Work

My contributions to this project are: 1) the architecture of the shallow CNN and 2) evaluating how well both a shallow CNN trained from scratch and a deep CNN trained using transfer learning perform when classifying bird roost. I apply existing start of the art image classification techniques to automated bird roost detection in radar data.

My classification method vastly reduces the number of images that need to be manually searched in order to find bird roosts, especially since most radar images don't contain visible bird roosts. Both the Inception-v3 and the shallow CNN models performed well and are the two best models. Eliminating the final false positives from the dataset by hand will be much less time consuming that sifting through all 70,000 radar images a year, searching for bird roosts. I am able to reduce the amount of time it takes to process radar image data and I believe these results can be improved in the future with a temporal analysis of the data and more dual polarization labels. I am planning on making these results publicly available to citizen scientists in the Spring of 2018.

In the future, we hope to fully automate the bird roost detection process by preprocessing the radar images. Just as radar data is quality controlled for weather, we could filter out weather from our radar data to eliminate some of the noise from our radar data. Additionally, biology generally falls within a range of reflectivity values (-10 dBZs to 10 dBZs) (Koistinen 2000), and by filtering out

values outside of this range we can eliminate some of the noise. We can not use a reflectivity filter to fully determine where birds are since light drizzle and insects are often detected in this range as well (Koistinen 2000). Biological scatter will likely have a high differential reflectivity and a low correlation coefficient, and we use this to further filter and clean the data (Van Den Broeke 2013). Some of these radar quality control filters or a combination of several of them may help increase our bird detection accuracy.

We are also not currently taking advantage of the temporal component of the data during learning, which is a key component when it comes to roost detection. The expanding roost rings over several radar snapshots are the signature used to detect bird roost in radars. Long Short Term Memory networks have been used on sequences of images, for example to re-identifying a person over disjoint cameras (Wu et al. 2016) or to detect the type of activity (run, jump, etc.) a person is performing in a video (Yeung et al. 2015). The catch with Convolutional LSTM networks is that they take longer to train than Convolutional Neural Networks and can sometimes require more data.

Our results could be improved with additional Dual Polarization Labels. As stated above, CNNs require lots of data to train properly. Our dual polarization radar results were better than our legacy radar results even though we had fewer dual polarization machine learning inputs. Hand classifying roost data is a time consuming process, however, it would be useful for better automated roost detection. One of the advantages of polarimeter radar for weather is that it helps quality control the biology more accurately from the weather data. It stands to reason that the same method that is used to remove biology from the radar data can be used to find it as well.

Figure 6.1: Our results show that convolutional neural nets can identify bird roosts in radar imagery, however there is still work to be done. After all, we haven't had five years and a research team to automate bird roost detections. Image from https://xkcd.com/1425/.

Reference List

Avery, K., 2018: *Automated Detection of Bird Roosts Using NEXRAD Radar Data and Artificial Neural Networks*. (undergraduate honors thesis), in preparation.

Bauer, S., J. W. Chapman, D. R. Reynolds, J. A. Alves, A. M. Dokter, M. M. Menz, N. Sapir, M. Ciach, L. B. Pettersson, J. F. Kelly, et al., 2017: From agricultural benefits to aviation safety: Realizing the potential of continent-wide radar networks. *BioScience*, **67**, 912–918.

Bauer, S. and B. J. Hoye, 2014: Migratory animals couple biodiversity and ecosystem functioning worldwide. *Science*, **344**, 1242552.

Bridge, E. S., S. M. Pletschet, T. Fagin, P. B. Chilson, K. G. Horton, K. R. Broadfoot, and J. F. Kelly, 2016: Persistence and habitat associations of purple martin roosts quantified via weather surveillance radar. *Landscape ecology*, **31**, 43–53.

Canziani, A., A. Paszke, and E. Culurciello, 2016: An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

Chilson, P. B., E. Bridge, W. F. Frick, J. W. Chapman, and J. F. Kelly, 2012a: Radar aeroecology: exploring the movements of aerial fauna through radio-wave remote sensing.

Chilson, P. B., W. F. Frick, J. F. Kelly, K. W. Howard, R. P. Larkin, R. H. Diehl, J. K. Westbrook, T. A. Kelly, and T. H. Kunz, 2012b: Partly cloudy with a chance of migration: weather, radars, and aeroecology. *Bulletin of the American Meteorological Society*, **93**, 669–686.

Dayhoff, J. E. and J. M. DeLeo, 2001: Artificial neural networks. *Cancer*, **91**, 1615–1635.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, 2009: Imagenet: A large-scale hierarchical image database. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 248–255.

DeVault, T. L., B. F. Blackwell, and J. L. Belant, 2013: *Wildlife in airport environments: preventing animal–aircraft collisions through science-based management*. JHU Press.

Diehl, R. H. and R. P. Larkin, 2005: Introduction to the WSR-88D (NEXRAD) for ornithological research.

Dieleman, S., K. W. Willett, and J. Dambre, 2015: Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, **450**, 1441–1459.

Efron, B. and R. Tibshirani, 1986: Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 54–75.

Fawcett, T., 2006: An introduction to roc analysis. *Pattern recognition letters*, **27**, 861–874.

Gauthreaux Jr, S. A. and C. G. Belser, 1998: Displays of bird movements on the wsr-88d: patterns and quantification. *Weather and Forecasting*, **13**, 453–464.

—, 2003: Radar ornithology and biological conservation. *The Auk*, **120**, 266–277.

Glorot, X. and Y. Bengio, 2010: Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Goodfellow, I., Y. Bengio, and A. Courville, 2016: *Deep Learning*. MIT Press, `http://www.deeplearningbook.org`.

Helmus, J. and S. Collis, 2016: The python arm radar toolkit (py-art), a library for working with weather radar data in the python programming language. *Journal of Open Research Software*, **4**.

Ioffe, S. and C. Szegedy, 2015: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 448–456.

Jain, A. K., J. Mao, and K. M. Mohiuddin, 1996: Artificial neural networks: A tutorial. *Computer*, **29**, 31–44.

Johnson, G. D., W. P. Erickson, M. D. Strickland, M. F. Shepherd, D. A. Shepherd, and S. A. Sarappo, 2002: Collision mortality of local and migrant birds at a large-scale wind-power development on buffalo ridge, minnesota. *Wildlife Society Bulletin*, 879–887.

Karpathy, A., 2016: Convolutional neural networks for visual recognition. URL `http://cs231n.github.io/convolutional-networks/`

Kelly, J. F. and S. M. Pletschet, 2017: Accuracy of swallow roost locations assigned using weather surveillance radar. *Remote Sensing in Ecology and Conservation*.

Kelly, J. F., J. R. Shipley, P. B. Chilson, K. W. Howard, W. F. Frick, and T. H. Kunz, 2012: Quantifying animal phenology in the aerosphere at a continental scale using NEXRAD weather radars. *Ecosphere*, **3**, 1–9.

Kingma, D. and J. Ba, 2014: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kohavi, R. et al., 1995: A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, Stanford, CA, volume 14, 1137–1145.

Koistinen, J., 2000: Bird migration patterns on weather radars. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere*, **25**, 1185–1193.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012: Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097–1105.

Kunz, T. H., S. A. Gauthreaux Jr, N. I. Hristov, J. W. Horn, G. Jones, E. K. Kalko, R. P. Larkin, G. F. McCracken, S. M. Swartz, R. B. Srygley, et al., 2008: Aeroecology: probing and modeling the aerosphere. *Integrative and comparative biology*, **48**, 1–11.

Laughlin, A. J., D. R. Sheldon, D. W. Winkler, and C. M. Taylor, 2014: Behavioral drivers of communal roosting in a songbird: a combined theoretical and empirical approach. *Behavioral Ecology*, **25**, 734–743.

Lawrence, S., C. L. Giles, A. C. Tsoi, and A. D. Back, 1997: Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, **8**, 98–113.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, 1989: Backpropagation applied to handwritten zip code recognition. *Neural computation*, **1**, 541–551.

Li, F.-F., A. Karpathy, and J. Johnson, 2016: Lecture 4: Backpropagation and neural networks part 1.

Li, M., T. Zhang, Y. Chen, and A. J. Smola, 2014: Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 661–670.

Lin, M., Q. Chen, and S. Yan, 2013: Network in network. *arXiv preprint arXiv:1312.4400*.

Mitchell, T. M., 1997: *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1st edition.

Muller, B. M., F. R. Mosher, C. G. Herbster, and A. T. Brickhouse, 2015: Aviation bird hazard in NEXRAD dual polarization weather radar confirmed by visual observations. *International Journal of Aviation, Aeronautics, and Aerospace*, **2**, 6.

Oquab, M., L. Bottou, I. Laptev, and J. Sivic, 2014: Learning and transferring mid-level image representations using convolutional neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1717–1724.

Raudys, Š., 1998: Evolution and generalization of a single neurone: I. single-layer perceptron as seven statistical classifiers. *Neural Networks*, **11**, 283–296.

RoyChowdhury, A., D. Sheldon, S. Maji, and E. Learned-Miller, 2016: Distinguishing weather phenomena from bird migration patterns in radar imagery. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 10–17.

Rumelhart, D. E., G. E. Hintont, and R. J. Williams, 1986: Learning representations by back-propagating errors. *NATURE*, **323**, 9.

Shin, H.-C., H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, 2016: Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, **35**, 1285–1298.

Shipley, J. R., J. F. Kelly, and W. F. Frick, 2017: Toward integrating citizen science and radar data for migrant bird conservation. *Remote Sensing in Ecology and Conservation*.

Shlens, J., 2016: Train your own image classifier with inception in tensorflow. URL `https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html`

Simonyan, K. and A. Zisserman, 2014: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014: Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, **15**, 1929–1958.

Stepanian, P. M. and K. G. Horton, 2015: Extracting migrant flight orientation profiles using polarimetric radar. *IEEE Transactions on Geoscience and Remote Sensing*, **53**, 6518–6528.

Stepanian, P. M., K. G. Horton, V. M. Melnikov, D. S. Zrnić, and S. A. Gauthreaux, 2016: Dual-polarization radar products for biological applications. *Ecosphere*, **7**.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2015: Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, 2016: Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Troxel, S., M. Isaminger, B. Karl, M. Weber, and A. Levy, 2001: Designing a terminal area bird detection and monitoring system based on ASR-9 data.

Tudor Ionescu, R., B. Alexe, M. Leordeanu, M. Popescu, D. P. Papadopoulos, and V. Ferrari, 2016: How hard can it be? estimating the difficulty of visual search in an image. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2157–2166.

Van Den Broeke, M. S., 2013: Polarimetric radar observations of biological scatterers in hurricanes irene (2011) and sandy (2012). *Journal of Atmospheric and Oceanic Technology*, **30**, 2754–2767.

Weber, P., T. J. Nohara, and S. Gauthreaux Jr, 2005: Affordable, real-time, 3-d avian radar networks for centralized north american bird advisory systems. *2005 Bird Strike Committee-USA/Canada 7th Annual Meeting, Vancouver, BC*, 7.

Werbos, P. J., 1990: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, **78**, 1550–1560.

Wilson, D. R. and T. R. Martinez, 2003: The general inefficiency of batch training for gradient descent learning. *Neural Networks*, **16**, 1429–1451.

Wu, L., C. Shen, and A. van den Hengel, 2016: Convolutional lstm networks for video-based person re-identification. *arXiv preprint arXiv:1606.01609*.

Yeung, S., O. Russakovsky, N. Jin, M. Andriluka, G. Mori, and L. Fei-Fei, 2015: Every moment counts: Dense detailed labeling of actions in complex videos. *International Journal of Computer Vision*, 1–15.

Yue-Hei Ng, J., F. Yang, and L. S. Davis, 2015: Exploiting local features from deep networks for image retrieval. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 53–61.

Zaugg, S., G. Saporta, E. Van Loon, H. Schmaljohann, and F. Liechti, 2008: Automatic identification of bird targets with radar via patterns produced by wing flapping. *Journal of the Royal Society interface*, **5**, 1041–1053.

Zhou, Y. and R. Chellappa, 1988: Computation of optical flow using a neural network. *IEEE International Conference on Neural Networks*, volume 1998, 71–78.