8-27-2020

Artificial intelligence for helicopter safety: Head pose estimation in the cockpit

Eric William Feuerstein
Rowan University

ARTIFICIAL INTELLIGENCE FOR HELICOPTER SAFETY: HEAD POSE ESTIMATION IN THE COCKPIT

by

Eric W. Feuerstein

A Thesis

Submitted to the
Department of Electrical and Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical and Computer Engineering
at
Rowan University
June 1, 2020

Thesis Chair: Ravi P. Ramachandran, Ph.D.

Acknowledgements

I would like to express my appreciation to Dr. Ravi Ramachandran for the opportunity to continue my education and for his continued positivity and aid while conducting this research. I would also like to thank Dr. Nidhal Bouaynaya, Dr. Ghulam Rasool, and Ph.D. student Hikmat Khan for their excess of knowledge and for their guidance throughout this research.

Thank you to my family and friends for their continued support during this time and for pushing me to be the best I can be.

This research was supported by the Federal Aviation Administration (FAA) Cooperative Agreement Number 16-G-015 and NSF Awards OAC-2008690 and DUE-1610911.

Abstract

Eric W. Feuerstein
ARTIFICIAL INTELLIGENCE FOR HELICOPTER SAFETY:
HEAD POSE ESTIMATION IN THE COCKPIT
2019-2020
Ravi P. Ramachandran. Ph.D.
Master of Science in Electrical and Computer Engineering

The recent impact of deep learning algorithms and their major breakthroughs on various aspects of our lives has led to the idea to investigate the application of these algorithms in different problem spaces. One of the novel areas of investigation is the aviation and air traffic control domain; as it offers a prime opportunity to enhance safety within the aviation community. Of particular importance to this community is improving the safety of rotorcraft operations, as this segment of the aviation industry is subject to a higher fatal accident rate than other segments of the industry. The improvement of safety for rotorcraft also directly improves the safety and efficiency of air traffic control, since rotorcraft operate primarily within low-level airspace; an area that is becoming increasingly complex with new entrants such as unmanned aircraft systems, urban air mobility, etc..

The novel method for improving rotorcraft safety, and the main topic of this research, is to create an algorithm that determines the head position of helicopter pilots and copilots through automatic post-processing of onboard flight video data. This information can then be used to aid in incident/crash analysis as well as future vision systems research. Both a classical computer vision technique and a deep learning approach were taken to provide possible solutions to this problem. Both solutions successfully deal with the issues of excessive cockpit background, extreme head positions, and added noise from the pilot's operational equipment which include helmets, microphones, and sunglasses.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

Figure Page

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

List of Tables

Chapter 1

Introduction

1.1 Statement of the Problem

The main objective of this work is to help improve the overall safety of rotorcraft operations by creating an algorithm that can determine pilot and copilot head positions using only flight video data collected from onboard cameras. The Federal Aviation Administration (FAA) continues to promote and highlight the importance of participating in aviation Flight Data Monitoring (FDM) programs. These programs are intended to improve flight safety and operational efficiency, and recorder safety was one of the topics on the agency's Top 10 Most Wanted List of Safety Improvements in 2017-2018 [1]. The FAA, National Transportation Safety Board (NTSB), and the United States Helicopter Safety Team (USHST) as well as other industry partners are working together to implement helicopter safety enhancements that promote the use of flight data recorders (FDR) to reduce the fatal accident rate in rotorcraft operations.

Although there is a need to integrate more FDRs into the rotorcraft community, certain obstacles still exist. The initial cost of FDRs can range from $9,000 - $50,000 which does not include the cost to utilize them as part of an overall FDM program [2]. These costs alone play a significant role in preventing FDM programs from being adopted by small operators. On top of that, these devices can require technical expertise and special reading devices or software. Due to those reasons, rotorcraft in general, typically have a lower participation rate in FDM programs than other forms of aviation (i.e. commercial fixed-wing or Part 121 airline operations) [3].

On the other hand, even small helicopter operators often have the financial means to purchase one or more off-the-shelf video cameras which can be mounted inside the helicopter cockpit. These cameras, when pointed at the instrument panel, offer an alternate method of collecting the same data as a traditional FDR by utilizing post-processing of cockpit videos. Onboard video data also offers several possibilities for improving rotorcraft safety such as flight replay and the ability to extract information from where the pilot and copilot were focusing their attention during critical phases of flight. The crash survivability of the data being collected is also increased, because video information can be stored remotely.

This area of research also considers the obstacles faced when pilots transition from flying in Visual Meteorological Conditions (VMC) to Instrument Meteorological Conditions (IMC). VMC refers to clear visual scenarios where the pilot is able to fly by looking directly out the window and using visual references, while IMC refers to cloudy or obscure flight conditions where the pilot must fly using only the information presented on the instrument panel in front of them. Flying in IMC sometimes results in loss of control due to phenomena such as spatial disorientation, and usually result in fatal accidents. These events can often be traced back to improper instrument flight rule (IFR) scanning techniques or attention tunneling where the pilot fixates on a particular instrument at the expense of other instruments offering needed information. The rate of this happening can be reduced however, with better pilot training and with the added use of technologies such as enhanced/synthetic vision and heads-up displays. Gathering information about where the pilots are looking during critical phases of flight can be crucial in improving overall rotorcraft safety through better training, and for researching which new techniques and

technologies allow for them to focus more of their attention outside the cockpit, rather than down at the instrument panel.

1.2 Motivation

The motivation of this research is to create a low-cost method using a combination of computer vision and deep learning techniques to determine the head position of helicopter pilots and copilots given onboard cockpit videos. Even in cases where a helicopter is equipped with an FDR, an accident investigator may not know what the pilot or copilot was focusing on during the moments leading up to or during an incident or crash. Cockpit video offers the ability to understand not only the aircraft state but also the pilot and copilot's actions in a potentially unsafe situation. Therefore, the goal is to automate post-processing of flight video data which will provide safety analysts or accident investigators with information on where a pilot was focused during any particular moment of any given flight. Admittedly, without the proper governance, this type of information could be used inappropriately by rotorcraft operators. However, the policies regarding the use of this information, while an important topic in its own regard, are outside the scope of this thesis.

Before the initial implementation of the head pose estimation algorithm, the problem of gaze estimation for helicopter pilots was considered. This estimation technique looks at the eyes of the test subject in the videos and is able to determine exactly where the subject is looking at any given time. However, it is commonly found that pilots wear sunglasses or tinted face shields during their flights, and for that reason the eyes of the pilots are very frequently occluded from the camera's point of view. As a substitute for gaze estimation, head pose estimation was selected as the next best choice for solving the

problem at hand. While the exact location that the pilot is looking will be unknown due to the sunglasses, the general direction of gaze can be estimated using a head pose estimation technique.

Head pose estimation is a well-researched computer vision topic and it is a solved problem when it comes to dealing with clean, frontal, passport-type photos. However, the challenge of identifying the head position at extreme angles with added noise and excessive background information is still a topic of discussion in the computer vision community. In most real world test videos supplied by the FAA, the pilots were looking at extreme angles and wearing helmets, sunglasses, microphones and other equipment that obstructed the camera's view of their face. Both a hybrid computer vision algorithm and a purely deep learning algorithm were created to classify the head positions of the pilots despite these additional obstacles. The computer vision algorithm presented in this thesis classifies the head positions of the pilots into three main classes: (0) straight out the window, (1) down at the instrument panel, and (2) out the window to the side. The deep learning algorithm is capable of classifying the head positions into one of nine classes, allowing for a more fine-tuned head pose estimation: (0) Down, (1) Down_Left, (2) Down_Right, (3) Left, (4) Right, (5) Straight, (6) Up, (7) Up_Left, and (8) Up_Right. This information will be used for incident/crash analysis, future vision systems research at the FAA, and for improving the overall safety of rotorcraft operations.

1.3 Thesis Objectives

The objectives of this thesis are:

1. Create a low cost method for accurately determining the head positions of helicopter pilots and copilots by utilizing post-processing of cockpit video data.

2. Explore the possibility of implementing a classical computer vision algorithm that does not require labeled ground truth data to be available.
3. Create a sufficiently large, labeled ground truth dataset that consists of images of helicopter pilots and copilots with various head positions.
4. Train multiple deep learning models for determining head positions of helicopter pilots and copilots using the labeled ground truth dataset.
5. Discuss the advantages and disadvantages of the purely deep learning solution.

1.4 Thesis Focus and Organization

The focus of this thesis is to discuss two algorithms that can accurately determine the head position of helicopter pilots and copilots given supplied flight video data. The first algorithm utilizes both computer vision and deep learning techniques and the second algorithm uses a purely deep learning approach.

The first chapter provides an introduction to the problem and the motivation for creating a head pose estimation algorithm.

The second chapter discusses the technical background knowledge needed to understand the computer vision and deep learning techniques used in this thesis. The topics of face detection, facial landmark annotation, and the pinhole camera model are covered. General machine learning concepts are discussed, and a more detailed explanation of each network architecture used in this thesis is included as well.

The third chapter explains the approach and methodology behind the initial creation of the hybrid algorithm. It also contains a description of the process for training and evaluating the different deep learning models as well as the final structure of the deep

learning algorithm. A comprehensive discussion is also included to provide details on each of the datasets that were used throughout this thesis.

The fourth chapter contains the results of both the hybrid computer vision algorithm and each of the deep learning models. The benefits and limitations of each proposed solution are discussed as well.

The fifth and final chapter provides a summary of the overall results and accomplishments of this thesis. A brief discussion of future improvements is also included in this section.

Chapter 2

Background

This chapter contains a complete review of all the technical aspects that are relevant to the work presented in this thesis. Two solutions to the problem of head pose estimation have been suggested in this thesis: a hybrid computer vision technique that incorporates deep learning components, and a purely deep learning approach.

The process of face detection as well as facial landmark extraction is covered. A discussion is included that defines the coordinate systems and pinhole camera model that were used in the hybrid computer vision algorithm. The calculation of Euler angles from a camera's extrinsic rotation matrix is also discussed.

A general overview of deep learning network design is included. A number of terms are defined to sufficiently explain the layout of basic neural network architectures. The importance of data collection and data distributions is also highlighted in this section. The process of hyperparameter selection for training a network using a loss function and forward/backward propagation is discussed. Methods for improving the generalization of networks and a brief discussion of convolutional networks is included as well.

Three network architectures were used primarily in this thesis to train the majority of the purely deep learning head pose estimation models. The concepts of Residual networks, Inception networks, and Xception networks are all discussed in brief, to emphasize their novel enhancements in the field of deep learning.

2.1 Hybrid Computer Vision Algorithm

The first proposed algorithm for solving the problem of head pose estimation of helicopter pilots uses a mixture of computer vision techniques and deep learning based detectors. The algorithm has three main steps: deep learning based face detection, semi-automatic facial landmark annotation, and angle calculation for classification. There are a number of head pose estimation techniques that already exist for frontal facing images but few that deal with heads at extreme angles [4]. Additionally, the only input available to the algorithm is a video of helicopter pilots during flight. Therefore, more complex head pose estimation techniques that utilize depth camera information or person specific template models could not be considered [5].

2.1.1 Face detection. In order to estimate the head position of a pilot from a video or image, the location of the subject's head in each frame must be found. Face detection is a solved problem in most cases and there are a number of publicly available face detectors that use a variety of methods to perform detection. For the algorithm discussed in this thesis, two face detectors from the OpenCV library were considered.

The first face detector that was examined uses a histogram of oriented gradients (HoG) classifier for detection. The first step to create this type of classifier is to extract HoG descriptors, or features, from a set of training images. Simply put, the vertical and horizontal gradients of an image are calculated, the image is split into cells, and block normalization is applied to reduce the influence of lighting changes. The information from the block normalization is then used to create a one-dimensional feature vector to describe the entire image. A Linear Support Vector Machine (SVM) is then trained to distinctly classify images for face detection [6][7].

The second detector utilizes deep learning techniques and was trained using a Single Shot Detector (SSD) framework with a Residual Network (ResNet) base. The base network is responsible for extracting simple features from the image (edges, shapes, colors). The SSD is created by adding additional convolutional layers to the output of the base network. This allows the network to perform object detection for multiple objects in a single image both quickly and accurately. The architecture of the network as well as the model weights are publicly available to download on the OpenCV website as a text file and a caffe model file [8].

2.1.2 Facial landmark annotation. A facial landmark, also referred to as a facial feature, can be defined as the localization of vital key points on the face. Common facial landmarks include the tip of the nose, corners of the eyes or mouth, and the jawline. Automatic facial point detection plays a very important role in face analysis, and has been a focus in computer vision for more than two decades.

The facial landmark predictor that is considered in this thesis uses a semi-automatic methodology for facial landmark annotation. It defines a 68 point annotation model that is consistent with the MultiPIE database as shown in Figure 1.

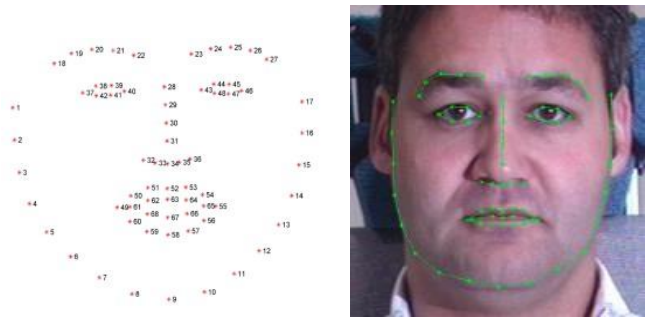

Figure 1. Layout of 68 facial landmark annotations

The semi-automatic annotation tool makes use of a type of Active Appearance Model (AAM) known as the Active Orientation Model (AOM). These models are known to generalize well to unseen variations which make them suitable to use in a scenario where facial landmark annotations are unknown.

The semi-automatic annotation method begins by training an AOM using a known subset of annotated images. This trained AOM is then applied to a new subset of images where the annotations are unknown and the results are used as an initialization to fit the trained AOM to the new subset. The results from the fitting procedure are then classified manually as "good" or "bad" according to the paper. The good images are removed from the subset and the AOM is continuously fit to the subset of bad images until all the images have been removed from the initial subset. The annotations are then manually corrected and can be made more accurate by building a person specific model for each subject [9].

This tool was used to annotate the 300-W dataset which was created as part of the *300 Faces in-the-Wild Challenge* in 2013. This challenge was the first facial landmark localization challenge which in turn, created a standard by which automatic facial landmark annotation methods could be measured. This challenge tasked competitors to automatically annotate both 300 indoor and 300 outdoor images of in-the-wild subjects downloaded from the internet. However, it is important to note that 87% of the 600 test images have subjects with a horizontal pose variation angle between ±15° and the most 'extreme' angle considered is ±30°. Also, 70% of the test images are non-occluded images [10].

The facial landmark annotation tool that was considered for the first algorithm presented in this thesis, trained an AOM using the 300-W dataset. It performs well even in the case of occlusion and provides accurate annotations at frontal poses. However, its

accuracy drastically decreases as the head pose angle becomes more extreme. There are very few, if any, annotated databases that consider truly extreme angles greater than ±30°.

2.1.3 Pinhole camera model. The pinhole camera model is one of the most commonly referred to models in the field of computer vision and it serves as the primary step of the hybrid algorithm. This model is used as a way to map a set of 3D object coordinates to a set of 2D image coordinates.

An image is recorded on a film by placing a barrier with a single opening between the object and the film. This hole is known as the aperture and in the ideal pinhole camera model, this opening is assumed to be a single point. The aperture prevents the film from being exposed to all angles of light rays emitted by a point on the object. Limiting the amount of light that passes through to the film allows for a mathematical relationship to be defined. The purpose of the aperture is demonstrated in Figure 2.

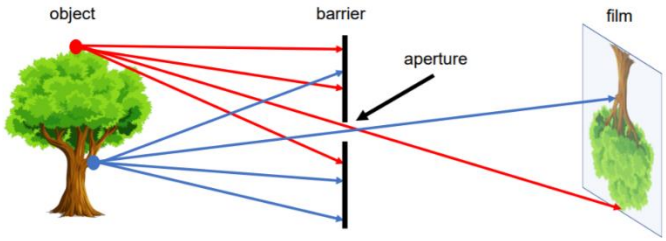

Figure 2. Simple camera model with aperture in place

To mathematically define the relationship between the 3D coordinates and the 2D coordinates, two coordinate systems are defined. The first coordinate system defined is the camera coordinate system $[i\ j\ k]$ where the origin is at the center of the aperture and the k axis is perpendicular with, and pointing towards the film. The second coordinate system is

the standard world coordinate system [*X Y Z*] where the *X* axis corresponds with the *j* axis, the *Y* axis corresponds with the *i* axis, and the *Z* axis corresponds with the *k* axis but points towards the object rather than the film. The camera coordinate system is depicted below, in Figure 3.

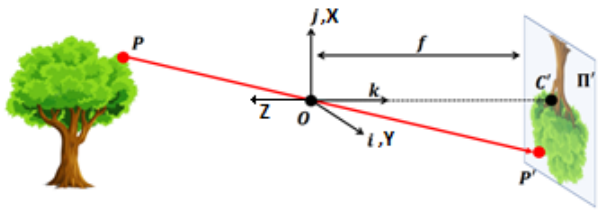

Figure 3. Camera coordinate system

From this point forward, the film will be referred to as the image plane. In Figure 3 above, the point *P* is in 3D world coordinates *[X Y Z]* and the point *P'* is the result of the 3D points being projected onto the 2D image plane. The focal length of the camera *f*, is defined as the distance between the pinhole and the image plane, and the point *O* is the origin of the camera coordinate system. The line created between the origin *O*, and the projected point *C'* is what is known as the optical axis of the camera. Drawing a line along the optical axis from the object to the image plane allows for a geometrical relationship to be derived using the law of similar triangles as demonstrated in Figure 4.

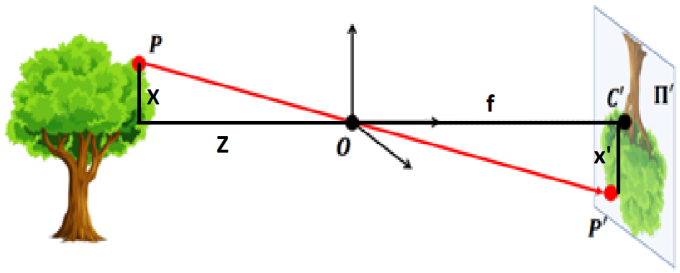

Figure 4. Demonstration of triangles formed by the optical axis

The law of similar triangles states that if two triangles are the same shape, but not necessarily the same size, the lengths of the sides of each triangle are proportional. This results in Equation 2.1:

$$\frac{X}{x'} = \frac{Y}{y'} = \frac{Z}{f}$$

(2.1)

If the position of the 3D world coordinates are known, Equation 2.1 can be rewritten to solve for the projected 2D image coordinates:

$$x' = f\frac{X}{Z} \ , \ y' = f\frac{Y}{Z}$$

(2.2)

Equation 2.1 and Equation 2.2 demonstrate how a set of world coordinates are transferred to the image plane in an ideal pinhole camera model [11][12].

2.1.3.1 Internal camera matrix. Equations 2.1 and 2.2 above map the relationship between a set of 3D world coordinates to a set of 2D image coordinates in an ideal situation. However, in the real world things are not this simple. In order for the Euclidean geometry from the world coordinate system to properly transfer to the projected space on the image plane, the coordinates need to be converted to homogeneous coordinates. Homogeneous

coordinates allow for N-dimensional coordinates to be represented with N+1 numbers and can easily be converted by adding one extra variable to the Equation 2.2 [13]. It is easier to visualize the homogeneous form of Equation 2.2 in matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(2.3)

Equation 2.3 now shows the basic method for calculating the projection of a set of 3D points onto the 2D image plane in matrix form. Oftentimes the world coordinates are given in real world measurements such as meters, whereas the image plane coordinates are given in pixels. For that reason, a scale factor is introduced to the focal length variable. Equation 2.3 can then be rewritten as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(2.4)

In Equation 2.4, f_x and f_y are simply the focal length multiplied by a scale factor s_x and s_y respectively.

The next detail to discuss is the difference between the origin in the camera coordinate system and the origin of an image. The camera coordinate system defines its origin in the center of the aperture, while the image origin is often in the top left corner of the image. To adjust for this translation, two more variables are added to Equation 2.4.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(2.5)

In Equation 2.5 above, c_x and c_y account for the translation of the origin in the x direction and y direction respectively. This new equation now considers the conversion from real world measurements to pixel measurements, as well as the difference between camera and pixel origins. The matrix *K* is defined as the internal camera matrix. In the context of this research, this representation of the camera matrix in Equation 2.6 was sufficient without the need to also add a skew factor [14].

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim K \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(2.6)

2.1.3.2 External camera matrix. The final aspect of the pinhole camera to discuss is the external or extrinsic camera matrix. Thus far, the model discussed assumes that the 3D coordinates and 2D coordinates share a similar coordinate system centered on the optical axis. However, in practice this is rarely the case. The external camera matrix is now defined to provide information about the rotation and translation of the world coordinate system with respect to the camera coordinate system.

The external camera matrix in simplest form is a 3x3 rotation matrix concatenated with a 3x1 translation vector. This form displayed in Equation 2.7, covers all degrees of freedom for rotating the coordinate system, as well as all three directions that the coordinate system can be translated.

$$[R \mid \boldsymbol{t}] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} \mid & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}$$

(2.7)

A row of (0,0,0,1) is commonly added to the bottom of this matrix both to satisfy matrix dimensions when multiplying, and to allow for the decomposition of the single matrix into a translation followed by a rotation.

$$\begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} I & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} x \begin{bmatrix} R & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(2.8)

Equation 2.8 is sufficient to define the external camera parameters and describe the rotation and translation of the camera coordinate system with respect to the world coordinate system [14]. The final pinhole camera model used for the context of this research is shown in Equation 2.9 below.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim K \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(2.9)

In order to estimate the head position of the pilots, the suggested computer vision algorithm seeks to calculate the external camera matrix with respect to a manually defined 3D reference frame.

2.1.4 Euler angle calculations. The information from the rotation matrix within the external camera matrix can be used to calculate Euler angles of pitch, yaw, and roll for

classification of pilot head poses. The derivation of calculating Euler angles from a rotation matrix is discussed.

A rotation matrix has three degrees of freedom and the standard definition of the rotations about these three principle axes is shown in Equation 2.10.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(2.10)

These three matrices can be multiplied together to get one general rotation matrix, *R*.

$$R = R_z(\gamma)R_y(\beta)R_z(\alpha)$$

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma - \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma - \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}$$

(2.11)

From the expanded version of the rotation matrix in Equation 2.11, each Euler angle α, β, and γ can be derived. Rearranging the matrix in Equation 2.11 for element R_{31} leads to Equation 2.12.

$$R_{31} = -\sin\beta$$

$$\beta = -\sin^{-1} R_{31}$$

(2.12)

The expanded matrix in Equation 2.11 is used to solve for α using equation 2.13:

$$\frac{R_{32}}{R_{33}} = \frac{\sin\alpha\cos\beta}{\cos\alpha\cos\beta} = \frac{\sin\alpha}{\cos\alpha} = \tan\alpha$$

$$\alpha = \tan^{-1}(R_{32}, R_{33})$$

(2.13)

A similar method is used to calculate γ in Equation 2.14:

$$\frac{R_{21}}{R_{11}} = \frac{\cos\beta\sin\gamma}{\cos\beta\cos\gamma} = \frac{\sin\gamma}{\cos\gamma} = \tan\gamma$$

$$\gamma = \tan^{-1}(R_{21}, R_{11})$$

(2.14)

Equations 2.12, 2.13, and 2.14 will be utilized by the algorithm to convert the calculated rotation matrix to three Euler angles: pitch (α), yaw (β), and roll (γ) [15].

2.2 Deep Learning Algorithm

A number of different deep learning techniques were used to solve the problem of head pose estimation of helicopter pilots. The necessary knowledge needed to understand the general concepts of deep learning are discussed in the following sections.

2.2.1 Neural network overview. One of the most common uses of neural networks is to model a relationship between a set of inputs and a set of outputs. In its simplest form, a two-layer neural network consists of an input layer, one hidden layer, and an output layer. The input layer does not count towards the total number of layers. These layers consist of multiple neurons which are tasked with learning how specific features of the input correlate

to the output. The early layers of a more complicated network will learn basic features of the inputs and as the network gets deeper, and more layers are added, each layer will learn a more complex feature that is specific to the input.

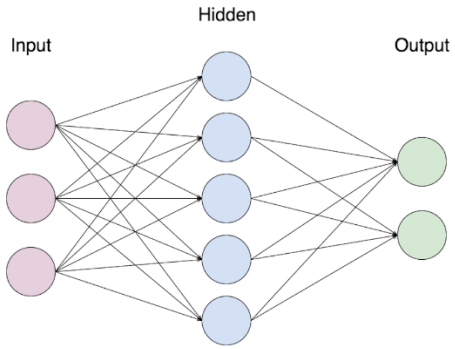

Figure 5. Basic two-layer neural network architecture

The basic building blocks that neural networks use to define the input-output relationship are a set of weights and biases, and a set of non-linear activation functions. Figure 5 above depicts two fully connected layers, where each neuron of the previous layer is connected to each neuron in the following layer. Each of these inputs is multiplied by a weight, and a certain bias value is added as well. The weights and biases are commonly referred to as learnable parameters. The objective of any network is to learn which weights and biases will provide a generalizable relationship between the inputs and the outputs. To calculate the value of each neuron, Equation 2.15 is used:

$$z = w^{\tau}x + b$$

(2.15)

In Equation 2.15, x is a vector of the inputs, w is a transpose vector of the weights for one neuron, and b is a vector of the bias for that neuron. Equation 2.15 is used to

calculate the value of each neuron in a layer, and this process is repeated for each hidden layer of the network until the output layer is reached. This process defines a linear relationship between the inputs and the outputs.

In more complex problem spaces where the number of inputs may be in the thousands or millions, the relationship between the inputs and outputs will never be linear. For that reason, a non-linearity known as the activation function is applied to each neuron. This allows for a more complex relationship between inputs and outputs to be defined. One of the most common activation functions, and the one used primarily in this thesis, is the Rectified Linear Unit (ReLU). The output of each neuron with this added non-linearity becomes:

$$a = \max(0, z)$$

(2.16)

Equation 2.16 says that the value of the neuron will either be set to zero if z is less than or equal to zero, or the value will not change if z is greater than zero. By forcing negative values to zero, this creates a sparser model which can provide a more accurate prediction at lower computational cost.

The output layer of a neural network will often have a different activation function than ReLU. Two common activation functions that are used in the output layer are the sigmoid function and the softmax function. The sigmoid function is used in binary classification problems and forces the output to be either zero or one. The softmax function is used in multiclass problems, and can turn the value of multiple outputs into a set of probabilities that sum to one. The class index with the highest probability is then selected as the output of the network [16].

2.2.2 Model training. A deep learning network utilizes different optimization algorithms to find the best weights and biases that produce a generalizable relationship between the input and the output. In order to perform training, a sufficiently large set of input data is needed where the true output of each input sample is known. The network learns through a repeated process of forward and backward propagation where each input sample is passed through the network, the total loss of the system is calculated, and the weights and biases are updated with respect to that loss.

For the process of forward propagation, a loss function is defined. There are a number of loss functions that can be used depending on the problem space. One commonly used loss function is cross entropy loss:

$$\mathcal{L}(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

(2.17)

In Equation 2.17 above, $\hat{y}$ is the predicted class label from the network and y is the true class label of the input sample. Cross entropy loss increases as the probability of the predicted label diverges farther from the true label. This instills greater penalties on predictions that are much different than their true value. Equation 2.17 defines the cross entropy loss for a binary classification problem, but the equation can be easily adapted to a multiclass problem by calculating a separate loss for each class label and summing the results. This loss is calculated for each sample through the training process and the total loss across all samples is defined in Equation 2.18 as the cost function:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

(2.18)

The cost function is a function of the weights and biases of the network. The value of the cost function is calculated after all the training samples have been passed through the network once. In order to find the optimal weights and biases, this cost function must be minimized by updating the weights and biases to reduce loss.

The weights and biases are updated through the process of backward propagation. The most commonly used optimization algorithm for updating the weights and biases is gradient descent. A variety of optimization algorithms have been developed including Momentum, RMSProp, and Adam but each of these algorithms are built from the general principal of gradient descent. The update rule for gradient descent calls for each weight and bias to be updated based on the derivative of the cost function with respect to each individual weight and bias.

$$w := w - \alpha \frac{dJ(w,b)}{dw}$$

$$b := b - \alpha \frac{dJ(w,b)}{db}$$

(2.19)

In Equation 2.19 above, α is the learning rate of the algorithm. The learning rate is one of the most important hyperparameters to consider when training a deep learning model. Hyperparameters are defined by the user before training, and unlike weights and biases these parameters are not learned by the network during training. This value is selected by the user and determines how much the weights and biases will be updated during backward propagation. A large value of α results in faster learning, but can prevent the network from converging to a minimum, while a small value of α increases training

time and therefore increases computational cost. The exact learning rate that will provide the best results will be different for each problem space.

Each cycle of forward propagation and backward propagation is defined as one epoch. The number of epochs that an algorithm will take to converge is often unknown so certain stopping criteria must be defined. Stopping criteria can include: stopping after a specific number of epochs, stopping when the loss reaches a certain value, or stopping early if the validation accuracy stops increasing. The stopping criteria that will produce an accurate model while remaining computationally efficient will be unknown but will vary depending on the problem space.

Once the algorithm has finished training, the weights and biases are frozen so that predictions can be obtained on a real world test set. The test set will consist of input samples that the network has not seen during the training process [16].

2.2.3 Data distributions. The most important part of any deep learning algorithm is the data. In order for a network to learn, it must be supplied with a known subset of data where the input and the output are known. These datasets need to be sufficiently large to generalize all possible inputs and outputs of the problem that is trying to be solved.

The entire labeled dataset is generally split into a training set, validation set, and test set. The training set is always the largest and can range anywhere from 60-98% of the total amount of labeled data. This data is used to help the network learn the optimal weights and biases needed to map the inputs to the outputs. The validation and test sets are split evenly, based on the remaining percentage of labeled data that was not used in the training set. The validation set is passed through the network at the end of each forward pass to help measure how well the network is generalizing to an unknown dataset. The test set is used

to evaluate the network's overall performance after training is completed. The network does not actually learn and update its parameters based on the validation and test sets, but these sets do provide important information on how the network will perform in real world scenarios after training.

Two common problems that occur when training a neural network are underfitting and overfitting. Underfitting occurs when the algorithm is unable to achieve a high training accuracy, meaning it cannot learn the input-output relationship. This can be fixed by adding more training data or increasing the size and depth of the network so it can learn more complex features of the training data. Overfitting occurs when the training accuracy is very high but the network performs poorly on the validation and test sets. This is often caused when the network memorizes specific noise in the training set and is unable to generalize well to an unknown dataset. This can be fixed by adding more data, trying a different network architecture, or including regularization techniques which will be discussed briefly in the next section.

It is important that the data be distributed evenly as well. The number of labeled examples should be similar across all classes and the validation and test sets must also be of the same distribution as the training set. Without the proper organization of the labeled data, any network will be unable to produce accurate results [16].

2.2.4 Improving neural networks. The most common ways to increase the generalizability of a network and reduce overfitting is to add regularization. There a vast number of regularization techniques that are used in the deep learning community but only the ones used in this thesis are discussed [17].

2.2.4.1 L2 regularization. L2 regularization is a method that seeks to push all the network weights as close to zero as possible. This method adds an additional term to the cost function of the network:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \lambda\sum_{i=1}^{m} w_i^2$$

(2.20)

The second term in Equation 2.20 is the L2 norm of the weights multiplied by a user defined scalar, λ. The value of λ will affect how much the regularization term will contribute to the total cost. A high value of λ will force the weights closer to zero because the regularization term will have a greater effect on the total cost of the network. The squared values of the weights in the L2 norm will cause this term to get very large when weights have large values, and will help put more focus on these weights to be minimized. L2 regularization helps to reduce overfitting because it prevents the network from relying too heavily on certain neurons and allows the knowledge of all features to influence the output. Weights with lower values also make the network simpler, which decreases computational cost and makes the network more robust and generalizable.

2.4.4.2 Image normalization. Networks that have images as their inputs generally undergo the process of normalization. Normalizing pixel values is done by dividing each pixel by 255 since this is the maximum value a pixel can have. This forces all pixel values to be in the range of 0-1. The purpose of this is to make the network more robust to changes in intensity value and to lower the total computational cost.

2.4.4.3 Dropout regularization. Dropout regularization allows the network to share its knowledge across all neurons in a layer, rather than focusing on just a few. The dropout

process effectively “turns off” a certain percentage of neurons in each layer of the network. This concept is displayed in Figure 6.

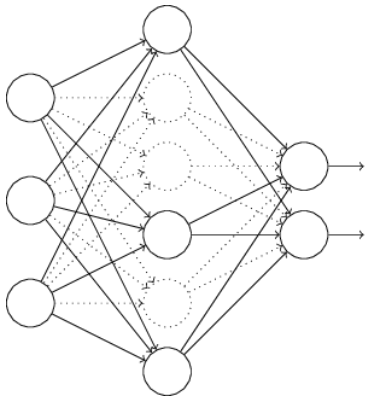

Figure 6. Dropout regularization

The neurons that are turned off are randomly selected and will be different for each training example and each forward pass of the network. This process works to reduce overfitting because it prevents the network from becoming too reliable on a single feature of the input.

2.4.4.4 Transfer learning. Transfer learning is the process of using the weights and biases from a network trained on one problem, and applying this knowledge to train a new network on a different but related problem. If two problems are similar enough, most of the low level features of the inputs will be very similar as well. By initializing a new network with the weights and biases from a pre-trained network, this prevents the network from needing to learn these lower level features again and allows the new network to immediately start learning more complex features of the input. The process of transfer learning is useful for cutting down on training time and for generalizing a new network when more data becomes available in the same problem space.

2.2.5 Convolutional neural networks. One very specialized area of deep learning, and a topic of great research and discussion in the community, is the area of image processing. Convolutional neural networks (CNNs) were created with images in mind, and make use of three types of layers in order to train accurate networks on image input data. All CNNs follow a similar architecture scheme as shown in Figure 7.

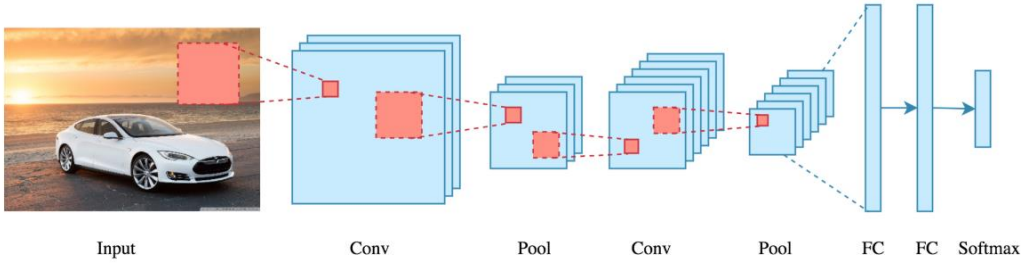

Figure 7. Basic layers in a convolutional neural network

As with any other type of neural network, the goal of a CNN is to learn specific features of the input, and to learn how these features relate to the true output. A CNN learns the features of an image by using filters or kernels. Filters are commonly 3x3, 5x5, or 7x7 pixels in size, and are used in other areas of image processing for feature extraction, edge detection, or identifying shapes or patterns in an image. Each filter in a CNN is used to learn a different feature of the input. Examples of a vertical and horizontal edge filter are displayed in Figure 8.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Figure 8. Horizontal and vertical edge filters

A convolutional layer in a CNN uses different filters and the process of convolution to create a feature map that is a representation of the original image. The process of convolution involves sliding each filter over the input image so that it is in every location one time. At every filter location, element-wise matrix multiplication is performed between the filter pixel values and the image pixel values, and the results are summed. This sum is then transferred to a feature map which becomes a representation of how well that specific feature was represented in different parts of the image [18].

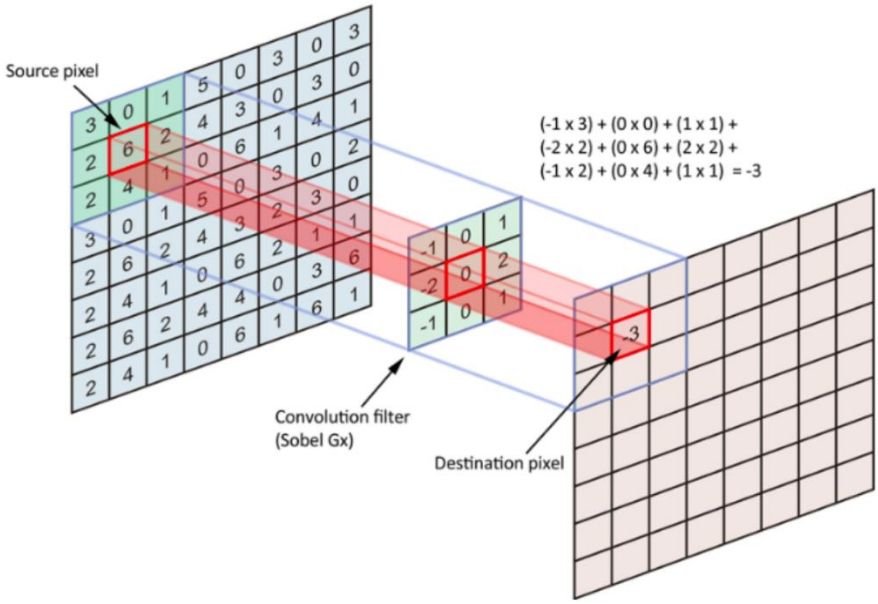

Figure 9. Example of one convolutional step

The process shown in Figure 9 is repeated for multiple filters in order to create an output volume. It is important to note that the filter width must be equal to the number of channels in the input image. Therefore if the input to the network is a color image with three channels, the filter must also be three channels wide, as shown in Figure 10.

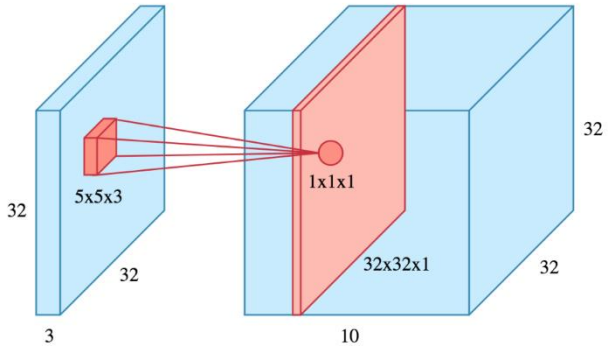

Figure 10. One convolution operation and its feature map

As with other types of neural networks, a non-linear activation function such as ReLU is applied to the feature map to allow for a more complex relationship to be defined between the inputs and the outputs.

The second layer that is utilized by a CNN is a pooling layer. Many types of pooling exist but max and average pooling seem to be most commonly used in practice. Pooling is a similar process to convolution in the sense that a window is slid across each position of an image or feature map. However, instead of doing an element-wise matrix multiplication, max pooling will take the maximum value from the window and move it to the next layer. Average pooling will take the average of the entire window and move that to the next layer. Pooling is commonly done with a 2x2 window and a stride of two as depicted in Figure 11.

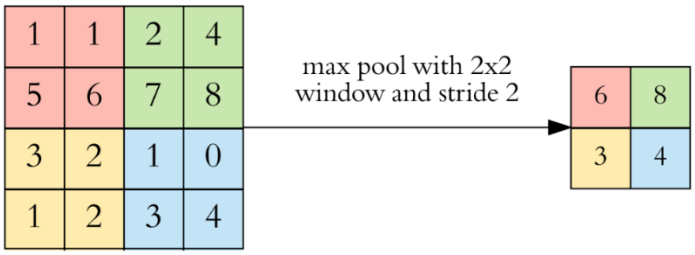

Figure 11. Max pooling

Pooling layers are used to reduce the dimensions of the input images. This in turn, reduces the total number of learnable parameters, which shortens the overall training time and the possibility of overfitting. Pooling layers effect each layer of the volume independently and don't reduce the depth of the volume, only the height and width.

The final layers of a CNN are fully connected layers. In order to move from a volume to a fully connected layer, CNNs flatten their three-dimensional layers to a single one-dimensional vector. It is common that a few fully connected layers be implemented at the end of a CNN before the actual output layer. These layers act exactly the same as in other networks where input values are multiplied by a weight, added to a bias, and then passed through a non-linear activation function.

CNNs are trained with the same process of forward and backward propagation as discussed in section 2.2.2, although there are additional aspects that come along with it. The detailed knowledge of forward and backward propagation of CNNs is not required to understand the concepts discussed in this thesis [19].

2.2.6 Specific network architectures. A total of three different network architectures were considered in depth throughout this research, each of which explore a

novel idea in the area of deep learning. A high-level discussion of each is included in this section.

2.2.6.1 Residual blocks. In 2015 the concept of residual networks (ResNets) was introduced. It has already been stated that network depth is of crucial importance because it allows for more complex features of the input to be learned. However, as a network becomes deeper by adding more layers, the issue of vanishing gradients begins to limit network performance. The gradient begins to get infinitely small during backward propagation which effects the process of gradient descent when updating weights and biases. ResNets work by introducing a shortcut connection or what is more commonly referred to as a skip connection to the network architecture. The skip connection moves the information from a previous layer and injects it to a deeper layer.

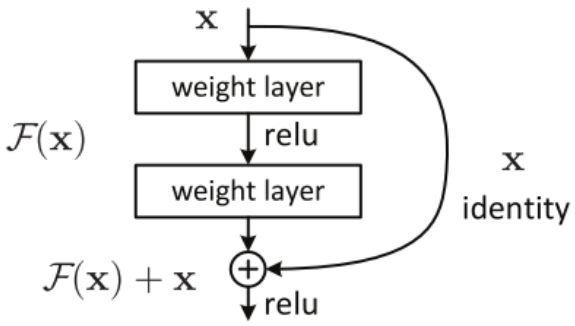

Figure 12. Residual block

In the residual block shown in Figure 12 above, the activation *x* is added to the activation of *F(x)* a few layers deeper, and this sum is passed through the ReLU non-linearity. The skip connection helps to mitigate the issue of vanishing gradients because even if a lot of information is lost in *F(x)*, the information from *x* is still present in the layer.

These residual blocks can allow for a network's architecture to become 56, 100, or even 1202 layers deep without seeing a dramatic decrease in performance. Prior to the addition of skip connections, the deepest network architectures at the time were limited to 19 or 22 layers.

Different ResNet depths have been tested on the ImageNet dataset, a publicly available dataset which has become the standard for measuring the performance of network architectures over the past few years. A ResNet architecture with 34 layers on the ImageNet dataset had a training error of 7.76% compared to an error of 10.02% on a plain 34 layer network without skip connections [20][21].

2.2.6.2 Inception modules. In 2014, the concept of the inception module was introduced to enhance convolutional neural networks. Prior to the inception module, one of the hyperparameters that needed to be decided manually by the user was which filter size to use. Rather than selecting only one filter size by hand, the inception module computes a 1x1, 3x3, and 5x5 convolution as well as a 3x3 max pooling. Performing a 3x3 and 5x5 convolution can become computationally expensive rather quickly so dimension reduction is included by first using a 1x1 convolution, before performing the more computationally expensive convolutions. These 1x1 convolutions also include the use of ReLU activation functions which add an additional non-linearity to the process. An inception module with and without dimension reduction is shown in Figure 13.

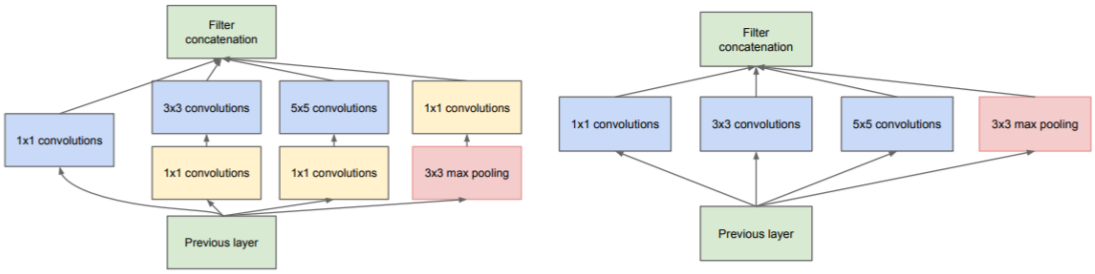

Figure 13. Inception Modules with and without dimension reduction

The outputs of each of the convolutions and pooling are concatenated together to create a single output volume which becomes the input to the next layer in the network. The benefit of the inception module is that it allows the network to learn what filter sizes and pooling will have the greatest effect in improving the model's accuracy. The network can decide what filter size is needed at a certain layer in the network and removes the need for a filter size to be selected by the user [22].

2.2.6.3 Xception architecture. The Xception architecture is modelled after the Inception framework and is based entirely on depthwise separable convolutional layers like the one shown in Figure 14. A depthwise separable convolution consists of a spatial convolution performed independently over each channel of an input, followed by a 1x1 convolution to change the dimensions. Traditional convolution in other CNNs is performed across all channels of a volume at once, but this is not necessary with the inclusion of the depthwise separable convolutions. This decreases the number of connections and therefore the learnable parameters of the model.

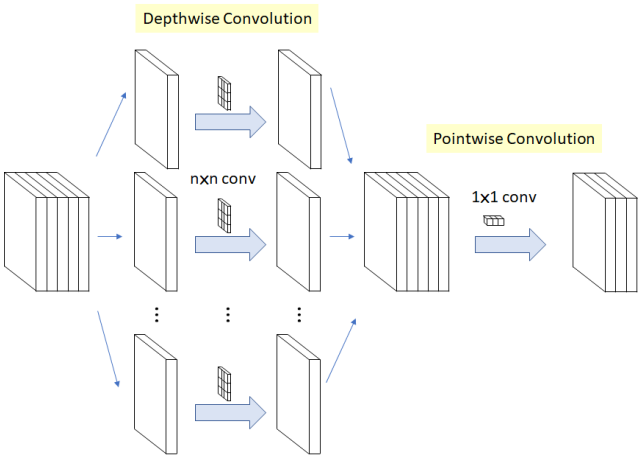

Figure 14. Depthwise separable convolution

The full Xception network architecture is a linear stack of 36 depthwise separable convolution layers with skip connections. By including skip connections the Xception architecture is able to take advantage of both the ideas presented in the ResNet and Inception networks. By combining these two concepts with the depthwise separable convolutional layers, the Xception network was able to outperform both the 152 layer ResNet and the Inception Network on the ImageNet dataset [23][24].

Chapter 3

Approach and Methodology

Chapter 3 will explain the approach and methodology for creating the hybrid computer vision algorithm and the purely deep learning algorithm. A description of the datasets used for evaluating the algorithms, and the procedure for creating the FAA Simulator Dataset will be discussed. The method used to obtain optimal parameters for the hybrid algorithm is included. This chapter will also cover the method for training and optimizing the deep learning models and the structure of the final deep learning algorithm.

3.1 Datasets

A total of four datasets were used throughout the course of this research. These datasets were used to validate certain aspects of the hybrid algorithm, and to train the deep learning models. They were also used to calculate a final quantifiable accuracy of both algorithms.

3.1.1 Head pose image database. The first dataset considered was a publicly available benchmark dataset used to validate the accuracy of the hybrid algorithm. This dataset consists of fifteen subjects with ninety-three images corresponding to each subject, for a total of 2790 monocular face images. A few images showing two different subjects from this dataset are displayed in Figure 15.

Figure 15. Sample images from the Head Pose Image Database

Each image in the dataset has a corresponding pitch and yaw angle in degrees that serves as the ground truth for the image. The (vertical) pitch angle ranges from ±60° and the (horizontal) yaw angle falls within the range of ±90°. An image with a pitch and yaw label of 0° corresponds to the test subject looking straight forward at the camera [25].

3.1.2 Synthetic dataset. The second dataset was created manually and consists of a series of synthetic test videos which display a subject moving their head at extreme angles. Two videos were created, one clean video with no added noise and one video with the subject wearing sunglasses, shown in Figure 16. These videos were created to simulate the conditions of the helicopter pilot test videos. This dataset was used heavily for the validation and experimentation of different aspects of the hybrid algorithm.

Figure 16. Sample images from the Synthetic Dataset

3.1.3 FAA flight dataset. The third dataset was created using one of the real world flight videos provided by the FAA. A sample frame from the flight test video is shown in Figure 17.

Figure 17. Sample image from the FAA Flight Dataset

This video consisted of 30976 copilot images of which the first 10,000 images were manually labeled so a quantifiable accuracy could be calculated on both the hybrid and deep learning algorithms. Each image was given a class label between zero and eight. These labels are displayed in Table 1 and Figure 18.

Table 1

Class labels for FAA Flight Dataset

Class Name	**Class Label**
Down	0
Down_Left	1
Down_Right	2
Left	3
Right	4
Straight	5
Up	6
Up_Left	7
Up_Right	8

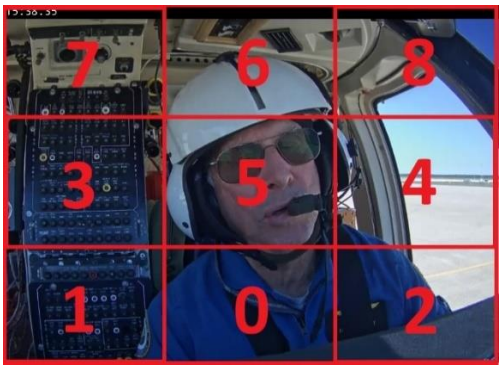

Figure 18. Class label grid

The naming convention was kept alphabetical, and these labels are from the point of view of the camera and not the point of view of the copilot.

This manually-labeled dataset was primarily used to get an approximate accuracy of both algorithms so the results could be compared. It is important to note that because the data was labeled manually, the ground truth is somewhat subjective. This stems from the fact that there is no way to set explicit boundaries to determine the exact time when a subject's head belongs to each class. That being said, this real world dataset was created more to prove concepts of the algorithms and to compare the results of both methods, rather than to quantify the true overall accuracy of the algorithm.

3.1.4 FAA simulator dataset. The final dataset was created after the completion of the hybrid head pose algorithm, and was used to train the deep learning models. The creation of this data was an extremely important step in this research and required a lot of fine tuning to ensure that a large amount of data could be collected quickly, while making sure that the data simulated the real head positions of helicopter pilots during an actual flight. This data was created with the same nine classes listed in Table 1.

3.1.4.1 First data collection. Once the number of classes were defined, data was collected on two separate occasions. During the first collection, there were 5 test subjects of various heights, genders, and ethnicities. These test subjects were asked to sit in both the pilot and copilot seats of a Sikorsky S76D helicopter located in the hangar at the William J. Hughes Technical Center (WJHTC) in Egg Harbor Township, NJ. An Axis S2016 NVR was used to record constant video of the subjects during each of the test runs. Each test run consisted of the subject holding their head in each of the nine classes for a total of thirty seconds each, while including slight variations in their head movement. Each subject did this on the pilot side of the cockpit as well as on the copilot side.

A total of six test runs were conducted for each subject to better simulate the various equipment that pilot's often wear during flights. In each of the six test runs, the subjects were wearing a different combination of equipment. The equipment that was worn during each of the test runs are displayed in Table 2 and Figure 19.

Table 2

Equipment worn during each test run

Test Run	**Equipment Combination**
Test 1	Headset only
Test 2	Headset and sunglasses
Test 3	Helmet only
Test 4	Helmet and sunglasses
Test 5	Helmet with clear visor
Test 6	Helmet with dark visor

Figure 19. Examples of equipment worn in each test run

These six runs were done from the pilot's and copilot's seats for a total of twelve test runs per test subject. The time stamp for the start and end times of each test run were recorded as well.

Once the videos were recorded, they were converted to images by extracting the individual frames, and manually sorted into the nine classes using the recorded time stamps. The initial data distribution after the first data collection is shown in Table 3 and Figure 20.

Table 3

Data distribution for first data collection process

Class	**Pilot**	**Copilot**
(0) Down	7081	7040
(1) Down_Left	6714	6600
(2) Down_Right	6503	6443
(3) Left	6680	6698
(4) Right	6644	6763
(5) Straight	7178	7282
(6) Up	6013	5866
(7) Up_Left	6705	6444
(8) Up_Right	6524	6600

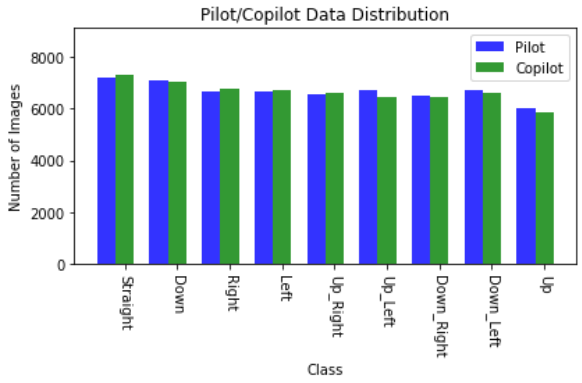

Figure 20. Distribution of data after first data collection process

This dataset is distributed fairly evenly and was used for the preliminary training of the deep learning models. However, after taking a closer look at the data, and after some poor initial results, it was decided that a larger and cleaner dataset would be needed to train the networks.

3.1.4.2 Second data collection. The second time the data was collected, there were only four test subjects with the same diversity as before. Instead of sitting in the actual Sikorsky S76D helicopter, this data was collected with the test subjects sitting in the FAA's Sikorsky S76-D simulator, shown in Figure 21.

Figure 21. Sikorsky S76D simulator used for collecting head pose data

Similar to the first data collection process, the four test subjects did six test runs each, changing the equipment on each run. This time however, the cameras were positioned in such a way that the pilot video was almost an exact mirror of the copilot video when flipped on the y-axis. The camera positions and camera views are both displayed in Figure 22 and Figure 23.

Figure 22. Camera positions from inside the cockpit. The camera in the red box was used to record copilot data and the camera in the yellow box was used to record pilot data

Figure 23. Camera positioning for data collection. The left box on the screen shows the camera view of the pilot's seat and the right box shows the camera view of the copilot's seat

This camera configuration allowed for data augmentation to be used to create more training images without the need for each subject to physically switch seats. The pilot images collected were flipped over the y-axis and used as copilot images, and the copilot images were flipped over the y-axis and used as pilot images. By augmenting the data in this way, the test subjects were able to hold their head in each position for one full minute, rather than just thirty seconds, allowing for double the amount of data to be collected in the same amount of time as before.

The second major change from the first data collection process was that each subject was given a harsher set of constraints as to where their head position could be in each class. A diagram of the valid head poses for each vertical and horizontal movement is shown in Figure 24.

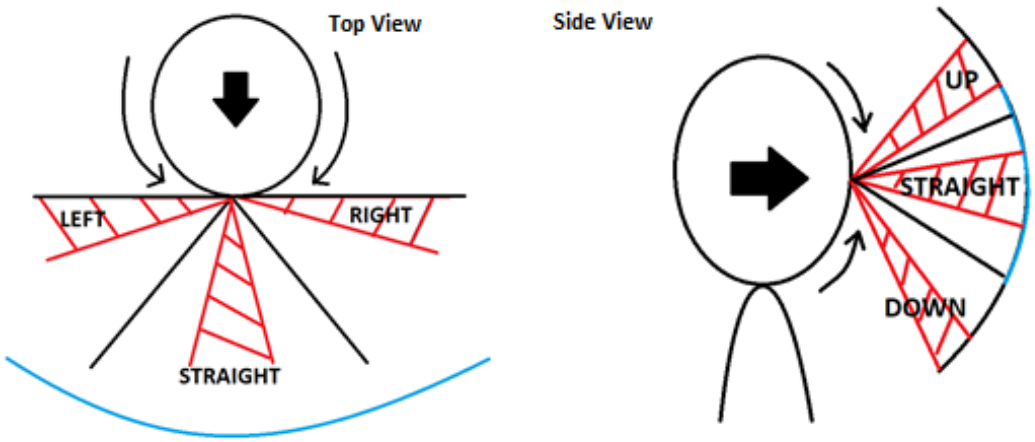

Figure 24. Constraints for head positions during second data collection process

In each view, the solid black arrow shows the direction that the subject's head is facing and the blue curved line represents the windshield of the helicopter. The horizontal movements of the head can be observed by looking at the top view on the left side of Figure 24. The red dashed lines show the valid area that a test subject was allowed to look in each of the horizontal directions. The purpose was to leave a buffer around the boundary between classes as much as possible, so that there would be a sufficient difference between the data that belonged to each class.

The side view on the right side of Figure 24 shows the vertical movements of the subjects. After watching real flight videos it was observed that the difference between a pilot looking up, down, and straight is actually quite subtle, meaning they don't often look straight up or straight down at any point during a flight. For that reason, the test subjects were given specific points to look at to simulate the real world flight data. When the subjects were looking down they were asked to look at the bottom of the instrument panel. When they were looking up, they were asked to look directly at the intersection of the top of the windshield and the cockpit interior. These added constraints were critical when this

data was collected to ensure that the data used for training the networks simulated the real world flight videos.

Once the full test videos were created, the AxisFilePlayer software was used to cut each test run into nine, one minute class videos. This allowed for the videos to be converted to images and written directly to the proper directory which cut down on the time required to organize the data. After the videos were converted to images and organized accordingly, the data distribution was observed. The distribution for the pilot and copilot were exactly the same because the videos were cut at exactly the same times and due to the use of data augmentation mentioned previously. The data distribution is shown in Table 4 and Figure 25.

Table 4

Data distributions for second data collection process

Class	0	1	2	3	4	5	6	7	8
Total Frames	21853	22091	21990	22260	22181	21507	22079	21961	22028

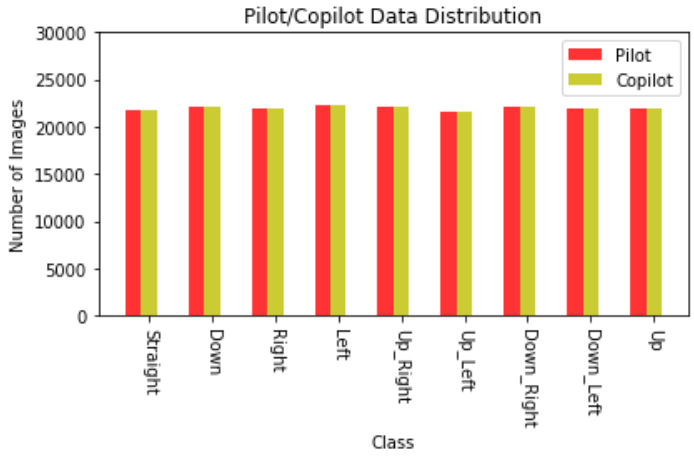

Figure 25. Distribution of data after second collection process

The images from the first collection process are not included in the second dataset. However, this new dataset is sufficiently large with about 21,000 images per class, and it is very evenly distributed. This data was used for the training and testing of each of the deep learning network architectures.

3.2 Hybrid Computer Vision Algorithm

At the start of this research, a deep learning approach for making accurate predictions of the pilot's head position was considered. However, there were no labeled images of pilot's head positions available, and therefore a deep learning network could not be trained. Due to the lack of labeled ground truth data, a combination of classical computer vision techniques and deep learning based detectors was originally selected as the best option to solve the problem of head pose estimation. The hybrid algorithm works with three main steps: (1) face detection, (2) facial landmark annotation, and (3) angle calculations for classification. The algorithm is performed on each frame of the supplied videos independently of one another and serves to classify images into one of four classes: (0) straight out the window, (1) down at the instrument panel, (2) out the window to the side, and (3) none of the above. This algorithm was developed only for the real world copilot video because of the limited amount of test videos available at the time this method was being researched.

3.2.1 Face detection. The first step of the algorithm is to locate the copilot's face in the image and draw a bounding box around it. To accomplish this, two face detectors were considered. The first detector uses a Histogram of Oriented Gradients (HoG) method to perform face detection and the second detector was trained using a deep learning approach. The synthetic test videos were used to see which detector performed best on

clean face images, and on noisier face images where the subject is wearing sunglasses. It was found that the deep learning detector accurately detected a face in all of the images from the clean synthetic video, and 98% of the images from the synthetic video with sunglasses. Two images with correctly detected faces are shown in Figure 26, below. The HoG detector labeled much fewer frames than the deep learning based detector with an accuracy of 94% on the clean video and 70% on the video with sunglasses.

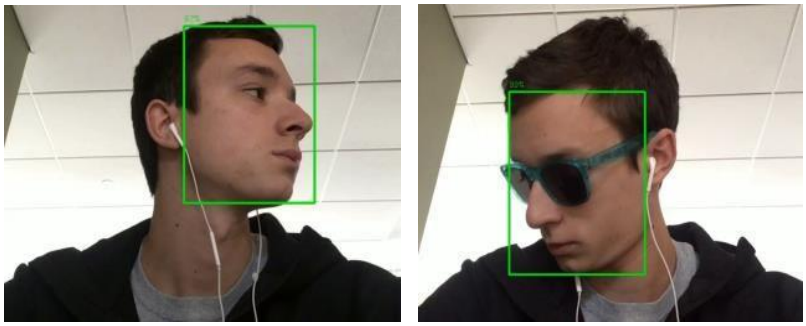

Figure 26. Correctly detected faces from the Synthetic Dataset using the deep learning detector

To further confirm that the deep learning detector was the best choice moving forward, the FAA Flight Dataset was considered. Before processing the face detectors on this dataset, it was observed that in certain instances, the copilot turns their head so far to the side that their helmet blocks their face entirely from the camera's view. An image demonstrating this scenario is shown in Figure 27.

Figure 27. Non-detected face due to occlusion from the helmet

For that reason, each frame of the video that did not contain a face was removed before collecting the following results. Of the 30976 frames in the test video it was found that only 29275 of them have a face present in the frame.

After each face detector was supplied with all 29275 unaltered images, it was recorded that the HoG detector detected 26.02% of the total faces whereas the deep learning detector detected 51.13% of the total faces. These accuracies are much lower than what was observed on the synthetic dataset due to the increase in background information, as well as the added noise of the copilot's helmet, microphone, and other equipment. In an effort to increase the total number of faces detected, the frames were manually cropped to remove the excess background. Due to the small cockpit space and limited mobility of the copilot within this space, the cropped area could be set manually so that a large amount of background could be removed but the copilot never left the cropped region. The difference between the unaltered video frame and the cropped video frame can be observed in Figure 28.

Figure 28. An image from the FAA Flight Dataset before and after cropping

After all frames were cropped, they were again supplied to the two face detectors. The HoG detector performed about the same, predicting 25.54% of the total faces and the deep learning detector found more faces at 75.20%. From these results, it is clear that the deep learning based detector is much more accurate and robust to noise than the HoG detector. From this point forward, only the deep learning based face detector was used.

A closer examination of the missed frames was conducted to better understand why faces were not always being detected. The biggest and most obvious reason is the added noise from the helmet, microphone, and other equipment. However, it was also observed that certain faces were not being detected due to a type of occlusion that causes the face to no longer resemble a face. This can occur when the copilot is at an awkward angle or if their hands are blocking a portion of their face. To a human, it may be easy to identify a face in the presence of noise but to a computer that only sees an image as pixel values, it is more difficult. An example of a partially occluded image is shown in Figure 29.

Figure 29. Non-detected face due to occlusion

This type of image is the most common cause for missed detections where a face is actually present in the frame. It is also important to remember that in a real world application, the detector will always miss video frames as shown in Figure 27, where a helmet is blocking the face of the copilot.

An added benefit of the deep learning detector is that each prediction comes with an accompanying confidence metric. This confidence value describes how sure the detector is that a detection is actually a face. This metric is a valuable addition to the algorithm because it allows for weak predictions or false positives to be eliminated. It should be noted that due to the nature of the SSD framework that is used in the detector, there is often more than one face bounding box per frame even when only one face is present.

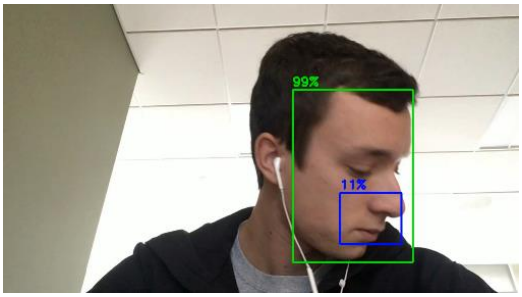

Figure 30. Multiple bounding boxes on an image containing only one face

In Figure 30, the detector predicted one face with 99% confidence (green) and another with only 11% confidence (blue). It is common that most frames with only one face and multiple predictions will have one prediction with a high accuracy and the rest will be low compared to the first one. Also, because of the problem space, it is known that there will only ever be one copilot in each video frame. For that reason, only the prediction with the highest confidence is considered by the algorithm. In Figure 31, only the green bounding box will be considered because it has the highest confidence, and the blue bounding boxes will be omitted.

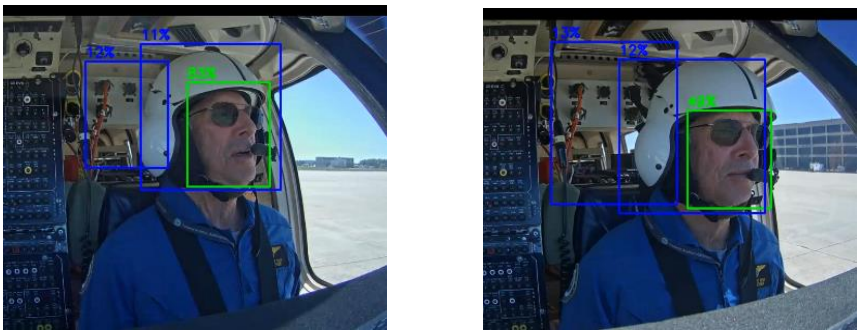

Figure 31. The highest confidence prediction (green) and multiple low confidence predictions (blue) in images with only one face

By setting a minimum confidence threshold value, the algorithm gains the ability to omit false detections if the confidence threshold is not met by the best prediction.

After manually watching the real world flight video with the face detection overlay, another common case of false detections was observed where a very small bounding box is placed incorrectly in the image. These false detections occur almost exclusively when the copilot is looking out the window to the side.

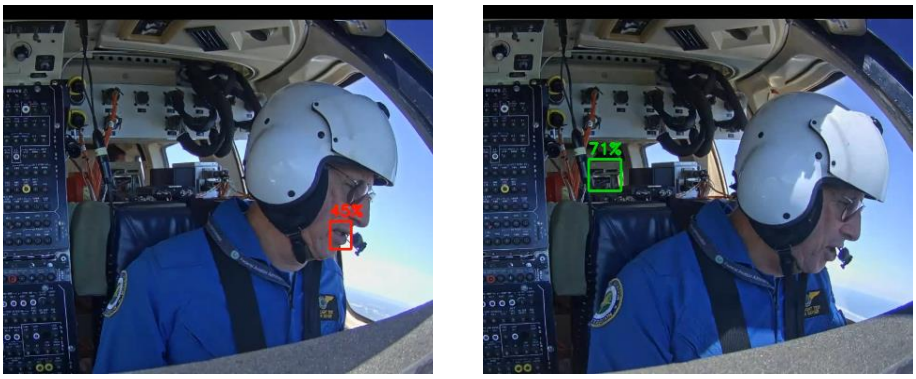

Figure 32. False detections with small bounding boxes and high confidence values

These false detections shown in Figure 32 are easily removed by checking the diagonal distance of the predicted bounding box. Since the copilot is confined to a relatively small space, they cannot move far enough that the size of their face's bounding box will change drastically from frame to frame. Therefore, the algorithm checks that the diagonal distance of the bounding box is greater than a manually defined value, which in this case was determined to be 200 pixels.

After an inspection of the false detections was conducted, various confidence thresholds were tested to find the optimal threshold value. The thresholds were measured using the FAA Flight Dataset. The total number of faces detected as well as the total number of mispredictions based on diagonal distance were recorded. Table 5 summarizes the results with confidence values ranging from 90% to 30% in increments of 10%.

Table 5

Confidence threshold testing

Confidence Threshold	Total frames with face	Total false predictions	Percent of total frames found	Percent of false predictions from number of total faces
90%	18455	0	59.57%	0%
80%	20587	8	66.46%	0.03%
70%	21729	34	70.14%	0.15%
60%	22516	102	72.68%	0.45%
50%	23294	164	75.20%	0.70%
40%	24035	256	77.59%	1.06%
30%	24888	391	80.34%	1.57%

As the confidence threshold decreases, the total number of detected faces and the total number of false detections both increase. The threshold value was selected to be 40% because it detects a large amount of faces correctly while only having about a 1% misprediction rate. Figure 33 shows a few examples of frames where the face was detected accurately even at extreme angles and with some occlusion.

Figure 33. Correct face detection at extreme angles and with some occlusion

The first step of the algorithm is the most important because the position of the bounding box is required for the next two steps of the algorithm. However, a method for estimating the head position when no face is detected is discussed in a future section.

3.2.2 Facial landmark annotation. The second step of the algorithm looks within the bounding box provided by the previous step and annotates important facial landmarks on the face. The facial landmark annotation tool that was selected for this step will always output 68 x-y coordinate pairs which outline the jawline, eyes, eyebrows, nose, and mouth. However, only six of these pairs are saved as part of the algorithm: the tip of the nose and chin, the two outside corners of the eyes, and the two outside corners of the mouth. These points were selected because they are sufficient in approximating the basic geometry of a face. The other 62 points are omitted from use.

Figure 34 displays examples of correctly placed facial landmark annotations on real world copilot images.

Figure 34. Properly placed facial landmark annotations

The results are quite impressive because the facial landmarks are extremely robust considering the added noise from the sunglasses, microphone, as well as occlusion from the copilot's hand. At frontal angles, these points are consistently and accurately placed on

the copilot's face even with added noise. However, at more extreme angles, the facial landmarks are often skewed or wrong entirely like the images in Figure 35.

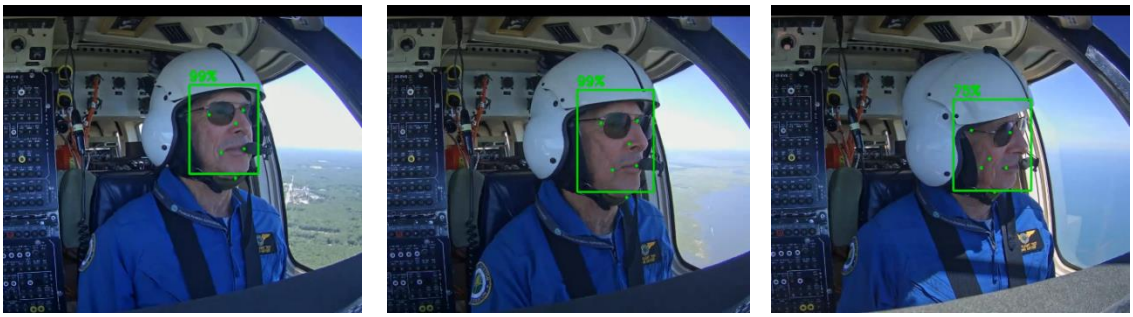

Figure 35. Misplaced facial landmark annotations

As stated in section 2.1.2, the facial landmark annotation tool was trained on a dataset that consisted of frontal faces only. Therefore, when the copilot turns to extreme angles, it makes sense that the facial landmarks will be placed incorrectly. However, it is found that in most cases, and in the first two images of Figure 35 above, the annotations are only slightly off and still give a reasonable estimate of the copilot's head position. Because the goal of the algorithm is not labelling these points exactly, but rather, estimating the direction of the head, certain compensations can be made for poorly placed facial landmarks. This step of the algorithm was identified as one of the biggest limitations due to its inability to accurately place facial landmark annotations at extreme angles.

3.2.3 Angle calculations and classification. Once the position of the facial landmarks are known, the next step of the hybrid algorithm uses the pinhole camera model to obtain a rotation matrix and Euler angles that define the rotation of the copilot's head. In order to do this, a set of 3D reference points was manually defined such that they roughly model a head looking straight forward in 3D space. The 3D reference model is shown in

Figure 36 and consists of the same six points that were annotated in the previous step: the tip of the nose and chin, the two outside corners of the eyes, and the two outside corners of the mouth.

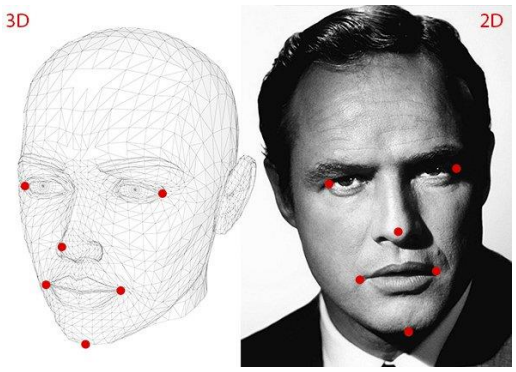

Figure 36. The 3D reference model points compared to the 2D annotated points

Once the 3D points were defined, an estimate of the internal camera matrix was created. The focal lengths are estimated by the height and width of the image and the optical center is estimated to be the center of the image. No radial distortion or skew is considered in the current model. The approximation of the internal camera matrix using the size of the image is intended to create a more generalizable algorithm that can work using different types of cameras as well as different camera mounting conditions. It also prevents the user from having to manually calibrate the camera matrix every time the conditions do change [26].

Referencing Equation 2.9 in Section 2.1.3.2, the internal camera matrix, K, as well as the 3D world coordinates and 2D image coordinates are now known. Therefore a rotation matrix can be calculated which defines the rotation of the world coordinate system in relation to the camera's coordinate system. Equations 2.12, 2.13, and 2.14 are then used to

calculate a pitch, yaw, and roll angle of the head. Pitch is defined as a vertical movement, yaw is defined as a horizontal movement, and roll is defined as a tilt of the head to either side. The standard x-y coordinate system convention is used to define positivity and negativity for the calculated pitch and yaw angles.

For the purpose of classification, only the pitch and yaw angles were considered for labeling any given head position in a video frame. After experimentation, it was concluded that the roll angle does not play a vital part in defining the direction of the head, and for that reason, it is omitted from the classification step of the algorithm. Each frame is applied a vertical or horizontal label depending on the pitch and yaw angles respectively. The threshold values for applying each label are shown in Table 6.

Table 6

Threshold values for applying pitch and yaw labels

Label	**Pitch (α)**	**Yaw (β)**
Up	$\alpha > 10°$	
Down	$\alpha < -10°$	
Straight	$-10° < \alpha < 10°$	$-10° < \beta < 10°$
Left		$\beta < -10°$
Right		$\beta > 10°$

These values were determined by manually watching the real world copilot video and deciding when the copilot was considered to be looking in each direction. It is important to point out that this information is slightly subjective, meaning the thresholds may be different between different people and between different camera mounting conditions. The labels were selected specifically to help increase the number of predictions

in class 1, down at the instrument panel. It was observed that the copilot does not need to move their head much to look down so a relatively small threshold was selected.

Due to the fact that the camera is already mounted at an angle and certain assumptions were made about the 3D reference model as well as the internal camera matrix, a manually defined offset value is applied to the angles calculated in each frame. This ensures that the displayed angles are more comprehensible to the user and the threshold values do not need to be changed if the camera position changes. These offsets were determined by manually locating a frame where the subject was looking straight forward with accurately placed facial landmark annotations, and then adding sufficient offsets to each angle so that they equaled zero. This concept is demonstrated in Figure 37.

Figure 37. The left frame shows the image before offsets are applied, labeled as class 2. The right frame shows the image after angles are zeroed, labeled as class 0

After the angles were properly calibrated, the horizontal and vertical class labels are used to classify the image into one of the four classes. There are a total of nine combinations of vertical and horizontal labels which form a 3x3 grid and correspond to the nine classes described in Table 1. Table 7 shows how the nine classes are categorized into the four classes of interest for this algorithm.

Table 7

All possible label combinations for classification

Class Number	**Class label combinations**
0 – Straight out the window	(3) Left (5) Straight
1 – Down at the instrument panel	(0) Down (1) Down_Left
2 – Out the window to the side	(2) Down_Right (4) Right (8) Up_Right
3 – None of the above	(6) Up (7) Up_Left

It is important to remember that these labels are from the point of view of the camera, not the point of view of the copilot.

The final output of the algorithm is an output video which has the class printed in the bottom left corner of the frame and a .csv file that contains the calculated head position of each frame. Class 0 is shown in red, class 1 in orange, and class 2 in yellow. If no face is detected in the frame it is labeled with a green 3, if it is misclassified based on diagonal distance of the bounding box it is labeled with a green 4, and if a face is detected but it is not classified into one of the three main classes of interest, it is labeled with a green 5. The algorithm has the option to display the face detection bounding boxes, facial landmarks, and angles if desired but will always output the class labels regardless of these other displays. In addition to the output video and output .csv file, the algorithm will also output the total number of frames that were labeled for each class.

3.2.3.1 Jitter compensation. One final aspect of the algorithm helps reduce the jitter of the calculated head positions in the video. The framerate of the supplied real world video is 15 frames per second meaning that the time between two frames is a fraction of a second

and the subject cannot physically move back and forth between classes that quickly. After observing the calculated head positions more closely, it was found that when the subject is transitioning between classes or holding a head position that is right on the threshold of two classes, the head position calculation will jump around between classes.

Figure 38. Three consecutives frames where the predictions jump between two classes

Figure 38 shows how little the copilot moved from frame to frame, but how this slight movement was enough to cause the classification to jump from class 0 to class 1 and then back to class 0. This can be fixed by checking the predictions before and after the current prediction and using Equation 3.1 to smooth the output:

$$if\,(\,pred_{x-1} = pred_{x+1})\;and\;(pred_x\;!= pred_{x-1}):$$

$$pred_x = pred_{x-1}$$

(3.1)

This can be better visualized in Figure 39 below where the subject is transitioning between looking straight out the window and down at the instrument panel.

Before:	0 0 3 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
After:	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

Figure 39. Classifications before and after jitter is removed

It is clear that the output with the jitter compensation is much cleaner and will be much easier to interpret by crash and safety analysts. The example in Figure 39 also demonstrates how removing jitter can actually classify frames where no face is detected. That being said, it should be noted that predictions that belong to class 0, 1, or 2 will not be changed if its surrounding predictions belong to class 3, 4, or 5. This prevents predictions from the classes of interest from being classified into the categories of no face detected, diagonal misdetection, or an image that does not belong to one of the three main classes.

3.2.4 Hybrid compensation method. As stated previously, the algorithm cannot estimate head pose angles if it doesn't first detect a face in the image. This limits the amount of correct predictions primarily in class 2: out the window to the side. In a real world scenario the copilot's helmet is almost always blocking their face from view when they are looking out the window, and therefore a face will not be detected.

After manually looking at the frames that had no face detected, it was observed that the majority of them were when the subject was looking out the window. The other most common case of a face not being detected was in the occlusion cases as discussed in section 3.2.1. However, since the majority of missed faces do happen when the subject is looking out the window to the side, an assumption can be made that if a face is not detected, the subject is most likely looking in that direction. The frame can then be labeled as belonging to class 2 even though a face is not present.

In order to prevent this assumption from also labelling cases of occlusion, each frame where no face is detected will check the previous frame's bounding box location to ensure that it is close to the window side of the cockpit. If the frame prior to no face being detected is close enough to the window, it is assumed that the subject continued to turn their head farther to the side to look out the window. The point that is considered "close enough" must be defined manually and will be dependent on the position of the camera in the cockpit.

3.3 Deep Learning Algorithm

The hybrid head pose estimation algorithm works well for frontal facing poses and certain compensations were made to increase the accuracy at more extreme angles. However, this method does rely on a certain number of assumptions and approximations and will require a fair amount of calibration each time the camera conditions change or a different copilot is in the video. For that reason, it was decided that a ground truth dataset would be created and a purely deep learning algorithm would provide the best solution for the problem of head pose estimation. The deep learning algorithm would require less calibration in different scenarios and would be able to accurately predict head positions

regardless of whether a face is present in the frame. The only input to the deep learning algorithm is the input test video, therefore the algorithm should be able to tell if the video is of a pilot or copilot, and should output the appropriate head pose class automatically.

The process of model selection and hyperparameter tuning is covered in the following sections. The methods for training and evaluating the different models as well as the final structure of the deep learning algorithm are discussed.

3.3.1 Dataset organization. The process of creating the labeled data has already been discussed in the background section of this thesis. From that data there were about 21,000 images for each of the nine classes, totaling 189,000 for the pilot and the same 189,000 images flipped over the y-axis for the copilot. A total of eight datasets were created by splitting the 189,000 images into four groups for the pilot and four groups for the copilot. A ninth dataset was created to determine if a video was on the pilot side or the copilot side of the cockpit.

The first grouping of data was created as the baseline dataset and consisted of a combination of helmet and headset images (with sunglasses, visors, etc.). However, after closer inspection of the data, it was a possible concern that the headset images and the helmet images that belonged to the same class were too different, specifically when the pilot is looking out the window. This difference can be observed in Figure 40.

Figure 40. Images belonging to the same class; one with headset and one with helmet

For this reason, two more groups of data were created: one that contained nine classes but only images of the pilot/copilot with a headset on, and another that contained nine classes but only images of the pilot/copilot with a helmet on. The point of separating the data into these two groups was to increase the overall accuracy of the algorithm especially in the case where the pilot/copilot is looking out the window. Due to the limited amount of labeled data at the time of this research, these separate helmet/headset datasets can provide a more accurate prediction than the combined dataset in some cases. In addition, having one combined prediction and one headset/helmet prediction also adds some redundancy to the algorithm and can help analysts or interpreters to identify misclassifications if they see a large difference between the two predictions.

Additionally, a two class dataset was created to identify the headgear that the pilot/copilot is wearing in the test videos. This dataset has two classes where the first class consists of headset images and the second class contains helmet images. The same images from all nine classes in the previous datasets were reused but relabeled into these two new classes. All the headset images were used totaling 63,000 images in class zero, and this number was matched with helmet images in class one to ensure an even data distribution.

The images in this dataset were selected at random from all available helmet images belonging to all nine classes.

Finally, a two class dataset was created to identify which side of the cockpit the video was taking place. This dataset was created to further automate the video processing step of the algorithm.

Each dataset was split into a training/validation/test split using a distribution of 90/5/5 respectively. A summary of the datasets and approximate distributions is provided in the table below. The numbers listed in Table 8 below are for the pilot datasets but the numbers will be identical for the copilot datasets.

Table 8

Dataset summary

Dataset Name	**Training Images per Class**	**Validation Images per class**	**Testing Images per Class**	**Total Images per Class**
PCombined_9Class	19,800	1,100	1,100	22,000
PHeadset_9Class	6,660	370	370	7,400
PHelmet_9Class	13,140	730	730	14,600
PClassifier_2Class	59,742	3,319	3,319	66,380
HelicopterSide_2Class	30,000	1,500	1,500	33,000

3.3.2 Model selection and hyperparamter tuning. The challenge of any deep learning approach is that the hyperparameter values such as learning rate, dropout rate, and pooling that will produce a viable solution are almost always unknown. On top of this, it is also difficult to know what network architecture will work best given the data available. If the data is very complex, the network needs to be deep enough to learn complex features, but if the data is simpler, a deep network may cause overfitting. In order to tune the

hyperparameters and to figure out which network architecture will work best for the task of head pose estimation, multiple networks architectures were trained with different combinations of hyperparameters in order to explore a variety of possible solutions.

The coding framework that was used for the training and evaluating process was Keras. This framework provides users with Keras Applications which allow for the easy implementation of prebuilt network architectures. There are a wide variety of network architectures that are available in Keras Applications and seven of them were considered in the initial testing phase of this research. The architectures include: ResNet50, VGG16, VGG19, InceptionV3, Xception, InceptionResNetV2, and DenseNet121. An output layer was added to each of these networks to ensure that the output of the network had the correct number of classes for the dataset being used.

Alongside different network architectures, different combinations of hyperparameters were used. Table 9 below shows the selected values of each hyperparameter used for training. The pooling refers to the pooling applied to the last layer only.

Table 9

Hyperparameter combinations

Learning Rate	**Dropout Rate**	**Pooling**
0.0009	0.5	None
0.005	0.25	Average
0.001	0.1	
0.01	0	

The first dataset that was considered for training was the PHelmet_9Class dataset. This nine class network consisted of pilot helmet images only and was selected as the

dataset for initial testing because it was smaller than the full dataset, allowing for multiple models to be trained quickly, and because the images in this dataset are generalizable to the images in the other eight datasets. For each of the seven network architectures listed above, a total of eight models were trained with different hyperparameter combinations. The initial results from these 56 trained models were used to narrow down the search for well-performing combinations of architecture and hyperparameters.

In order to evaluate each of the models, the model weights and network architectures were saved after each training session. The architectures and weights were then used to get a head pose prediction for each image in the test set. The networks did not see these test images at any time during the training process so the testing accuracies presented represent the generalizability of the model on real world data. The predictions for each image were recorded and aligned in a confusion matrix so the accuracy of each class could be observed. The total accuracy of each model was calculated as well.

A confusion matrix is a representation of how well a model is performing. An example is shown in Figure 41.

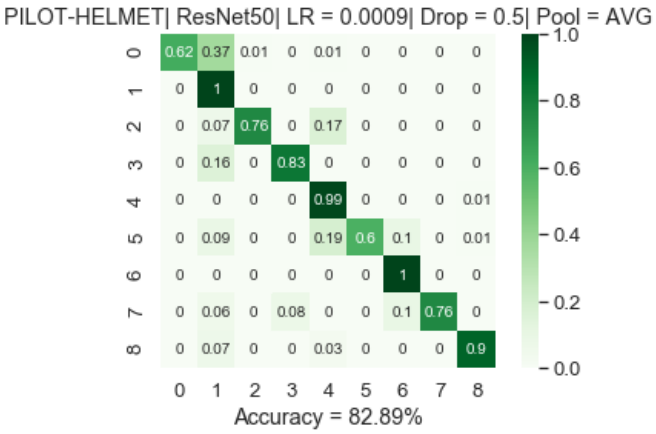

Figure 41. Sample confusion matrix

Each block of a confusion matrix represents how many test images from a specific class were classified into each of the nine classes. For example, the top row of the confusion matrix in Figure 41 represents all the test images from class 0, and the columns represent the output predictions of the model. The blocks in the first row show that 62% of the test images belonging to class 0, were correctly classified as class 0. However, 37% of the test images from class 0 were classified as class 1, and 1% were also classified as class 2 and 4. Looking at the second row from the top, it shows that 100% of the test images from class 1 were correctly classified as class 1.

A model that is 100% accurate on the test set should ideally have a confusion matrix with a diagonal of 1's going from the top left corner to the bottom right corner. The total accuracy of the model is calculated by taking the average of all the diagonal blocks in the confusion matrix, because these blocks represent correct class predictions.

The confusion matrices for all 56 models trained on the PHelmet_9Class dataset are shown in Figures 42-48. These confusion matrices are included to demonstrate which architectures performed well on the data and which combination of hyperparameters outperformed others. The architecture, learning rate, dropout rate, and pooling for each model are included in the title of each confusion matrix, and the overall accuracy is displayed at the bottom of each matrix.

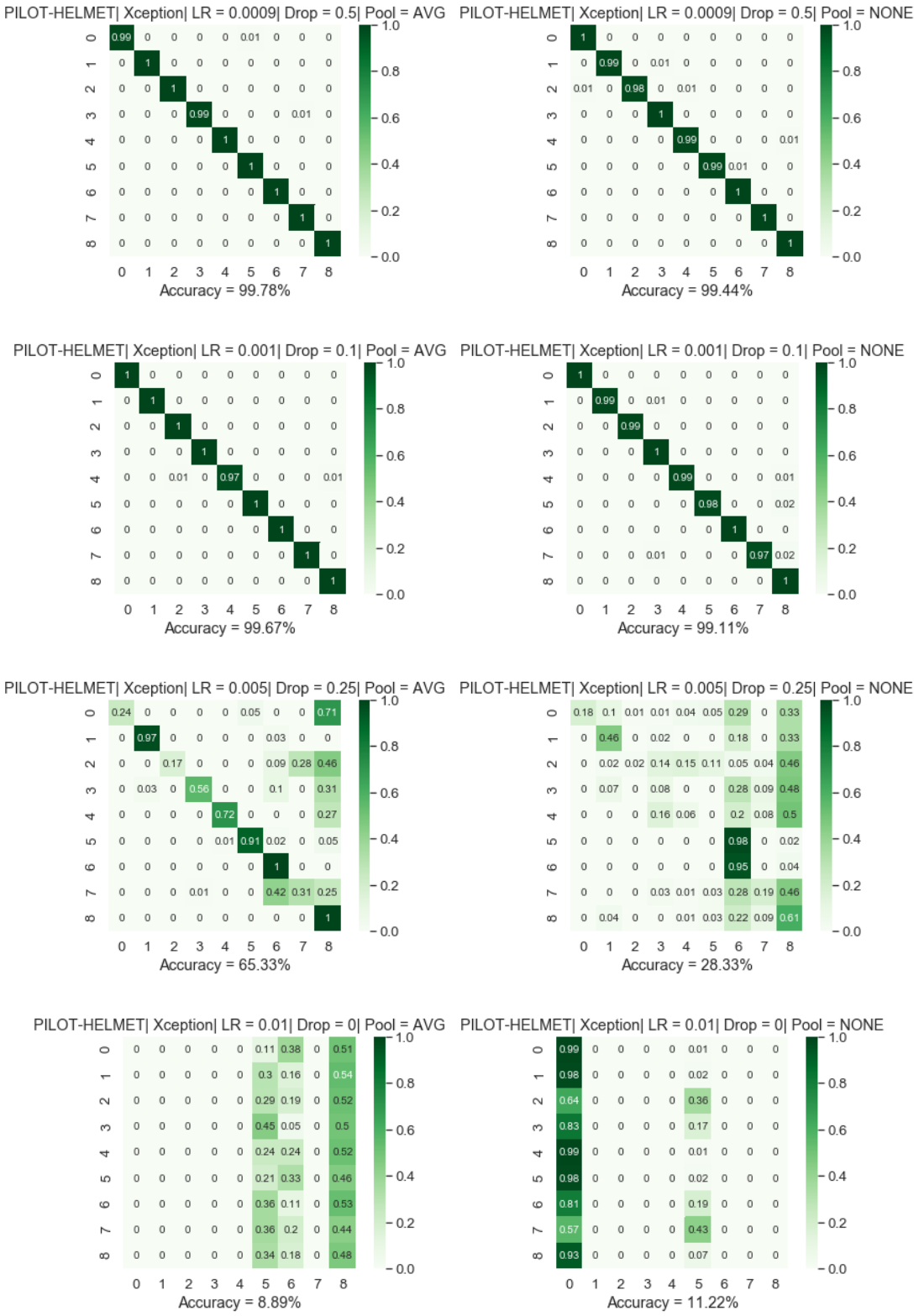

Figure 42. Xception models

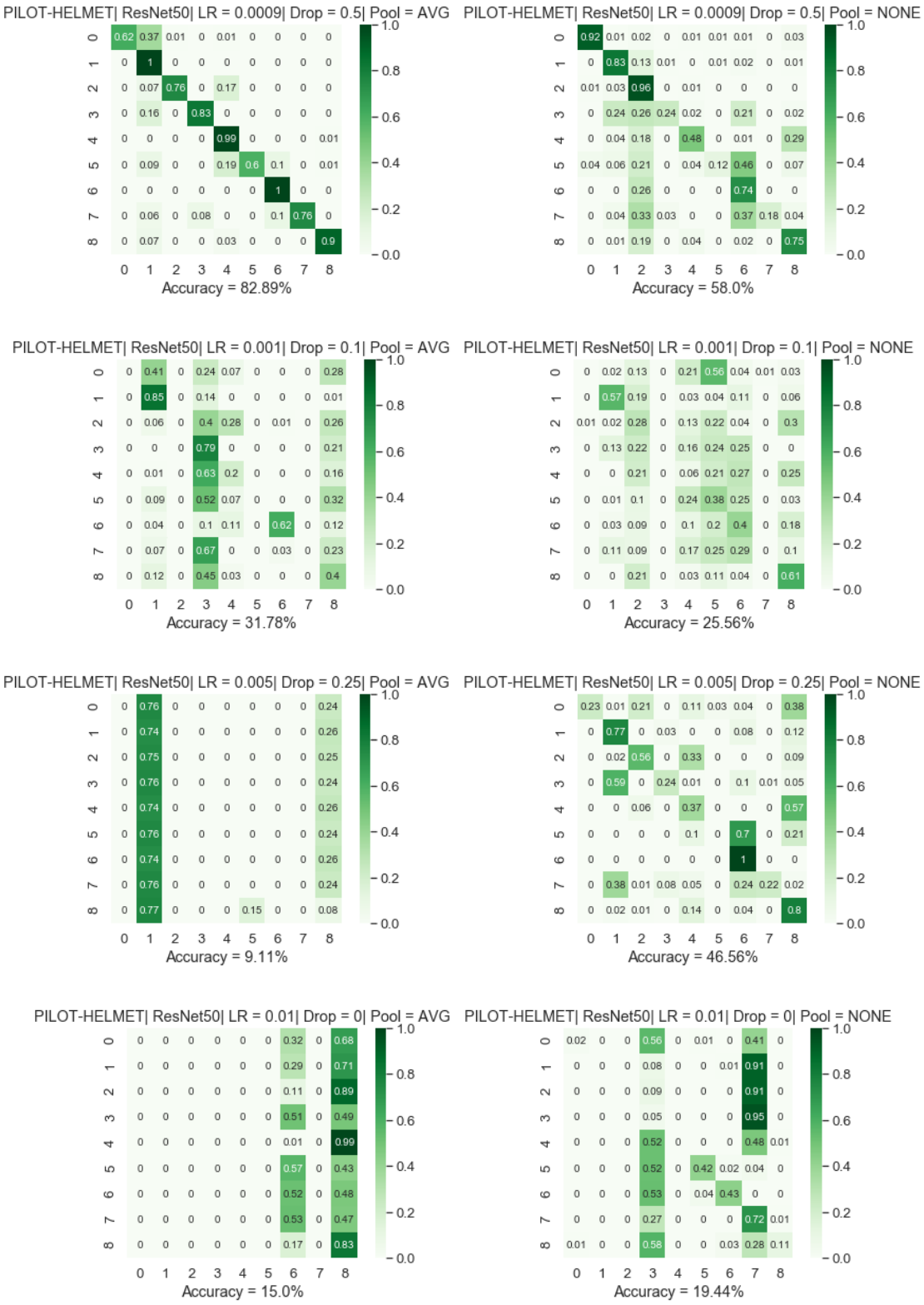

Figure 43. ResNet50 models

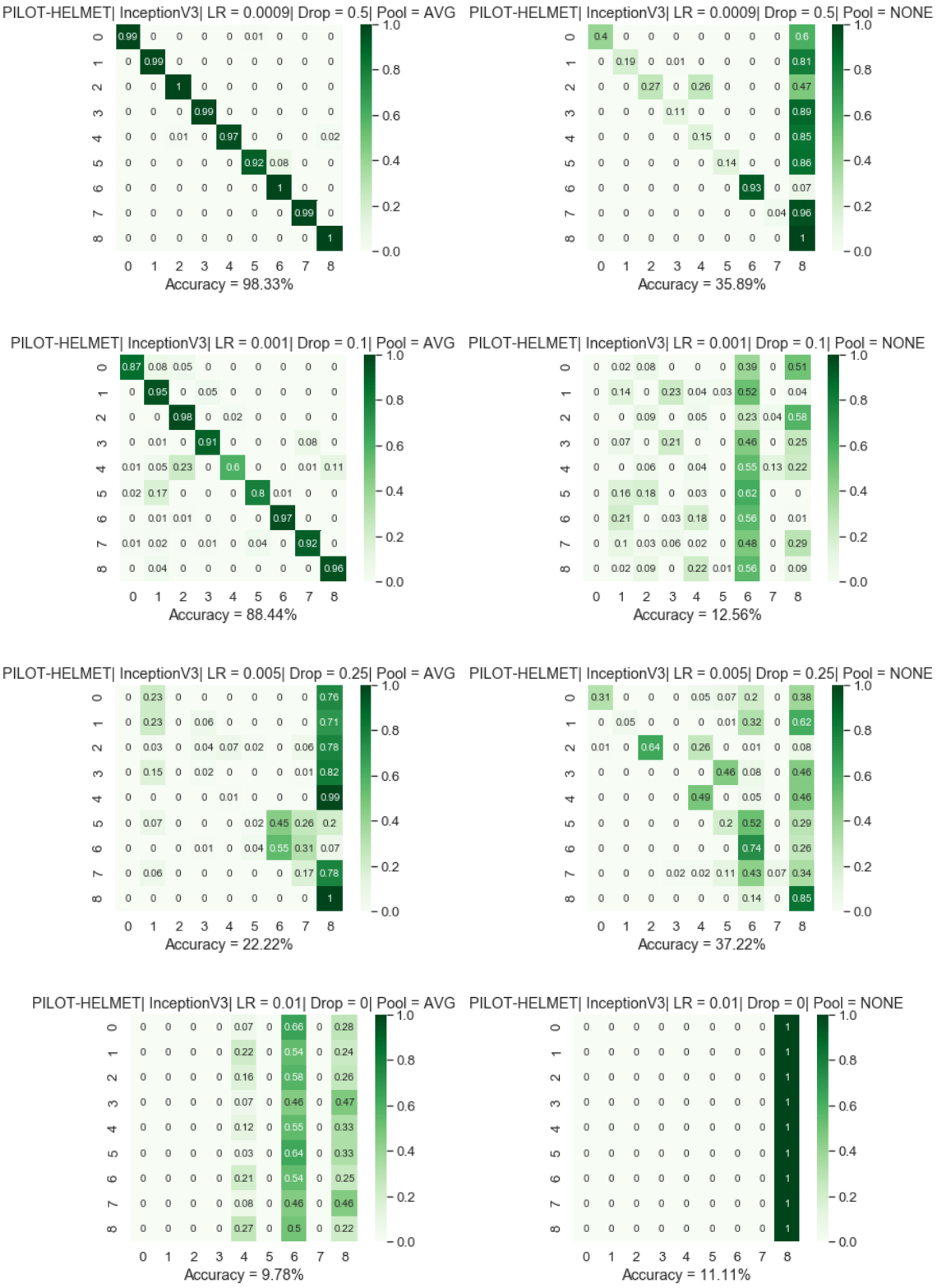

Figure 44. InceptionV3 models

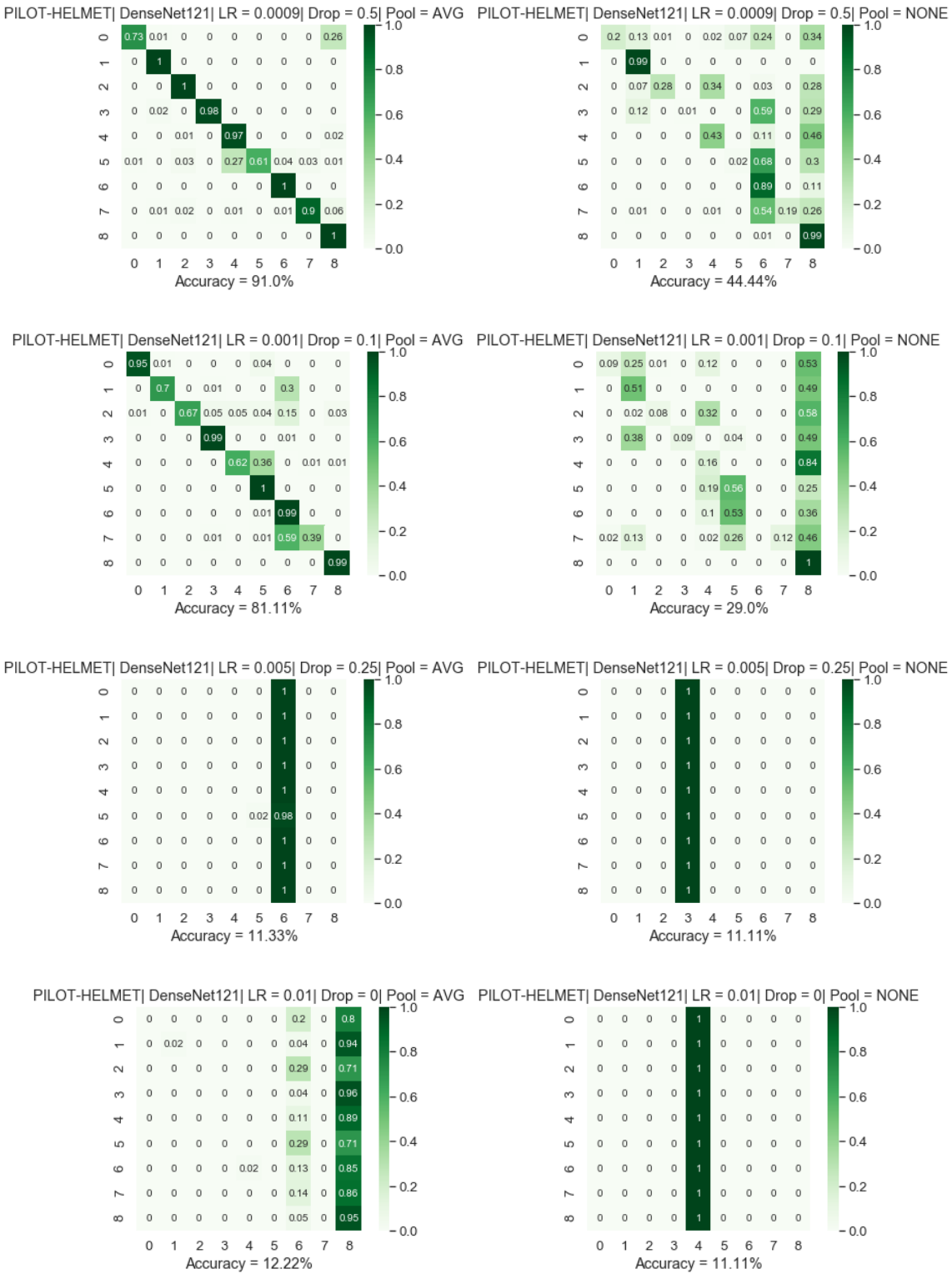

Figure 45. DenseNet121 models

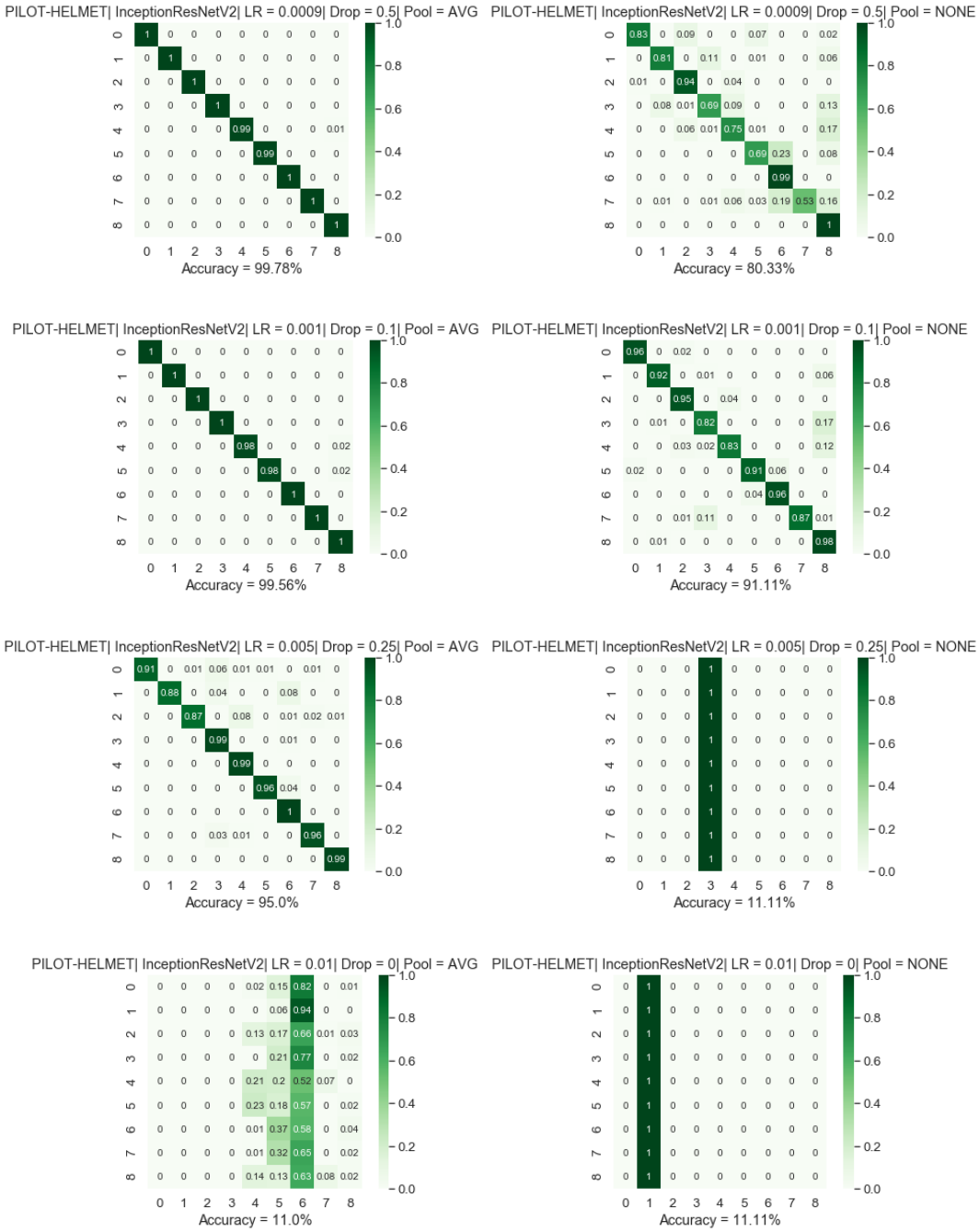

Figure 46. InceptionResNetV2 models

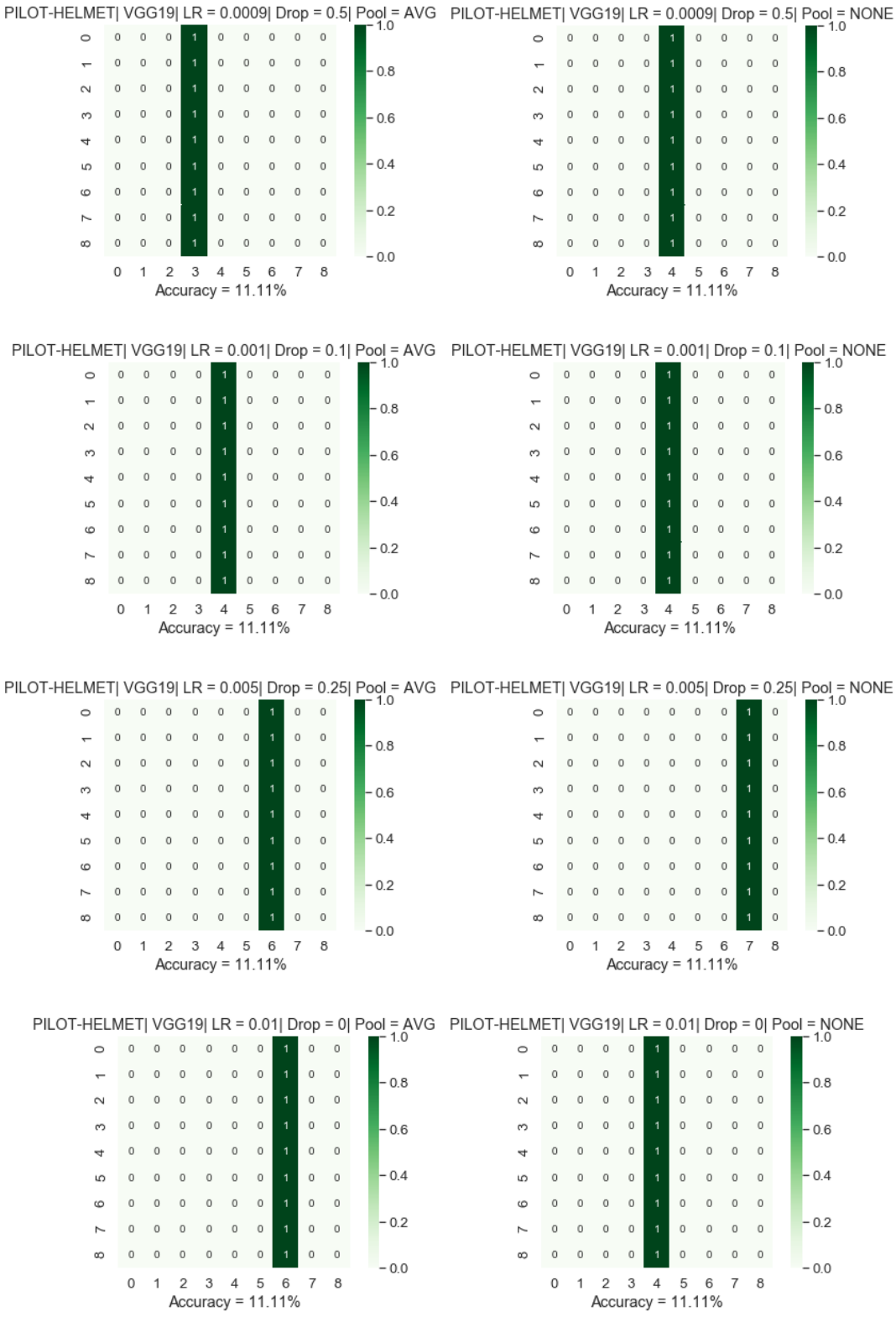

Figure 47. VGG19 models

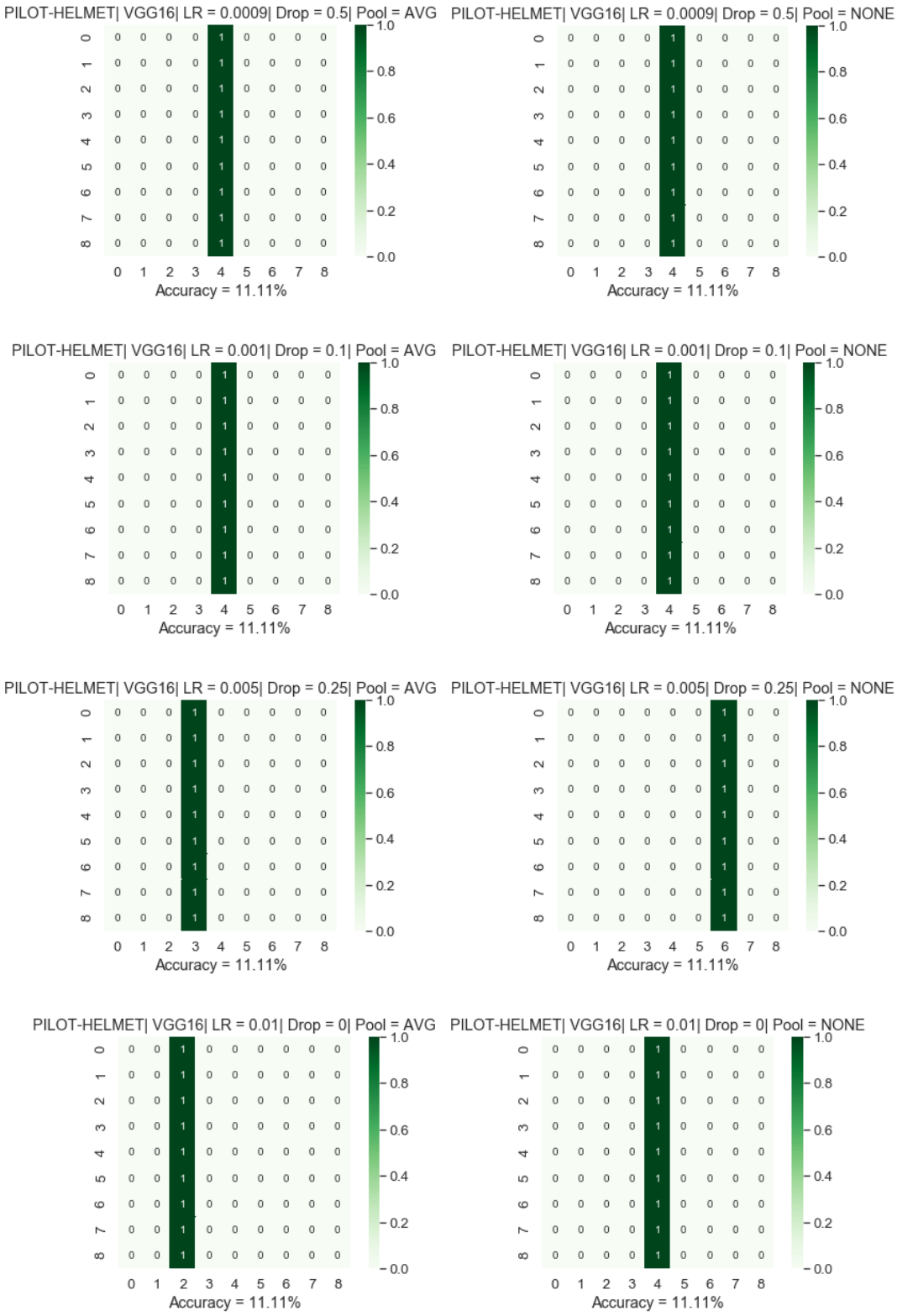

Figure 48. VGG16 models

From the initial results it was clear that the networks performed best with a relatively small learning rate and at least some percentage of dropout. For that reason, the dropout rate of 0 and the learning rate of 0.01 were removed from testing in the future. The most promising results were obtained in most cases using a learning rate of 0.0009 and a dropout rate of 0.5 with average pooling and no pooling both working in some cases. It was also observed in Figure 47 and Figure 48 that the VGG19 and VGG16 architectures produced poor results regardless of the hyperparameters. This is due to the fact that these network architectures are quite large and are most likely too complex for the data.

For the remaining eight datasets, the Xception, InceptionV3, and ResNet50 architectures were considered. Although the InceptionResNetV2 performed well on the most combinations of hyperparameters, the three architectures were selected because they each have their own unique aspects whether using residual blocks, inception modules, or depthwise separable convolutions. By narrowing down the architectures and hyperparameter combinations moving forward, the number of models trained for each dataset was greatly reduced from 56 models to 18 models.

Once the number of networks was reduced, the 18 different models were trained on each of the four copilot datasets. The copilot datasets were considered next in order to verify that the hyperparameters and architectures that produced good results on the pilot helmet dataset also produced good results on the copilot datasets. The results and overall accuracies of the 18 models were recorded for each of the copilot datasets and the model with the best overall accuracy was highlighted.

Once the copilot models were trained, the remaining pilot models and the helicopter side model were trained. Rather than training 18 networks per pilot dataset, only one or

two networks were trained using the combination of architecture and hyperparameters that produced the best results on the corresponding copilot dataset. This assumption was made because the pilot images are similar to the copilot images except that they are flipped over the Y-axis. The best results from the copilot classifier were used to train the helicopter side model.

3.3.3 Final algorithm structure. Once there was a working model for each of the nine datasets, the final structure of the algorithm was created. Each frame of the input video would be passed through a total of four networks. The final algorithm structure is depicted in Figure 49.

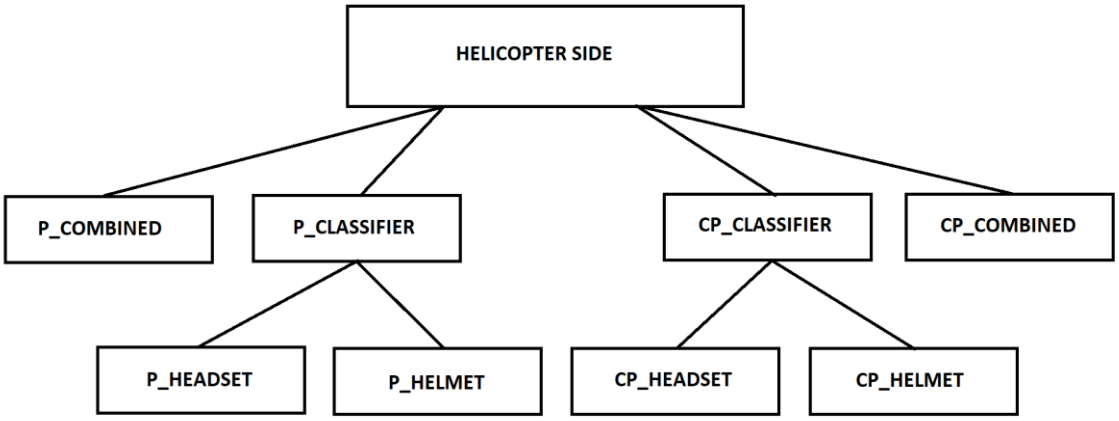

Figure 49. Deep learning algorithm structure

The input image will first pass through the helicopter side model to determine if the video is of a pilot or a copilot. Depending on that prediction, the image will then be given to both the combined model and the headgear classifier model. The output from the headgear classifier will then determine if the image will be given to the headset model or helmet model. That being said, each image will have an accompanying helicopter side

prediction, combined prediction, classifier prediction, and either a headset or helmet prediction.

The final output of the deep learning algorithm is an output video and an accompanying .csv file. All four predictions for each frame will be printed on the output video and these four predictions will be saved to a .csv file with its appropriate time stamp and frame number. A summary of the total number of frames classified into each class is also included at the bottom of the .csv file.

Up to this point all classes were labeled from the point of view of the camera. However, it is known that the actual direction of the head pose should be from the point of view of the pilot/copilot. For that reason, all labeled data for training and testing will remain from the camera's point of view, but the display on the output video and the predictions in the .csv file are changed to be from the pilot/copilot's point of view.

3.3.4 Generalizing to a real world dataset. The models discussed in the previous section were trained on simulator data only and no real flight data was included. That being said, these initial models did not perform well on the images from the FAA Flight Dataset. The difference between the simulator images and the real world images can be observed in Figure 50.

Figure 50. Simulator data compared to real flight data

Although the camera angle may not look much different to the human eye, remember that a computer only sees an image as pixel values. This means that any change of background, average pixel color, or camera angle can drastically effect the ability of the model to provide accurate predictions.

Since this algorithm will be used for real flight video data and not just simulator data, it is important that the models generalize to new cockpits, camera angles, or testing conditions without the need to have an excess of ground truth data each time the video conditions change. In order to generalize to more situations, more variations in the training data are required. The simulator data must be used at this stage of the research because labeled head pose data is limited, however future training data variations should include slight camera angle adjustments, different cockpit interiors, different pilots/copilots, different headgear/equipment, and different flight scenarios such as daytime or nighttime flights.

To demonstrate the process of generalizing the models to real world copilot data, images from this video were labelled manually and included in the training data along with the simulator images. From the 30 minute test video, the first 10 minutes were set aside for final testing, and the images from the second 20 minutes were manually labeled into each of the nine classes. The total data distribution from this manual labeling process is shown in Figure 51.

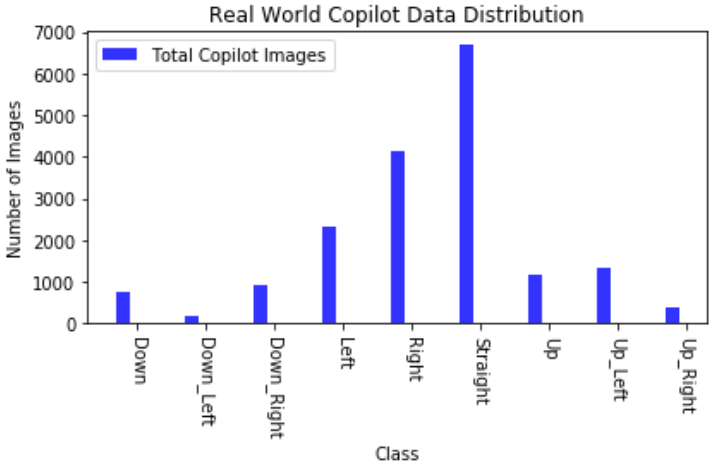

Figure 51. Data distribution from the last 20 minutes of the real world copilot video

A total of 17,907 new images were collected across all nine classes. While there is a clear imbalance in the new images, this imbalance became less obvious when these images were added and shuffled in with the current copilot datasets that contained only simulator data. Figure 52 shows the total data available for the copilot after the labeled real world data was added to the simulator images.

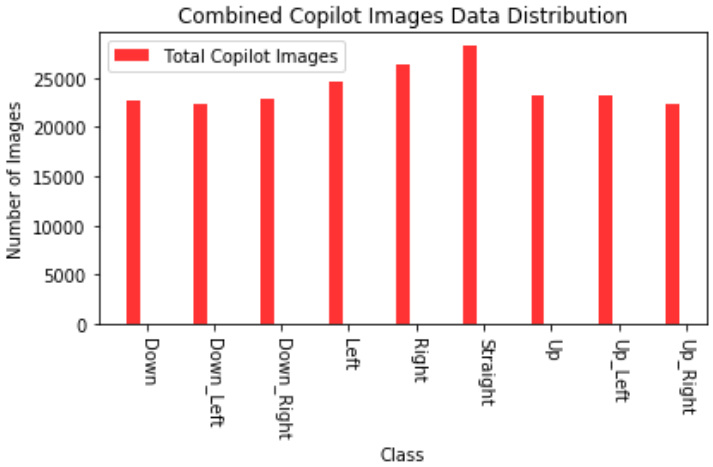

Figure 52. Copilot data distribution of combined simulator data and real world data

Once these new images were added to the training set, four of the nine models were retrained with the real world copilot images included. By adding only a few more examples to each class from this new dataset, the networks were able to perform well on real flight data without sacrificing their accuracies on the simulator data. An explanation of how these models performed will be explored in the next chapter.

Chapter 4

Results

This chapter will contain a full presentation of the results from both algorithms discussed in this thesis: the hybrid computer vision algorithm and the purely deep learning algorithm. The performance of the hybrid algorithm on a benchmark dataset and on a real world flight dataset will be explored. The effects of the hybrid compensation method for classifying images where no face is detected will also be discussed.

The best models that were trained for each of the nine head pose datasets are analyzed by looking at their training accuracies and confusion matrices. An overall analysis of the entire deep learning algorithm is included as well. A discussion on generalizing the models to new data is also presented in this chapter, and the strengths and weaknesses of both algorithms are highlighted.

4.1 Hybrid Computer Vision Algorithm Results

The first experiment conducted for testing the accuracy of the hybrid head pose estimation algorithm was performed on the Head Pose Image Dataset. As stated previously, this benchmark dataset is made up of 2790 images, each with a corresponding pitch and yaw label to define the position of the head in the image. The absolute error between the true pitch and yaw angle and the algorithm's calculated pitch and yaw angle was calculated for all 2790 images. Once the error was calculated for every image, the average absolute error for each individual pitch and yaw angle was obtained. The average absolute error is displayed in Figure 53 and Figure 54.

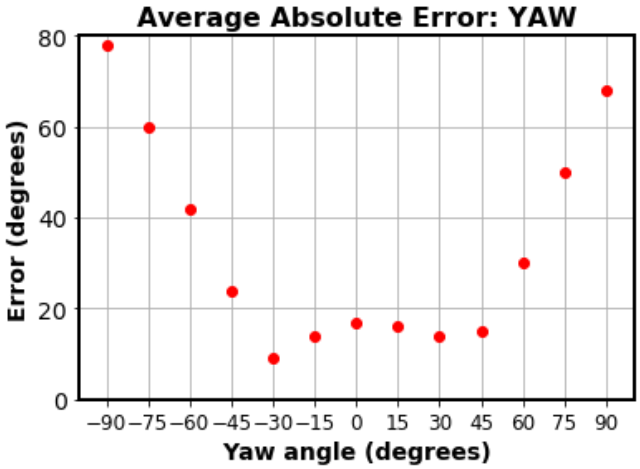

Figure 53. Average absolute error for yaw angles

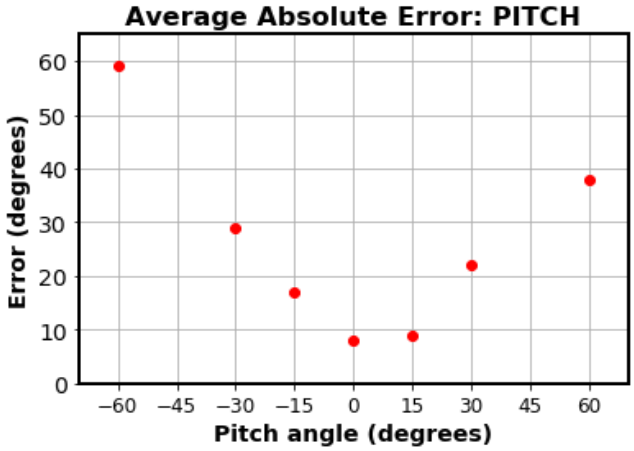

Figure 54. Average absolute error for pitch angles

These results clearly show that as the head pose angle becomes more extreme, the absolute error increases very quickly. The algorithm performs well in calculating yaw angles in the range of ±30° and pitch angles in the range of ±15°. This is not an unexpected result because the facial landmark annotation tool was only trained on faces in these ranges. The results presented in Figure 53 and 54 verify the landmark annotation tool can provide accurate annotations for frontal facing poses within a certain range.

After collecting a quantifiable accuracy of the angles calculated by the hybrid algorithm using the Head Pose Image Dataset, the next experiment was performed on the FAA Flight Dataset. The first 10,000 frames of this real world flight video were manually labeled into nine classes so the output classifications from the algorithm could be compared to ground truth values. Table 5 in chapter 3 displays how the nine classes are divided into the four classes of interest for the hybrid algorithm. Table 10 shows the total number of frames belonging to each class.

Table 10

Total number of frames belonging to each class in the FAA Flight Dataset

Class	**Total Frames**
(0) Straight out the window	5629
(1) Down at the instrument panel	627
(2) Out the window to the side	1859
(3) None of the above	1885

By observing Table 10, it is apparent that the majority of the time during flight, the copilot is looking straight out the window. That being said, the algorithm should be able to perform well on this class specifically. Figure 55 shows a correctly classified image from the first three classes.

Figure 55. Correctly classified copilot frames

After supplying all 10,000 test images to the hybrid algorithm and comparing its classifications with the ground truth, the overall accuracy was calculated to be 46.01%. While this accuracy does seem quite low, it is important to look at the accuracy of each class individually by observing the confusion matrix in Figure 56.

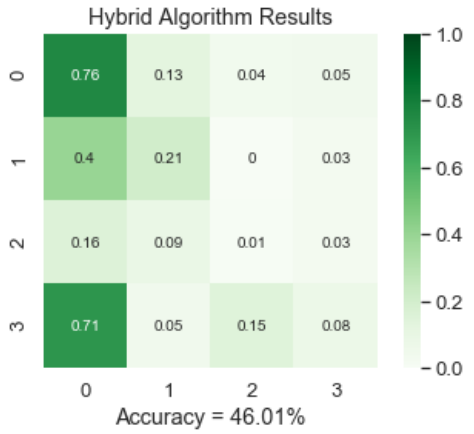

Figure 56. Confusion matrix for standard hybrid algorithm

The algorithm performs best in classifying frames where the copilot is looking straight out the window, correctly classifying 76% of those images. Of the 10,000 labeled test frames, about 56% of them belong to class 0. Therefore, it is a good sign that the algorithm performs well in that class specifically. However, the correct classifications for class 1 and class 2 are much lower than would be desired. The confusion matrix also clearly shows that when the algorithm classifies a frame incorrectly, it is frequently classifying the frame as straight out the window. This should not be a surprise, again because the facial landmark annotation tool was trained on frontal faces only.

Looking more closely at the incorrectly classified images, a few conclusions can be made. First, the algorithm struggles to classify images into class 1, down at the instrument panel, because the difference between the copilot looking straight and the copilot looking down is subtle. Therefore, the facial landmarks that are annotated onto the face need to be very accurate in order to detect this small change from class 0 to class 1. Due to the added noise from the copilot's sunglasses, microphone, and other equipment, the facial landmarks are not as accurate as they would be on a clean image with no noise. The inaccuracy of the

facial landmark annotations due to this added noise was concluded to be the major reason for the lack of correctly classified frames in class 1.

Second, the algorithm does not classify frames into class 2, out the window to the side, because of the limitations of the face detector. It was found that from the 1859 frames belonging to class 2, the face detector was unable to detect a face in 70% of them. As stated in chapter 3 of this thesis, a face must be detected in the frame in order to get a head pose classification. This is a huge limitation for the overall accuracy of the algorithm, however as discussed in section 3.2.4, a method for classifying head positions where no face is detected was included in a second version of the algorithm. After implementing this new method and recalculating the overall accuracy, the new accuracy was 55.26%, an increase overall of about 9%.

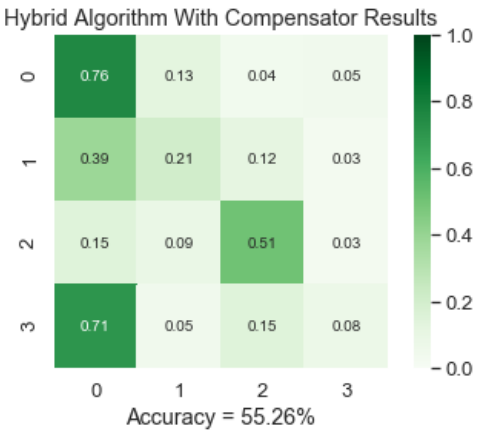

Figure 57. Confusion matrix for hybrid algorithm with compensator

Looking at the matrix in Figure 57, it is clear that classes 0, 1, and 3 have not changed much compared to the confusion matrix in Figure 56. However, the accuracy of class 2 increased by 50%. This improvement validates the assumptions made for the

compensation method. Of the 70% of frames that previously had no face detected, 59% were now correctly classified as belonging to class 2. Two images that were previously labeled as having no face detected but are now labeled correctly to class 2 are displayed in the Figure 58.

Figure 58. Correctly labeled frames where no face was detected

From these results it can be said that the hybrid algorithm proposed can provide an accurate estimation of helicopter pilot head pose in certain scenarios. Since the pilots are looking straight ahead most of the time, 56% in the first 10 minutes of this test video, the algorithm is a valid solution for predicting these head poses. However, there are some obvious limitations when it comes to detecting small changes between classes and when the head position is at an extreme angle. Certain methods have been implemented to overcome these limitations, but there is more to be desired in terms of accuracy in the extreme angle case. For that reason, these initial results were used as motivation to create a true ground truth dataset so that a more accurate deep learning model could be trained to predict head poses at both frontal angles and at extreme angles, regardless of whether or not a face is present in the frame.

4.2 Deep Learning Algorithm Results (Simulator)

A total of 18 models were trained on each of the four copilot datasets for a total of 72 copilot models. As stated in section 3.4.2, the network architectures considered in these tests were the ResNet50, InceptionV3, and Xception architectures. After the initial tests on the PHelmet_9Class dataset detailed in section 3.3.2, a total of six models were trained for each architecture using the following combinations of hyperparameters shown in Table 11.

Table 11

Hyperparameter combinations for each test model

Model	**Learning Rate**	**Dropout Rate**	**Pooling**
1	0.0009	0.5	Average
2	0.0009	0.5	None
3	0.005	0.25	Average
4	0.005	0.25	None
5	0.001	0.1	Average
6	0.001	0.1	None

The copilot datasets were trained with simulator images only, and the hyperparameters that produced the best results for each of the four datasets are displayed in Table 12.

Table 12

Hyperparameter summary for best models

Network	**Architecture**	**Learning Rate**	**Dropout Rate**	**Pooling**
CPHelmet_9Class	Xception	0.0009	0.5	Average
CPHeadset_9Class	Xception	0.0009	0.5	Average
CPCombined_9Class	Xception	0.0009	0.5	Average
CPClassifier_2Class	ResNet50	0.0009	0.5	None

From Table 12 it is clear to see that the best learning and dropout rates in all cases were 0.0009 and 0.5 respectively. Average pooling seemed to perform well on the nine class datasets, where no pooling in the last layer resulted in better performance on the two class headgear classifier. The Xception architecture also outperformed the other architectures on the nine class datasets, and the ResNet50 architecture performed best on the data with only two classes. The training accuracy, validation accuracy, and loss for each of the best models are shown in Table 13.

Table 13

Summary of accuracy and loss for copilot models

Network	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**	**Test Accuracy**
CPHelmet_9Class	99.84%	0.00529	99.96%	0.00197	98.67%
CPHeadset9_9Class	99.71%	0.01102	99.93%	0.00299	99.89%
CPCombined_9Class	99.91%	0.00296	100%	0.00034	99.11%
CPClassifier_2Class	99.98%	0.00083	99.98%	0.00105	100%

The training and validation accuracies are expected to be high because that confirms that the networks are learning the important features and information from the training images. The test accuracy was calculated using a test set of images that was not shown to the network at any time during the training process. Observing a small difference between the training, validation, and test accuracies validates that the model is not overfitting to the training data and is remaining generalizable to data that it has not seen before. The confusion matrices for these four models are shown in Figure 59.

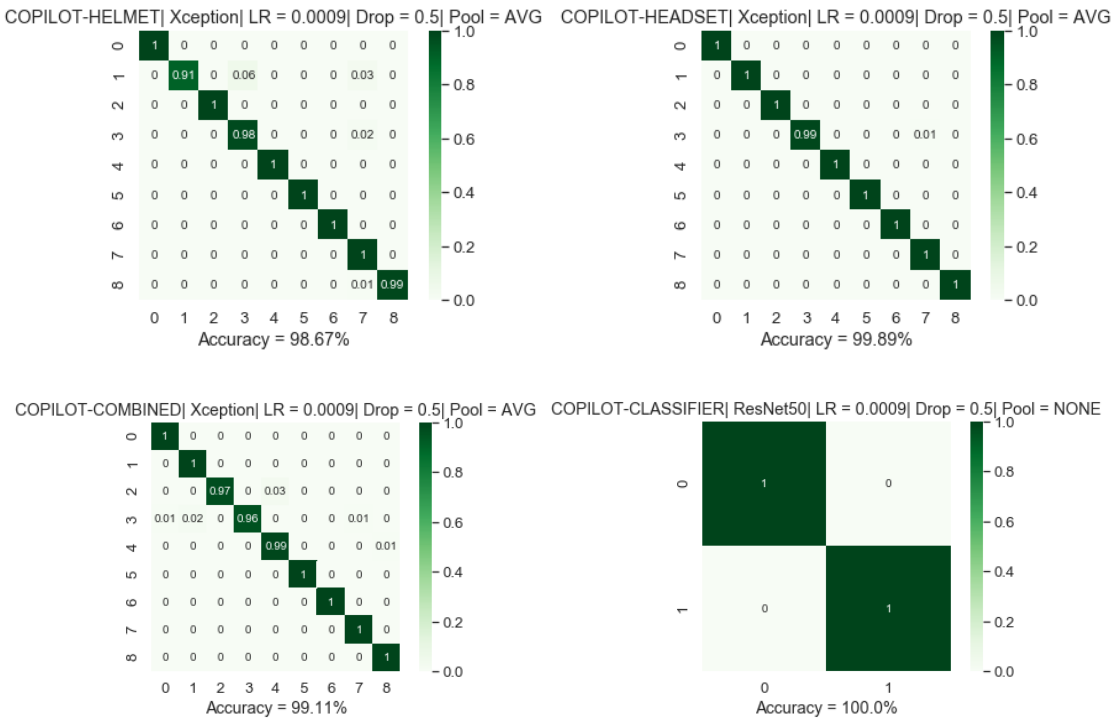

Figure 59. Confusion matrices for copilot simulator models

Taking a closer look at these test accuracies shows that they are very high (100% or almost 100% in all cases). In almost all deep learning solutions, a model will never be 100% accurate on the test set so this should be explained further. The labeled images used for training and testing were created in a controlled environment in the simulator, so a large amount of data could be collected and organized in an efficient way. However, collecting the data in this way limits the variations between images in each class. There are some variations in the form of equipment changes and different test subjects, however, this variation is minimal when looking at the entire dataset. The camera angle and background are exactly the same in all images and this can contribute to a very high training and testing accuracy because the training images are extremely similar to the test images. As more data becomes available with more variations of camera angle, cockpit interior, test pilots,

and background, the test accuracies will still be high, but will not be 100% accurate. At this time however, the labeled data that is available is contributing to test accuracies that are very high.

Once the four copilot datasets had working models, the same combinations of hyperparameters from Table 12 were used to train a single model for each of the four pilot datasets. These hyperparameters work for both pilot and copilot images because these images are essentially the same just flipped over the y-axis. The best results for the pilot models are shown below in Table 14.

Table 14

Summary of accuracy and loss for pilot models

Network	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**	**Test Accuracy**
PHelmet_9Class	99.87%	0.00575	100%	0.000106	99.78%
PHeadset9_9Class	99.82%	0.00808	100%	0.000039	99.89%
PCombined_9Class	99.95%	0.00206	100%	0.000049	99.33%
PClassifier_2Class	99.87%	0.00858	100%	0.000001	99.50%

Again, there is a small difference between training accuracy and testing accuracy which shows that the network is not overfitting. The accuracies are very high again because the variations in the data are still limited. The confusion matrix for each model is displayed in Figure 60.

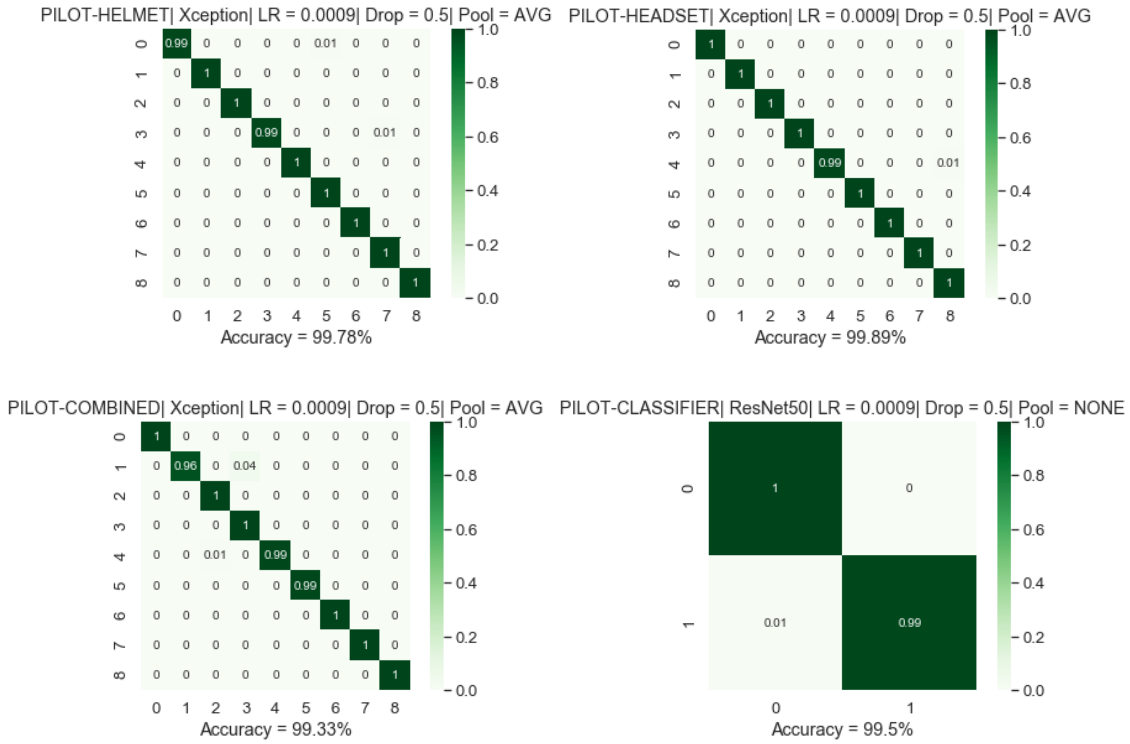

Figure 60. Confusion matrices for pilot simulator models

The final network to look at was the helicopter side network. This network was trained using the same combination of hyperparameters as the pilot and copilot head gear classifier models. The accuracy and loss of this network is shown in Table 15.

Table 15

Summary of accuracy and loss for helicopter side models

Network	**Training Accuracy**	**Training Loss**	**Validation Accuracy**	**Validation Loss**	**Test Accuracy**
HelicopterSide_2Class	100%	0.000001	100%	0.000001	95.5%

The model had extremely high accuracies with 100% in both training and validation meaning that the features for determining which side of the cockpit the video is on are fairly simple to learn. The confusion matrix for this model is displayed in Figure 61.

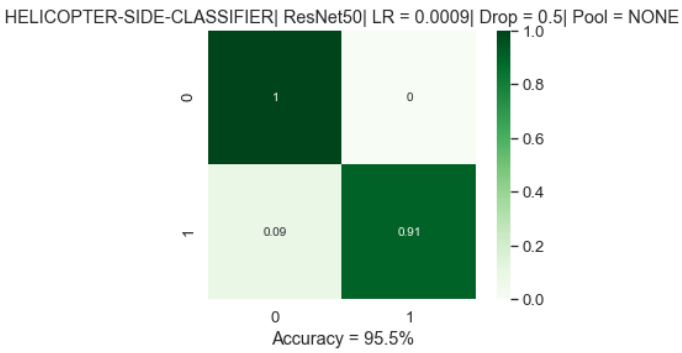

Figure 61. Confusion matrix for helicopter side simulator model

The initial results on the simulator data show that a working model with high accuracy was achieved for each of the nine models within the final algorithm structure. A few correctly classified images from the test sets are shown below in Figure 62 and Figure 63. The text in white is the prediction from the headgear classifier, the green/blue text is the headset/helmet prediction respectively, and the red text is the combined prediction. The helicopter side prediction, point of view, and frame number are printed in yellow at the bottom of the images.

Figure 62. Correctly classified pilot frames

Figure 63. Correctly classified copilot frames

These images demonstrate the deep learning algorithm's ability to correctly predict the head pose of pilots and copilots regardless of whether a face is present in the frame. The algorithm also consistently predicts the correct helicopter side as well as the head gear worn by the pilot and copilot. Figure 62 and Figure 63 also display examples of the combined predictions in red being consistent with the headset/helmet predictions in green/blue. When both head pose predictions are the same, it gives added confidence that the prediction is correct.

While the examples in Figure 62 and 63 quite clearly demonstrate the algorithm's success, there are also certain conditions that can cause the algorithm to struggle with making accurate head pose predictions.

Figure 64. Incorrectly classified pilot frames

The top left image of Figure 64 demonstrates the most common reason for any image-based deep learning algorithm to fail: occlusion. The pilot's hand is adjusting the camera and therefore slightly obscuring the view that the camera has of his head. An occlusion that causes a portion of the image to be blocked will almost always cause issues with a deep learning solution that consists of image data, regardless of the specific application. While the first image shows an error due to occlusion, the top right image in Figure 64 shows a misprediction due to image quality. The pilot is moving their head very quickly from left to right and this blurred image causes the algorithm to have a poor headset prediction.

The bottom left image in Figure 64 shows the helmet prediction as "Up" when the subject is looking down. This misclassification is due to the limited amount of data available for training. There were most likely not very many images in the training data that resembled this one, so the network will have a difficult time classifying these images in the test set. The final image in the bottom right of Figure 64 shows the helmet prediction and combined prediction as "Up_Left" and "Down_Left" when the pilot is just looking left. While these predictions are technically incorrect, it is important to remember that the algorithm was created for estimating head positions so that analysts and accident investigators can interpret the data. That being said, there is still information available in this last image that will describe the general position of the head even if the predictions are not exactly correct.

The problem of occlusion will always be an issue in an image-based deep learning solution. However, the majority of the remaining issues discussed can easily be resolved once more data is collected. As stated previously, the current datasets primarily consist of

simulator data that was created in a controlled environment. When more labeled examples with more variations are included during training, the algorithm should begin to learn more features of the input data and become more generalizable to new data and to specific outlier scenarios.

4.3 Deep Learning Algorithm Results (Generalized)

As an additional experiment, the deep learning models that were trained on simulator data only were used to evaluate the first 10,000 frames of the FAA Flight Dataset. The total number of frames belonging to each of the nine classes is shown in Table 16.

Table 16

Total number of frames belonging to each class in the FAA Flight Dataset

Class	**Total Frames**
(0) Down	440
(1) Down_Left	187
(2) Down_Right	402
(3) Left	1348
(4) Right	1155
(5) Straight	4280
(6) Up	1552
(7) Up_Left	333
(8) Up_Right	302

This experiment was conducted to observe the generalizability of the simulator models to a different set of data and observe whether or not the high accuracies from the simulator carry over to the real world. The algorithm performed well on this new data in some cases but the overall performance was much worse than on the simulator test data. This is somewhat of an expected result because the simulator images are different than the real flight video (Figure 50). The nine class combined network had an accuracy of 35.82%

on the real flight video and the nine class helmet network had an accuracy of 42.67%. The confusion matrices for the two nine class networks show these results in Figure 65.

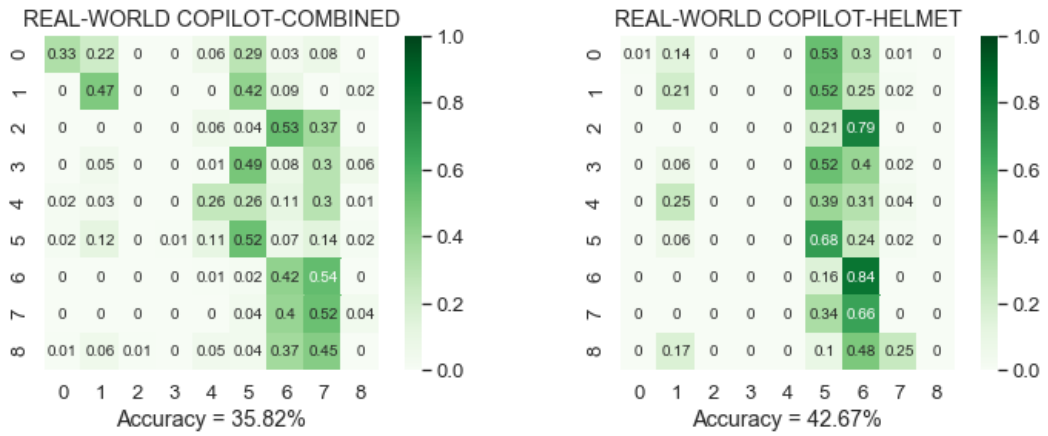

Figure 65. Confusion matrices of real world data evaluated by simulator models

Both models seem to be favoring classes 5, 6, and 7 with almost no predictions in classes 2, 3, 4, and 8. Two examples of incorrectly classified images are shown in Figure 66.

Figure 66. Real world images incorrectly classified by the simulator models

The helicopter side classifier was the only model that generalized well to this new data while the head gear classifier and both the combined and headset/helmet models struggled to provide accurate predictions. The change of camera angle and background were most likely the main causes for these decreases in overall accuracy.

To solve this problem, labeled images from the FAA Flight Dataset were added to the simulator training images using the method described in Section 3.4.4. With this new data available, the four models outlined in Figure 67 were retrained with these new images included during training. The CPHeadset_9Class model was not retrained because there was no real world headset image data available.

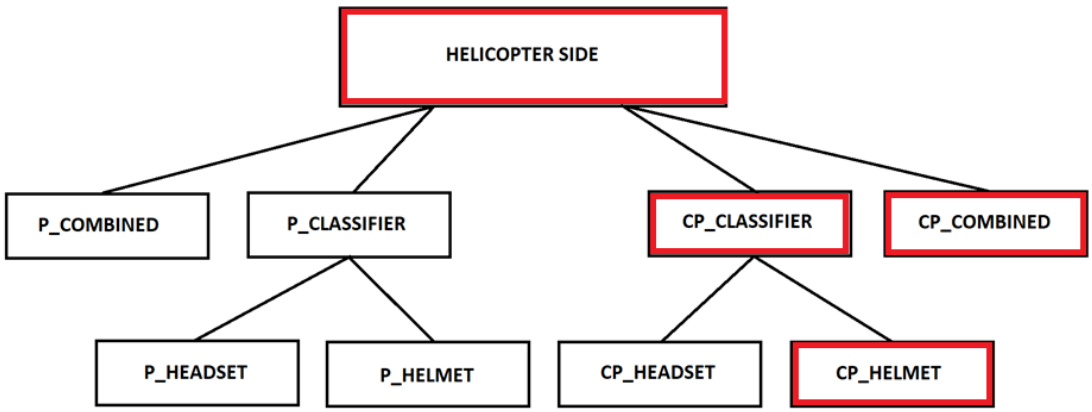

Figure 67. Retrained models with real world images included

The training, validation, and test accuracies of these new models along with their loss are shown in Table 17.

Table 17

Summary of accuracy and loss for generalized copilot models

Network	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Test Accuracy
CPHelmet_9Class	99.52%	0.01521	99.65%	0.00933	99.65%
CPCombined_9Class	99.81%	0.00577	99.81%	0.00685	99.69%
CPClassifier_2Class	100%	0.00001	100%	0.00001	100%
HelicopterSide_2Class	100%	0.00001	100%	0.00001	100%

These results show that both nine class networks performed with about the same accuracy compared to when they were trained and tested on the simulator data only. However, there is a slight increase in loss compared to the values in Table 13. The helicopter side model and head gear classifier model were both 100% accurate in all cases and this can be attributed to the fact that the features of the image that depict the correct helicopter side and depict the difference between a helmet and a headset are easy to learn. The confusion matrices for these new models are shown in Figure 68.

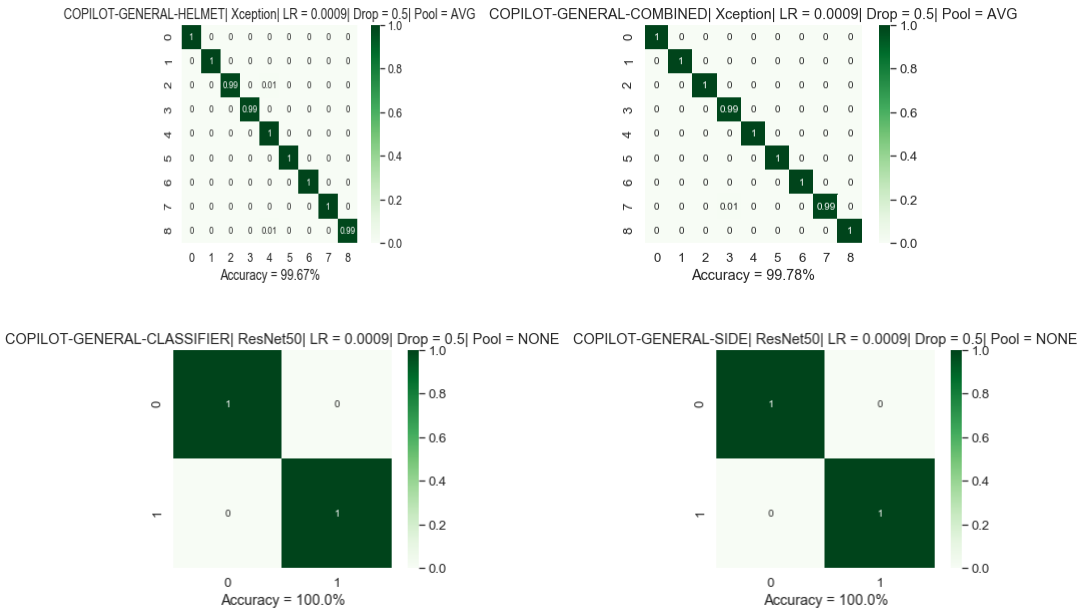

Figure 68. Confusion matrices for generalized copilot models

As shown before, the network architecture, learning rate, dropout rate, and pooling are displayed above each confusion matrix, and the accuracy is displayed below. With the addition of these real world copilot images to the training data, the models improved their generalization to the real world dataset. The nine class combined network had an accuracy of 91.49% and the nine class helmet network had an accuracy of 84.18%. The overall accuracies of the nine class networks after this change are shown in Figure 69.

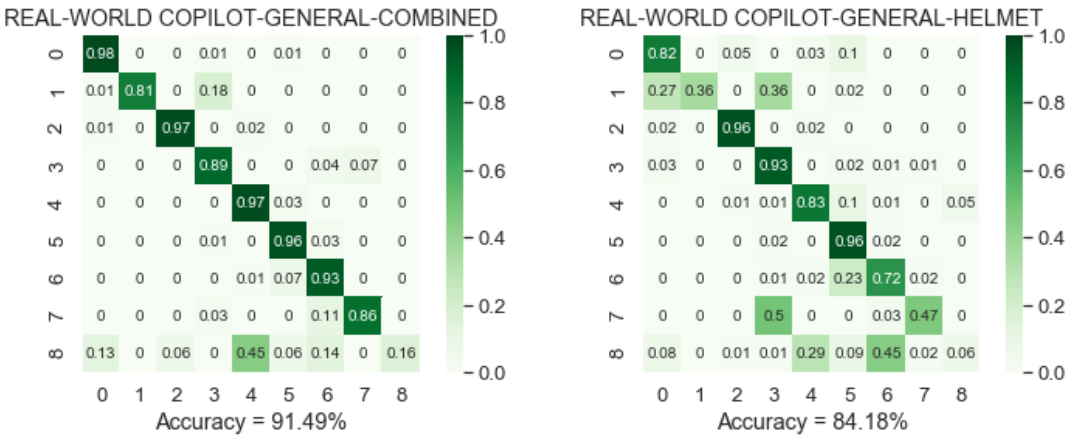

Figure 69. Confusion matrices of real world data evaluated by generalized models

The combined model with real world data outperformed the simulator-only model by about 55% and the new helmet model saw an increase in accuracy of about 40%. The generalized combined model also performed better than the generalized helmet model by about 7%.

While both models do well at classifying images from most classes, they also both struggle to classify images belonging to class 8, “Up_Right”. Similar to before, this was caused by lack of data in this class. From the entire 30976 images from the real world copilot video, only about 600 fell into class 8. For that reason, the network had a hard time

classifying these images simply because it hadn't seen many images from that class during training. A few examples of correctly classified images are displayed in Figure 70.

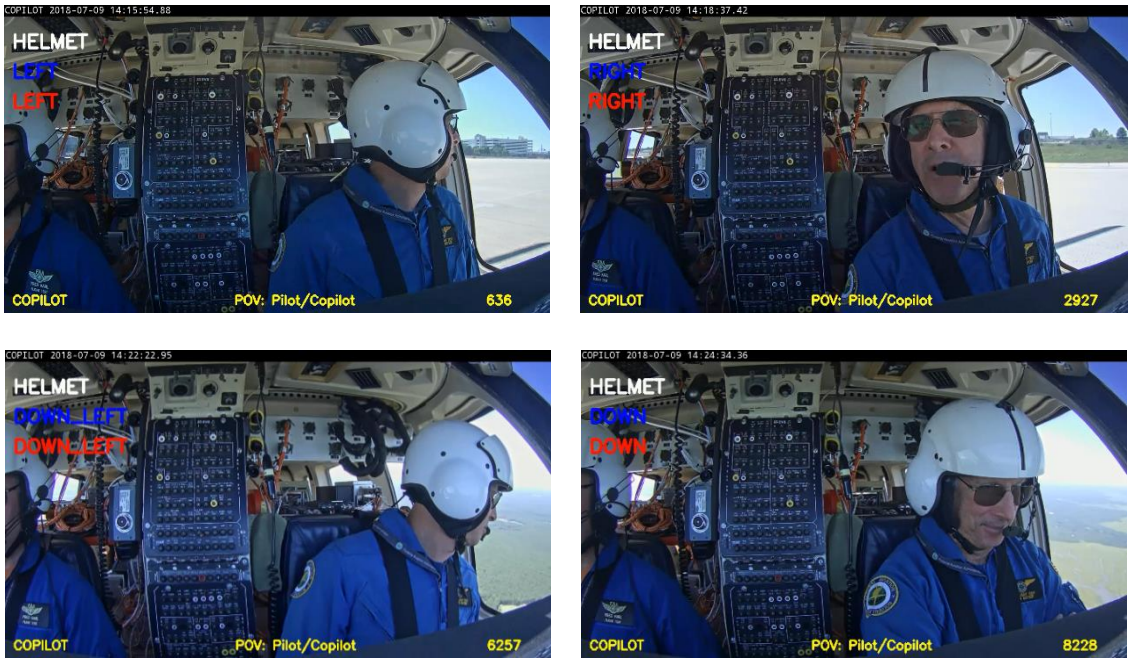

*Figure 70. Real world images c*orrectly classified by the generalized models

Alongside more accurate head pose predictions, the new head gear classifier correctly detected a helmet in all 10,000 images compared to only detecting 1836 images before. By adding just a few labeled images from the real world dataset to the training data, the models were able to generalize well even though the majority of the training images were created in the simulator.

4.4 Comparison of Hybrid Algorithm and Deep Learning Algorithm

The motivation of this research was to determine the head position of helicopter pilots and copilots given onboard cockpit videos of real flight video data. That being said, the final comparison of the algorithms discussed in this thesis is performed using the

accuracies that were calculated on the FAA Flight Dataset. The accuracies of the four algorithms are summarized in Table 18.

Table 18

Overall accuracies of both algorithms on the FAA Flight Dataset

Algorithm Name	**Accuracy**
Hybrid Algorithm	46.01%
Hybrid Algorithm with Compensator	55.26%
Simulator-Only Deep Learning Algorithm (Combined)	35.82%
Generalized Deep Learning Algorithm (Combined)	91.49%

The algorithm that performs the best is clearly the generalized deep learning algorithm with an accuracy of 91.49%. However, it is interesting to point out that the deep learning algorithm without the real world data included during training actually performs worse than both versions the hybrid algorithm. That being said, this continues to emphasize the point that a large amount of labeled data that adequately represents the test data is required for the deep learning algorithm to perform well on real flight videos.

If a sufficient amount of labeled data is available, there is no doubt that the deep learning solution will provide the most accurate head pose predictions when compared to the hybrid computer vision algorithm. However, creating a ground truth dataset or labeling any real world flight data can be challenging in this specific application because it is a labor-intensive process that is difficult to automate and has the potential to introduce human error. However, the added time needed to create labeled data is well worth the overall accuracy provided by the deep learning algorithm. In addition, the purely deep learning solution requires no preprocessing of the video data, and is very easy to implement after the training of the models has been completed.

On the other hand, the main benefit of the hybrid algorithm is that it does not require ground truth data in order to output head pose classifications. This makes the hybrid algorithm a possible solution if time is more important than accuracy. The hybrid algorithm does take some time to calibrate to each video but this time is negligible when compared to the amount of time it takes to manually label thousands of images. Since this head pose information will be used for incident/crash analysis however, it is very important that the algorithm provide accurate predictions in all situations, especially in the extreme angle case. For that reason, the generalized deep learning algorithm is the best choice when looking for accurate head pose predictions in the presence of excessive cockpit background, extreme head positions, and added noise from the pilot's operational equipment.

Chapter 5

Conclusions

The final chapter of this thesis reiterates the outline of the paper and summarizes the overall accomplishments of this research. The motivation and requirements of this research project are reviewed and a brief discussion of future work and research recommendations is also included.

5.1 Thesis Review

The first chapter provides an introduction to the problem and the motivation for creating a head pose estimation algorithm. The second chapter discusses the technical background knowledge needed to understand the computer vision and deep learning techniques used in this thesis. The third chapter explains the approach and methodology behind the creation of the hybrid algorithm and a description of the process of training and evaluating the different deep learning networks. The fourth chapter contains the results of both the hybrid computer vision algorithm and the final structure of the deep learning algorithm.

5.2 Research Accomplishments

The goal of this research was to automate post flight video processing and provide safety analysts or accident investigators with data on where a pilot was focused during any particular moment of any given flight. It was required that the head pose estimation algorithm be kept simple and low cost so that more helicopter operators would participate in Flight Data Monitoring programs, and therefore provide more information for analysts to use in the future. Both requirements were met using a deep learning algorithm whose only input was video data. The results show that a combination of deep learning models

can be trained to identify not only pilot/copilot head positions, but also to gather information about the pilot/copilot's head gear and which side of the cockpit the video took place. The objectives from the first chapter of this thesis are restated and the research accomplishments of this thesis are listed below:

1. *Create a low cost method for accurately determining the head positions of helicopter pilots and copilots by utilizing post-processing of cockpit video data.*

- Multiple methods were explored including a hybrid algorithm that utilizes both computer vision and deep learning techniques, and a purely deep learning algorithm that uses a total of nine deep learning models to gain information about head positions, pilot/copilot equipment, and cockpit side. The only input to both algorithms is video data which can easily be collected using inexpensive, off-the-shelf video cameras.

2. *Explore the possibility of implementing a classical computer vision algorithm that does not require labeled ground truth data to be available.*

- A hybrid head pose estimation algorithm was created that uses classical computer vision techniques of face detection, facial landmark annotation, and the pinhole camera model to calculate angles for classification. This algorithm performed well for frontal facing poses but struggled to classify head positions at extreme angles. However, a compensation method was introduced that aided to increase the number of correct classifications at extreme angles. This method was still not the best approach, but it was the main driving force for creating a labeled ground truth dataset for a purely deep learning approach.

3. *Create a sufficiently large, labeled ground truth dataset that consists of images of helicopter pilots and copilots with various head positions.*

- Working closely with the FAA, a dataset consisting of just under 200,000 labeled images was created in a Sikorsky S76D simulator for helicopter pilot and copilot head positions. The dataset consisted of nine total classes covering the full range of head poses and was used to train the deep learning models for the purely deep learning algorithm. Up to this point, there was no labeled head pose data for helicopter pilots available, and this FAA Simulator Dataset is one of the major contributions of this thesis.

4. *Train multiple deep learning models for determining head positions of helicopter pilots and copilots using the labeled ground truth dataset.*

- A total of nine models were successfully trained using the FAA Simulator Dataset. A two class model was trained to learn what side of the cockpit the video took place. Four networks were then trained for both the copilot and the pilot. A single two class network was trained to learn whether the pilot/copilot was wearing a helmet or headset. In addition, three nine class networks were trained for predicting head pose. The first model was trained on helmet and headset images combined, the second model was trained on headset images only, and the final model was trained on helmet images only. Each frame of the test video had four accompanying predictions: helicopter side, headgear classifier, combined head pose prediction, and headset/helmet prediction. The separate helmet and headset models were used to add robustness to the combined model since data was limited at the time. This algorithm structure

was able to correctly classify head positions for 91.49% of images from the first 10,000 frames of a real world flight video.

5. *Discuss the advantages and disadvantages of the purely deep learning solution.*

- The main disadvantage of the purely deep learning algorithm is the amount of data required to train a working model, and the time required to collect the labeled data. It can be very difficult to collect labelled data automatically resulting in the majority of data being collected by hand. Although time can be a disadvantage, there are a far larger number of advantages that accompany the deep learning solution. Contrary to the hybrid algorithm, the deep learning algorithm did not need a face to be present in the frame in order to output a head pose prediction. It also did not require any preprocessing of the video data and could be easily adapted to multiple cockpit scenarios without significant calibration. The deep learning models can also be easily improved as more data becomes available. Once enough data is collected with enough variations, the deep learning models will eventually be able to generalize to any real world flight video.

5.3 Research Recommendations and Future Work

As with any deep learning model, improvements can be made as more data becomes available. The current state of the models can be used to semi-automatically label new test videos to gather more training data. Once more data is collected, the new images can be added to the existing datasets and a new set of models can be trained.

Using the process of transfer learning allows for new models to be initialized with the same weights of the current version of the model, drastically reducing the amount of

time it will take for the network to learn the new features from the newly added images. Rather than relearning the early level features of all the images, transfer learning allows the network to begin learning the more complex features of the new images immediately.

Additionally, as more data becomes available, the less necessary the separate helmet/headset models become. At the time of this research, the amount of available data was limited so the added headset/helmet models provide some assurance to the combined head pose model prediction. However, once the amount of data becomes sufficiently large and the network is able to generalize well to all types of helicopter images, these added models should be removed to cut down on processing time. This simplifies the output of the algorithm, resulting in only two predictions per frame: the correct side of the cockpit, and a single, nine-class head pose prediction.

References

[1] National Transportation Safety Board 2017-2018 Most Wanted List of Transportation Safety Improvements, 2017-2018.

[2] B. Verna. Flight Data Monitoring Systems and Non-Required Safety Enhancing Equipment. HEMS Conference, Federal Aviation Administration, 2009.

[3] A. Payan, A. Gavrilovski, H. Jimenez, D. Mavris. Improvement of Rotorcraft Safety Metrics Using Performance Models and Data Integration. Journal of Aerospace Information Systems, Vol. 14, No. 1, pp. 26-39, DOI: 10.2514/1.I010467.

[4] E. Murphy-Chutorian and M. M. Trivedi. Head pose estimation in computer vision: a survey. PAMI, 31(4):607-626, 2009.

[5] S. Teulyakov, R. Vieriu, S. Semeniuta and N. Sebe. Robust real-time extreme head pose estimation. University of Trento Italy, 2014.

[6] S. Mallick. 2016, December 6. *Histogram of Oriented Gradients* Learn OpenCV. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/

[7] R. Gandhi. 2018, June 7. *Support Vector Machine – Introduction to Machine Learning Algorithms* Towards Data Science. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[8] R. Khandelwal. 2019, November 30. *SSD: Single Shot Detector for object detection using MultiBox* Towards Data Science. Available: https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca

[9] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou and M. Pantic. A semi-automatic methodology for facial landmark annotation. Computer Department Imperial College London, UK, School of Comptute Science, University of Lincoln, U.K., EEMCS, University of Twente, The Netherlands, 2013.

[10] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou and M. Pantic. 300 Faces in-the-Wild Challenge. Computer Department Imperial College London, UK, School of Comptute Science, University of Lincoln, U.K., EEMCS, University of Twente, The Netherlands, 2013.

[11] K. Hata and S. Savarese. Topic: “Course Notes 1: Camera Models.” CS231A: Computer Vision, From 3D Reconstruction to Recognition, Stanford University, Stanford, California., Mar., 2018.

[12] R. van den Boomgaard. Topic: “1.2 The Pinhole Camera Matrix.” Lecture Notes Image Processing and Computer Vision, University of Amsterdam, Amsterdam, The Netherlands., 2017.

[13] Y. Jia, Topic: “Homogenous Coordiantes.” CS577: Problem Solving Techniques for Applied Computer Science, Iowa State University, Ames, Iowa,. Dec. 2019.

[14] K. Simek, 2013, August 13. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix* Sightations. Avaialble: http://ksimek.github.io/2013/08/13/intrinsic/

[15] G. Slabaugh. “Computing euler angles from a rotation matrix,” unpublished.

[16] A. Ng. Class Lecture, Topic: “Neural Networks and Deep Learning.” Deep Learning Specialization, Online. Accessed, Aug., 2018.

[17] A. Ng. Class Lecture, Topic: “Improving Deep Neural Networks.” Deep Learning Specialization, Online. Accessed, Aug., 2018.

[18] A. Dertat. 2017, November 8. *Applied Deep Learning – Part 4: Convolutional Neural Networks* Towards Data Science. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[19] A. Ng. Class Lecture, Topic: “Convolutional Neural Networks.” Deep Learning Specialization, Online. Accessed, Aug., 2018.

[20] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.

[21] V. Fung. 2017, July 15. *An Overview of ResNet and its Variants* Towards Data Science. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[22] C. Szegedy *et al*., "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 1-9.

[23] S. Tsang. 2018, September 25. *Review: Xception – With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)* Towards Data Science. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

[24] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 1800-1807.

[25] N. Gourier, D. Hall, J. L. Crowley. Estimating Face Orientation from Robust Detection of Salient Facial Features. Proceedings of Pointing 2004, ICPR, International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK, 2004.

[26] S. Mallick. *Head Pose Estimation Using OpenCV and Dlib* Learn OpenCV. Available: https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/