UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

REAL-TIME GESTURE RECOGNITION WITH MINI DRONES

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

TANER B. DAVIS
Norman, Oklahoma
2018

REAL-TIME GESTURE RECOGNITION WITH MINI DRONES

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Amy McGovern, Chair

Dr. Deborah Trytten

Dr. Andrew Fagg

Acknowledgements

I'd like to thank Dr. Amy McGovern for serving as my committee chair, my life advisor, and my strength and reason for moving to this topic from a previous research project we had worked on. This project reignited my drive for working in the computer science field. I had such a blast working on it as it afforded me the ability to ask someone if I could "collect your hands real quick." So thanks to my frequent hand models: Ryan Lagerquist, Carmen Chilson, Khaled Nimer, Will Booker, Azadeh Gilanpour, Amanda Burke, David Harrison, Eli Jergensen, Kate Avery, and William McGovern-Fagg. Your hands carried this project to where it is today, so give yourself a hand and know your hand had a great hand in all of this. :)

A special thank you to Tabatha Dickinson for helping me slog through and trim the literal tens of thousands of raw photos, and then sort them by person, gesture, and environment. I would have cried having to do it alone, so I'm glad we could cry together.

Table of Contents

List Of Tables

List Of Figures

Abstract

Drones are being used worldwide to perform many functions - medical deliveries, video recording, and surveying to name a few. Their growing presence is paralleled by the growing world of Machine Learning (ML). This thesis presents a synthesis of machine and machine learning to create reactive agents in the form of mini-drones. Two prominent companies, DJI and Amazon, are patenting and selling drones that recognize hand gestures, but do not release their methods on how they made their intelligent drones. Deep learning and convolutional networks are used to train agents that control mini-drones based on hand gestures. This thesis details the process from collecting data to deploying an agent. Real-time gesture recognition agents are deployed in connection with a mini-drone that detects four gestures: a pointer finger pointing up as well as down, two fingers in a "peace" v-shape sign, and a flat, upward facing palm. The dependency of environment in which training data is selected as well as the apparent lighting of the subjects is investigated to see which creates an agent that detects the four gestures more frequently. The end product is a gesture-intelligent drone reacting to hand gestures - the foundation for a complex physical language interaction.

Chapter 1

Introduction

DJI, an online quadcopter retailer and front runner in drone video recording and recreation, sells drones that come preloaded with gesture detection technology (Heliguy 2018). Amazon, a worldwide online retailer, has also submitted a patent for drones with gesture controls and facial recognition (Cuban et al. 2018). Both companies are developing and deploying drones with gesture recognition capabilities, but neither release their learning methods, datasets, or showcase the ability to learn additional gestures. This thesis demonstrates the process of creating an intelligent drone that can recognize hand gestures and respond appropriately.

Deep learning has proven powerful in performing astounding image classifications with items such as cards (EdjeElectronics 2018), bird roosts (Chilson 2017), and faces (Viola and Jones 2004). Deep learning with convolutional neural networks (CNN) can learn on many mediums like photos, a collection of time-bounded photos, or videos. Some of the most common uses of deep learning with CNNs are facial recognition in cellphones, posture detection with video games, and image classification on web searches. However, many of these methods take large amounts of time and memory to learn and classify an image, and many require multiple measuring apparatuses (Kaâniche 2009). This spurred the creation of resource conservative learning techniques and real time

classifiers like Single Shot Detectors (SSD) (Liu et al. 2016) and You Only Look Once (YOLO) (Redmon et al. 2016).

There are cases where it is urgent to identify an object as quickly as possible. Drones interacting with humans are becoming more frequent with delivery drones and drone-guided rescue efforts, so taking too long to identify a critical object like a trapped person during a search, or a "stop" gesture could cause harm to an individual (Cuban et al. 2018).

With these real time techniques came the creation of Inception Networks (Szegedy et al. 2015) and Mobile Networks (Howard et al. 2017), both networks being fast image classifiers. Both Networks have been shown to be effective at performing real time object detection (Szegedy et al. 2015; Sandler et al. 2018), so their ability to perform real-time object detection was tested on a 640 x 360 RGB video-streaming drone to identify four static hand gestures.

This thesis evaluates the Inception Network and MobileNets on a dataset of four static gestures created and curated as the Gesture Images For Training (GIFT) database (Davis 2018). This thesis contributes the first public gesture database, released on GitHub. GIFT is intending to be a benchmark dataset for intelligent gesture recognition. The thesis also compares the trade off of speed of identification through frames per second, with the percentage accuracy of each classification. This is done by paralleling gestures to drone commands in hopes to communicate with the drone what the user would like for it to do. These agents are the first attempt that I am aware of to publicly release hand gesture datasets and agents trained to run on a mini-drone with video-streaming capabilities.

Chapter 2

Background

Before describing Faster R-CNNs, SSDs, and their deep learning innovations, it is important to begin with describing artificial neural networks as they provide the foundation for CNNs and other advanced networks.

2.1 Artificial Neural Networks

In its simplest form, Artificial Neural Networks (ANNs) transform some numerical input into an output that serves to perform some action or produce a probability array. The ANN architecture can be generalized to have three parts or layers: an input, a hidden layer, and an output as seen in Figure 2.1. Layers are the building blocks of neural nets, as they take in numerical input, transform the data, then output the data into a more useful form for the next layer to use (Chollet 2018). ANNs do not usually come out of the box ready to use for a special purpose. They must be trained to be useful. Training consists of inputting a changing set of numbers through each network, slowly tweaking the network as it trains, until the user is satisfied with the networks' reported accuracy and losses.

The "Neural Network" part of the name comes from the connected units inside the mechanism called "neurons" (Rumelhart et al. 1985). The term neuron parallels the discrete building structures of the brain, which receive,

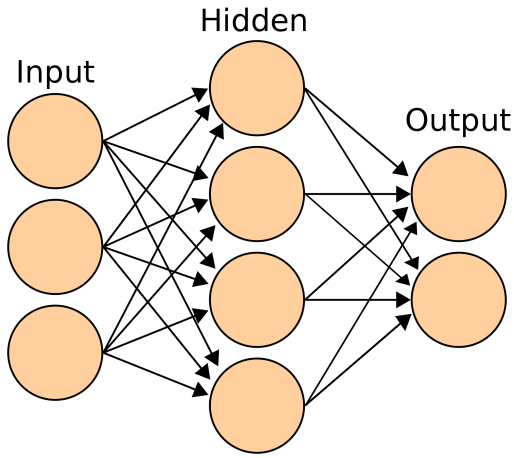


Figure 2.1: A single layered artificial neural network.

process, and output electrical signals for other neurons. Each artificial neuron functions in five steps as seen in Figure 2.2. The neuron first accepts input in the form of numbers where each number is multiplied by some weight value. Each neuron in each layer has their own weight, and this weight frequently changes as the network trains on more data. Second, these inputs are all summed together into a weighted sum. Third, this new result is passed into a pre-determined activation function. This activation function is either a threshold or a logistic function. Fourth, the output is the result of the activation function. For a threshold activation function, if the weighted sum is larger than an arbitrary threshold 'Q', then the output is a 1, and 0 otherwise. Finally, the neuron then passes on the output of the activation function to any other layer of neurons to which it is connected (Russell and Norvig 2016). The next neuron then repeats this process until the output layer is reached, where instead of passing an output directly to the user, a function is applied to the output layer to produce an array of probabilities or to perform some action. The abstraction of this process inside each layer is referred to as a "node." From hereon the term "node" will be used

as the entity that accepts inputs, transforms the input, and passes information into the proceeding layer.

Mathematically, a node looks like Equation 2.1 with inputs a_i, weights w_{ij}, activation function g, and output a_j:

$$a_j = g(w_{0j} + \sum_{i=1}^{n} w_{ij} a_i). \tag{2.1}$$

First, sum up the product of each input multiplied by its weight. Next, add the product of the bias (w_0) and its weight. This final sum then becomes the input to an activation function. For this thesis, I use Rectified Linear Units-6 also known as ReLU6 (Krizhevsky and Hinton 2010), an extension on the common ReLU activation function:

$$g(x) = max(x, 0). \tag{2.2}$$

ReLU6 simply limits the ceiling of the function to six and the floor to zero:

$$g(x) = min(max(x, 0), 6). \tag{2.3}$$

ReLU6 encourages models to learn sparse features of gestures earlier as all negative inputs become less important to the network, and large inputs are limited to 6, being considered just slightly more important than smaller positive values. This activation function is part of a general ReLU-n family where the function class is:

$$g(x) = min(max(x, 0), n). \tag{2.4}$$

Neural networks can take information in many different forms: a picture, a video, a spreadsheet, or a piece of music. The net does not handle the medium in its primary form. Instead, there is a layer of neurons that retrieve information from the medium to pass it through the networks. For pictures, it is very

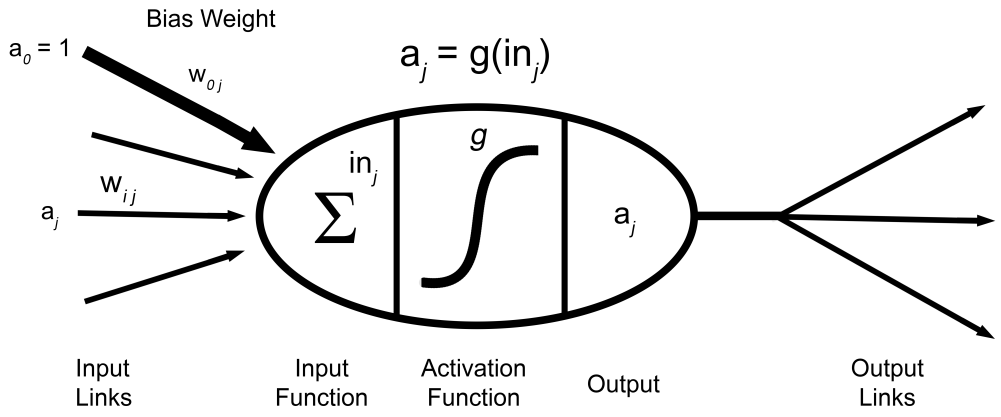


Figure 2.2: Dissection of a Neuron. Adapted from (Russell and Norvig 2016)

common to transform the image into an array of numbers where each number is a gray-scale pixel evaluation from 0, black, to 255, white. One can also pass in a matrix of numerical evaluations for RGB values for colored images. Instead of a pixel having a single value, it will have three values, a Red, a Green, and a Blue. As long as there is an algorithm to transform one's information from the world into some set of numbers, a neural net can work with it.

Once the information has been transformed, each node in the input layer gets some part (usually a single number) of the transformed information and passes it to the first layer of hidden nodes. A node must always send its information to at least one node unless it is in the output layer. It is not uncommon for a node to send its output to all nodes in the subsequent layer. A layer where all nodes pass their output to all nodes of the next layer is called "dense" or "fully connected" (Chollet 2018).

Where there are usually only one of each input and output layer, there can be zero or more hidden layers. The special case of zero hidden layers with an activation function defines the neural network to be a "Single Layer

Perceptron." Anything with at least one hidden layer is classified as a "Multi-Layer Perceptron" (Jantzen 1998).

Once the data has traversed through every layer, it is time for the neural network to produce something interpretable. This is where the Output layer comes into play. Here, one last calculation is performed to produce some array of probabilities or some interpretative output, such as a collection of ones and zeros to simulate pressing buttons on a game controller. When detecting objects, it is common to encode the output as a one-hot vector, an n-vector of zeroes with a single "1". The single one determines the class label of the image. However, probabilities at each node in the output layer can be reported. Each node reports the score of the input being one of the classes. Though, when summing all of the output nodes score together, this will sometimes not equal 1.0 or 100% - the image is a horse with probability 0.4 from one node, and is a dog with probability 0.2. In this instance, the softmax equation can be used to turn the output into a probability:

$$\sigma(a)_j = \frac{e^{a_j}}{\sum_{k=1}^{n} e^{a_k}}. \tag{2.5}$$

From the example above, the horse would be 0.55 and dog would be 0.45.

2.2 Convolutional Neural Nets

Convolutional Neural Nets (CNNs) are the current state-of-the-art in many vision tasks, especially object detection as CNNs introduce feature abstractions where every layer layer learns some feature of the images like edges and shapes (Chollet 2018) (Ciresan et al. 2011) . This works perfect as the goal of this thesis is to create an agent which is capable of object detection. They differ from ANN's one dimensional layers as each layer has a height, width, and depth

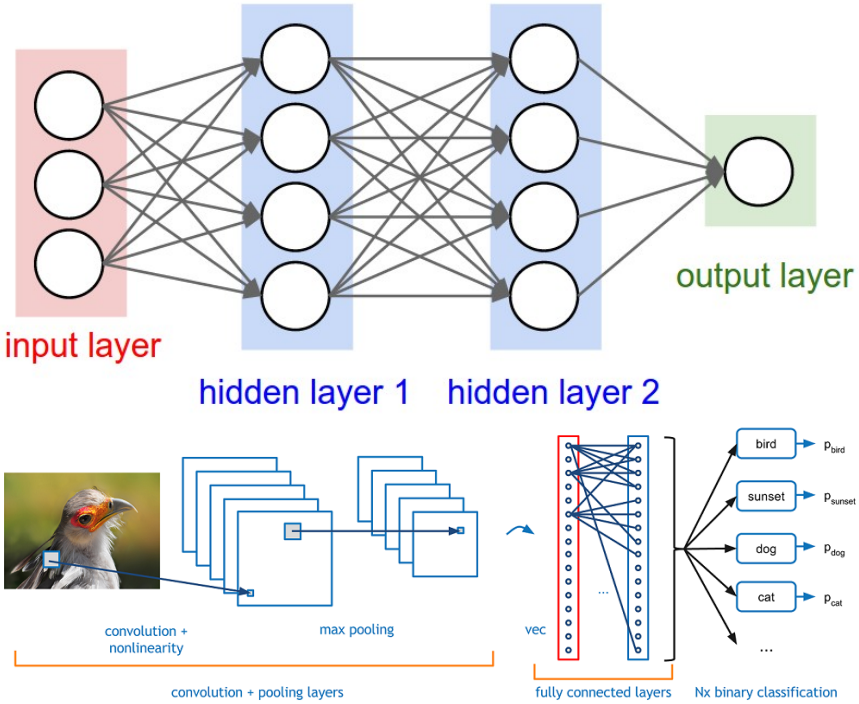


Figure 2.3: ANN architecture vs CNN architecture. The main difference is the CNN's introduction of convolutional calculations and pooling between successive layers. Top image from Karpathy 2016, bottom image from Deshpande 2016.

as seen in Figure 2.3. CNNs also take into account that each subsequent layer only connects to parts of the previous layer, minimizing the number of weights needed for each connection. This is largely beneficial to image processing, as most images use an RGB value at each pixel (Karpathy 2016). A CNNs output would typically be an n-vector of probability scores for each classification but any real-valued output is possible.

CNN nodes also share weights, rather than each node having its own weight. This is done by using sliding filters. This filter is smaller than the input, and moves by units called a stride length. When a filter moves over some part of the input, the subset of the input is known as the local receptive field, and it is

between the filter and this field that the convolution step occurs. The network then multiplies each of the weights in the filter by each of the variables in the local receptive field. This then makes a new entry in an output filter. Once the filter has traveled across the entire input, a feature map is produced. This process is demonstrated in Figures 2.4 and 2.5. This collection of dot product summations is called the Convolutional Layer.

After the convolutional layer, the network then reduces the output by selecting values from a filter of the input. This is called pooling. A common form of pooling is max pooling. The equation for this is:

$$\begin{aligned} y_{j,k} &= \max_{m,n}(x_{m,n}), \text{ where} \\ m &\in \{s \cdot j, ... \ s \cdot (j+1) - 1\}, \\ n &\in \{s \cdot k, ... \ s \cdot (k+1) - 1\}, \end{aligned}$$

and where s is the stride length. For each layer of output from the convolutional step, another filter is slid an arbitrary stride length over the input, and selects the largest number in the filter over the input. As this is done to every layer, the width and height of the feature maps shrinks, but no feature map is lost.

2.3 Finding Regions of Interest with R-CNNs

In object detection, it is not enough to say that there is a playing card or there is not a playing card in the image. The object must also be bound in the image and the network must declare that this region contains the playing card. CNNs do not vary their output layers in reaction to how many objects there are in an image. However, Ross Girshick et al. devised a way to include region proposals on which to use CNNs (Girshick et al. 2013). The general process is seen in Figure 2.7.

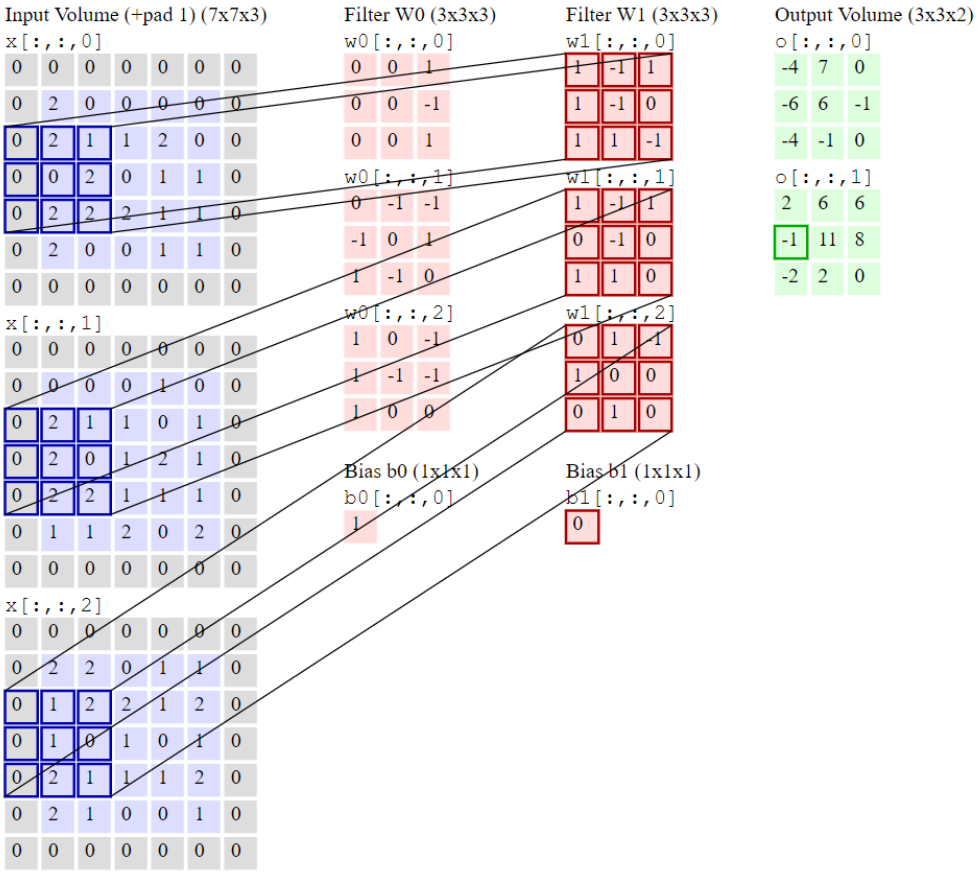


Figure 2.4: CNN Filter moving across an input. Filter W1, the red square, at each depth is multiplied against the input, the blue square, creating a new matrix. This matrix then has every cell summed together. The equation for this is $o_j = b1 + \sum_{k=0}^{2} x_{k+j} \cdot w_k$. The filters from W1 multiplied over the inputs and then summed are -1, -1 and 1 respectively. The three values are summed with the bias, b1 which is 0, and stored in the output matrix, the green highlighted cell. Here, $-1 + -1 + 1 + 0 = -1$. Image from Karpathy 2016.

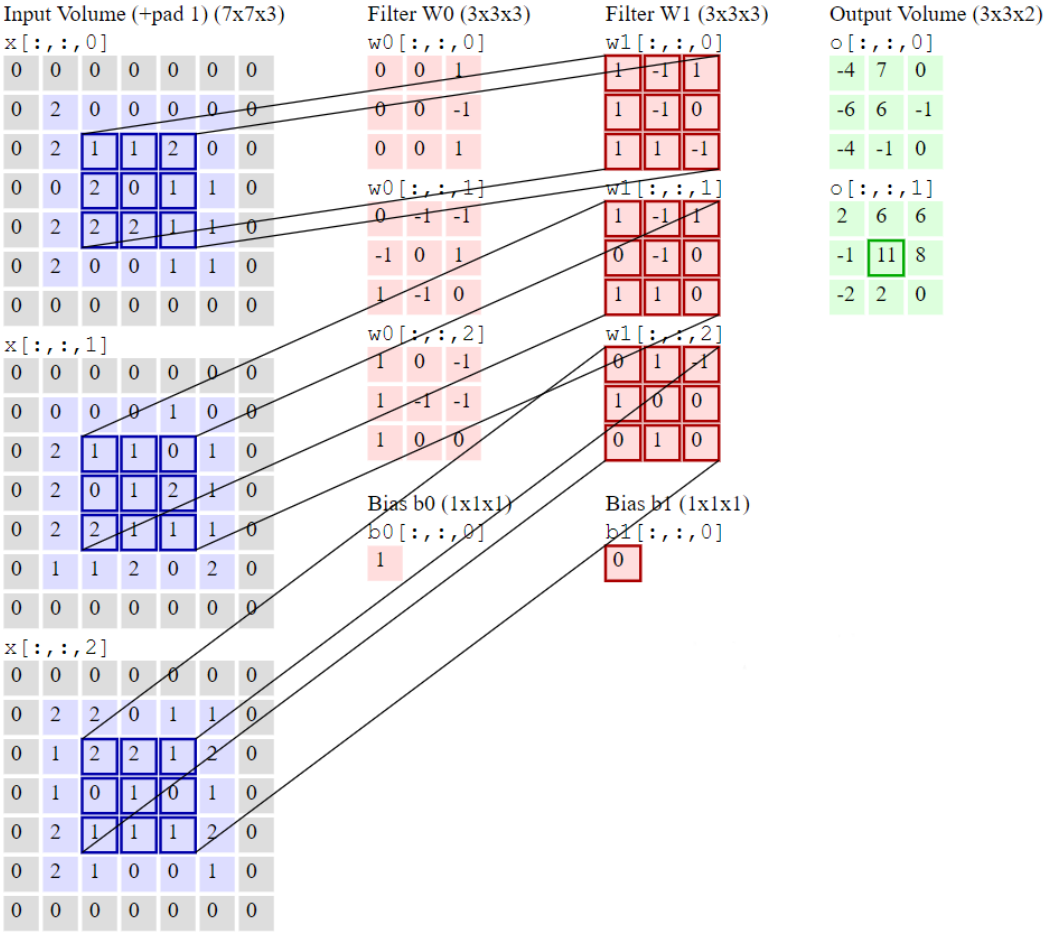


Figure 2.5: Here, with a stride of 2 from Figure 2.4, the filters are then multiplied against a new set of input, 2 units to the right of the previous filter. Image from Karpathy 2016.

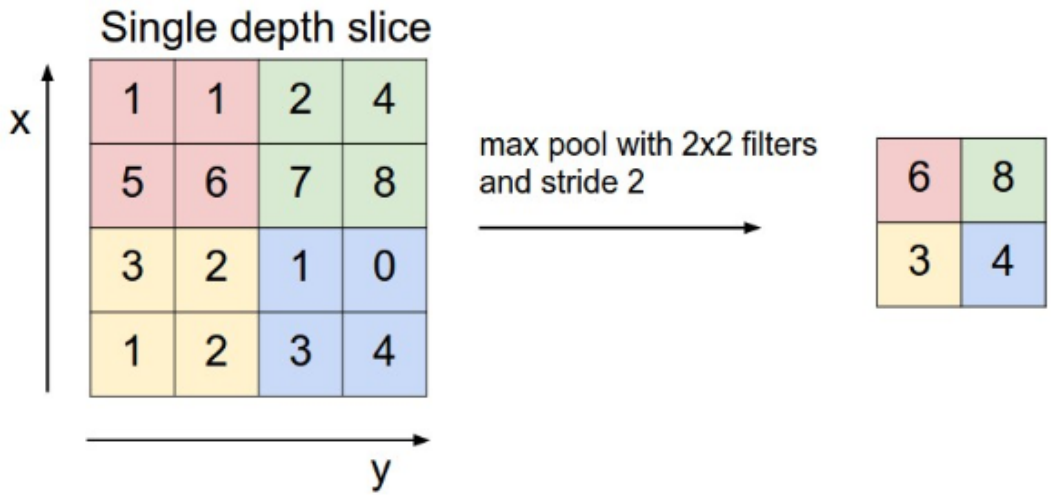


Figure 2.6: Max pooling input of 4x4 with a filter of 2x2 and a stride length of 2. The red square is the initial filter position. The green square is the filter's first stride moving two cells to the right. The yellow is the second stride, where, having met the edge of the matrix, the filter moves two cells downward and relocates to the left edge of the matrix. The Blue square is the final stride. The max value of each stride in order is 6, 8, 3, and 4. The results for the output matrix are in the order of the strides. Image from Karpathy 2016.

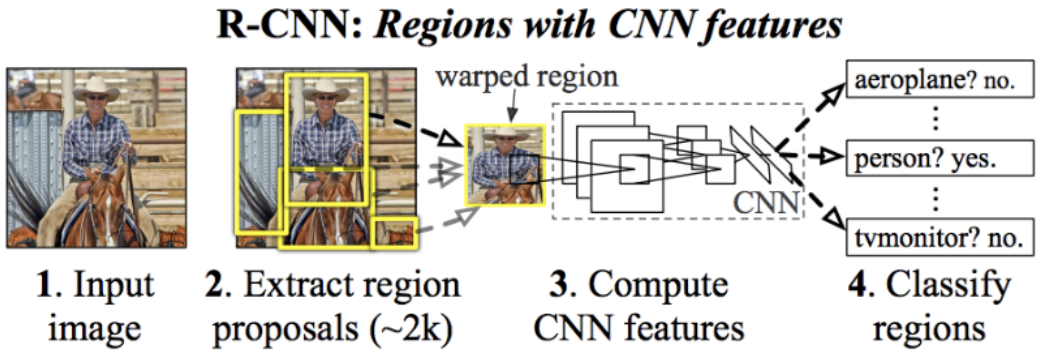


Figure 2.7: RCNN works by generating proposals on an image, running CNN on the proposals to create feature maps, and then classify each region. Image from Girshick et al. 2013.

R-CNN begins by generating about 2000 images (proposals) from the the main image using Selective Search (covered after this section). Each of the new images are then scaled into an n x n square, and become input for the CNN. Once the CNN creates a feature map for every proposal, the map is then fed into a Support Vector Machine to classify each proposal. The network also then guesses new values for the bounding box of the proposal to better classify the proposal from the CNN (Figure 2.8).

Selective search is an algorithm that determines where the proposals on an image should be, and is a solution to searching intelligently rather than exhaustively (Uijlings et al. 2013). First, the search needs to determine where objects might be in the images, so it divides the image into many regions using segmentation (Felzenszwalb and Huttenlocher 2004). In Figure 2.9, the individual RBG value of each pixel in the input image on the left is subtracted from each of its neighbors' RGB value, resulting in three subtractions times eight neighbors. When the differences of two pixels' red, green, and blue are all less than some threshold, the pixels are considered to be contained in the same component or region of the image. Once all pixels have been compared to their neighbors, a random color value is assigned to each component, and the output looks like the right image of Figure 2.9. From these segmentations, similarity of the components is determined. Similarity is determined as a function of two components in color, texture, size, and shape (Uijlings et al. 2013). If the similarity surpasses a threshold, the two components are combined into a single, larger component. Selective search continues comparing components until all remaining components are no longer considered similar to their neighbors. The process of merging components produces a hierarchy of successively larger regions that can be abstracted as objects in the image, similar to Figure

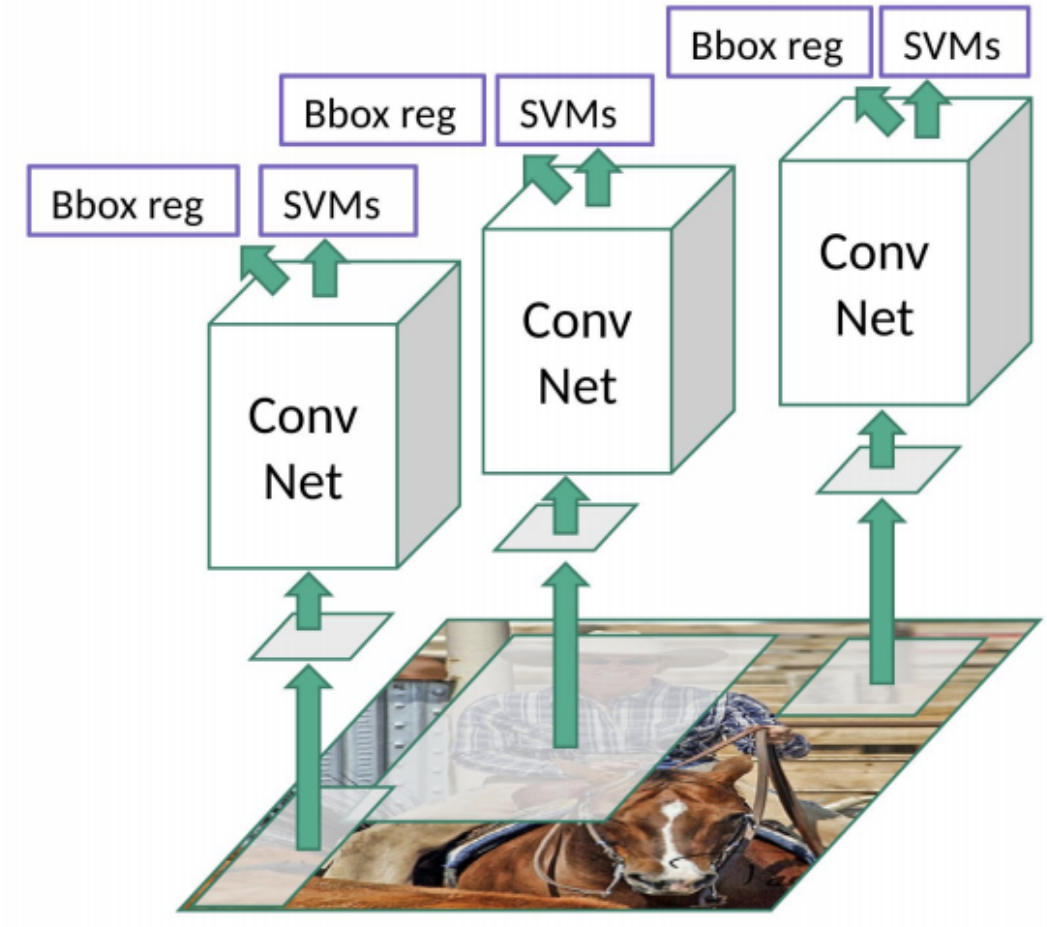


Figure 2.8: Each proposal's CNN pass the feature map to a SVM for classification, as well as suggests new values of the bound box on the proposal to better classify the proposal. Example: proposal contains a face, but only half of a face. The bounding box could be moved or the size changed to better contain the face. Image from Girshick et al. 2013.

Figure 2.9: Complete segmentation on an indoor scene. Each color on the output image (right) identifies a different component such as a face, a shirt, a nametag, or a book from the imput image (left). Image from Felzenszwalb and Huttenlocher 2004.

2.10. Using these objects, proposals are selected from the components and are used as inputs to train the RCNN.

2.4 Fast RCNN

RCNN still requires generating 2000 proposals per image, then using each proposal as input for a CNN. The algorithm for proposal selection is also not stochastic (Girshick et al. 2013). Ross Girshick, part of the team who created RCNN, improved the algorithm by removing the CNNs on every object proposal on the main input as seen in Figure 2.11(Girshick 2015). Now the image is directly input into a CNN to create the feature maps. Then, from the feature maps, proposals are found and scaled into a square. Each layer is then pooled with Max pooling to create the Region of Interest (RoI) layer. The RoI becomes the input for a Fully Connected (FC) layer. The FC layer outputs an

Figure 2.10: Segmentation (top row) into detection and hierarchy creation (bottom row). Image from Uijlings et al. 2013.

RoI feature vector (like the previous feature maps but only as an n-vector) and by using softmax, it produces a probability to predict the classification of the RoI. The ROI feature vector then serves as the input to a fully connected layer which then produce per-class bounding-box regression offsets (Girshick 2015).

This algorithm only requires convolutional operations once on the image instead of 2000 times, once on each of the proposals. Girshick found that with removing the thousands of convolutions, training time and testing time improved significantly (Girshick 2015).

Faster-RCNN (Figure 2.12) further improves on Fast RCNN by removing selective searches for region proposals and instead learns region proposals through the feature map of the image through a CNN (Ren et al. 2015). Just like Fast-RCNN the base image is input into a CNN. However, instead of finding proposals in the feature map using selective search, a fully convolutional network is used to output object proposals (the top left corner coordinate as well as the width and height) as well as an objectness score - the higher the score the more likely

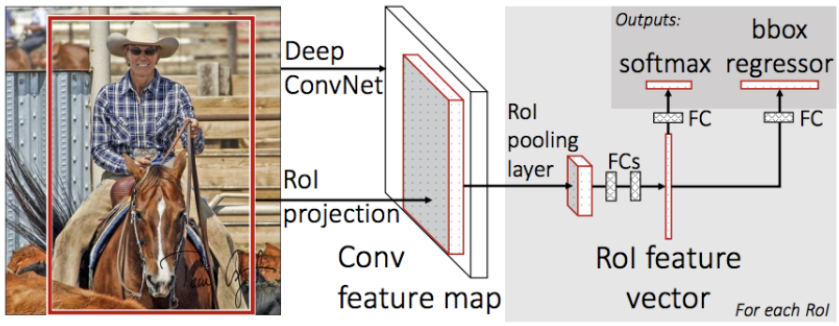


Figure 2.11: Improvement to RCNN by feeding the image to the CNN, find the proposals among the feature map, and scale the proposals into squares for a fully connect layer. Image from Girshick 2015.

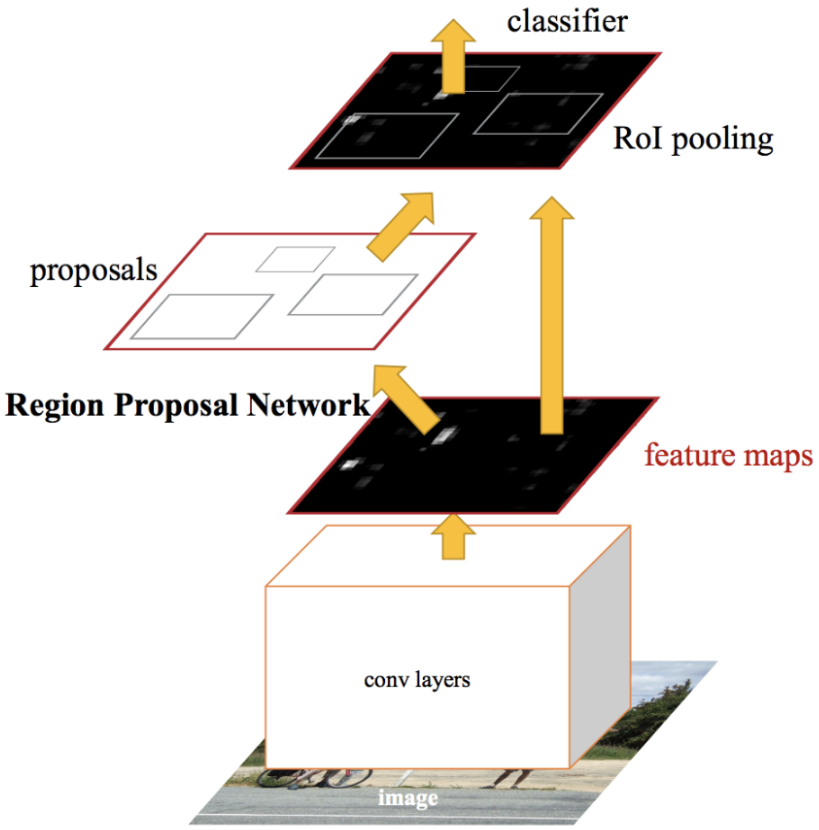
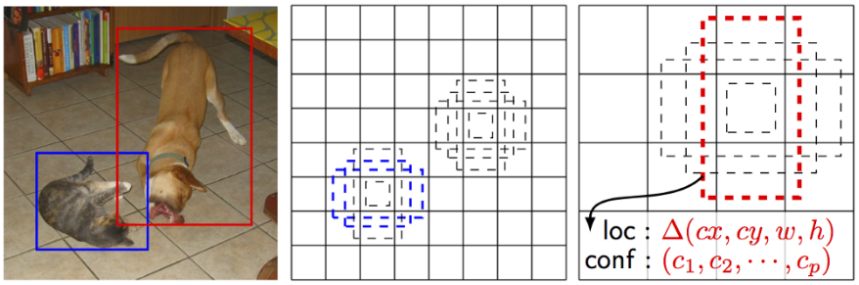


Figure 2.12: Faster-RCNN removes the selective search and instead learns the proposals as well as classifications. Image from Ren et al. 2015.

this region is an object instead of the background (Ren et al. 2015). The proposals are then pooled into an RoI pool that creates n x n squares, and the proposals are surveyed. The object is then finally classified with a probability vector, as well as the offset for the proposal boxes. Although algorithms are usually measured using O(), Shaoqing et al. measured a variety of algorithms in real-world circumstance on the same machine to provide practical numbers. They found that using their Faster-RCNN was even faster than Fast-RCNN, taking only 0.2 seconds to classify an image (Ren et al. 2015).

2.5 Single Shot Detectors

Single Shot Detectors (SSD) (Figure 2.13) increase the speed of the Faster-RCNN model by eliminating the region proposal step entirely, but with a slightly different architecture (Liu et al. 2016). SSDs first pass the image through multiple convolutional layers, creating feature maps of different sizes (20x20, 10x10, 6x6, etc.) On each feature map, a 3x3 convolutional filter evaluates a small set of bounding boxes with arbitrary sizes (Liu et al. 2016). The network then guesses the classification probabilities and the offset of the previous bounding boxes. With these multiple feature maps and bounding boxes of varying sizes, this helps detectors find large and small objects as smaller objects are found on a finer map with equally finer bounding boxes, and larger objects are found with larger maps and bounding boxes.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2.13: SSD creates multiple feature maps of varying sizes. The 8x8 finer map finds the cat in two of the bounding boxes, while the 4x4 larger map finds the dog using the same, scaled bounding boxes relative to the feature map dimensions. Image from Liu et al. 2016.

2.6 MobileNets V2 and Inception V2

MobileNets (Figure 2.14) trade-off accuracy for processing speeds, and are designed for embedded vision applications on mobile devices (Howard et al. 2017). They also introduce two multipliers: a width multiplier to make smaller CNN models as it reduces the input layer, and a resolution multiplier to control the input images resolution in the network. MobileNetsV2 (Figure 2.15) uses linear bottlenecks and inverted residuals to activate certain features earlier and architecturally uses fewer parameters using pointwise convolutions on CNNs from the original MobileNets architecture (Sandler et al. 2018).

The Inception network improved the CNN process by solving the issue of finding desired objects when the size and orientation of the object can vary greatly (like dog breeds). This would require determining the right filter size to find both large and small objects in an image. Having multiple filters naively stacked on each other is computationally expensive. Also, making a CNN larger though able to handle more parameters increases the possibility of overfitting

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2.14: The architecture of MobileNets. The innovative addition to object detection is the use of depthwise separable convolutions and pointwise convolutions. The table's top row is the first convolution on the raw image. The output of the first convolution is the input for the next row below in the table. "dw" in the table stands for depthwise. Table from Howard et al. 2017.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 2.15: MobileNetsV2 architecture. t is the expansion rate, c is the number of input channels, n is number of times the block is repeated, and s is the stride length. Table from Sandler et al. 2018.

on the data (Szegedy et al. 2015). The solution was to not stack the filters, but to place them beside each other. Each filter would then use the same input to perform convolutions, rather than the output of one filter's convolution being the input of another filter's convolution. This produced the architecture in Figure 2.16. Calling these Inception modules, they stacked nine together (Figure 2.17) and created the Inception network, or the "GoogLeNet" (Szegedy et al. 2015).

As with any deep network, the middle areas of GoogLeNet are prone to dying off (vanishing gradient). To prevent this, they added two intermediate classifiers to encourage the middle area to discriminate over dying (Szegedy et al. 2015). These classifiers add their loss multiplied by 0.3 to the total loss during training. During inferences, these intermediate classifiers are not used.

Inception V2 removed the larger filters entirely, finding that a 5x5 filter was 2.78 times more computationally expensive compared than a 3x3, meaning they could use two 3x3 filters and save resources. This produced a new architecture for a filter layer (Figure 2.18). Going further, they found that a 3x3 filter could become cheaper by a 1x3 filter and a 3x1 filter in succession (Figure 2.19). Finally, to reduce representational bottle-necking from deep nets reducing dimensions, they made the final filters in the module wider rather than deeper (Figure 2.19).

2.7 Mambo Mini-Drones

The Parrot Mambo Drone, Figure 2.22, is a mini-drone that is seven inches long and wide, and one and a half inches tall with the camera attachment. The drone camera was accessed through scripts from a public pyparrot package (McGovern 2018). The DroneVisionGUI script was used to connect with the

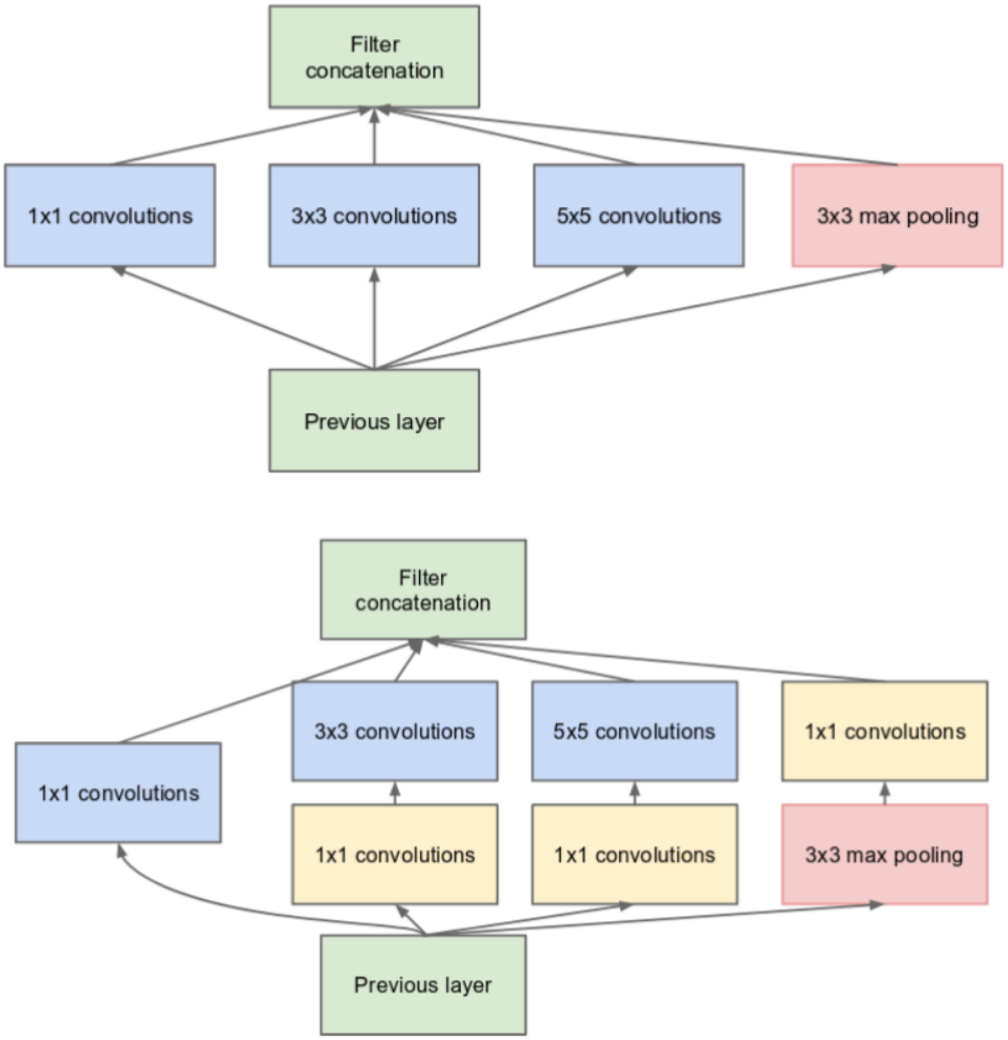


Figure 2.16: The naive architecture (top) of the Inception network, followed by the introduction of 1x1 convolutions (bottom) to make the large filters less computationally expensive. Image from Szegedy et al. 2015.

Figure 2.17: The entire Inception/GoogLeNet architecture. Image from Szegedy et al. 2015.

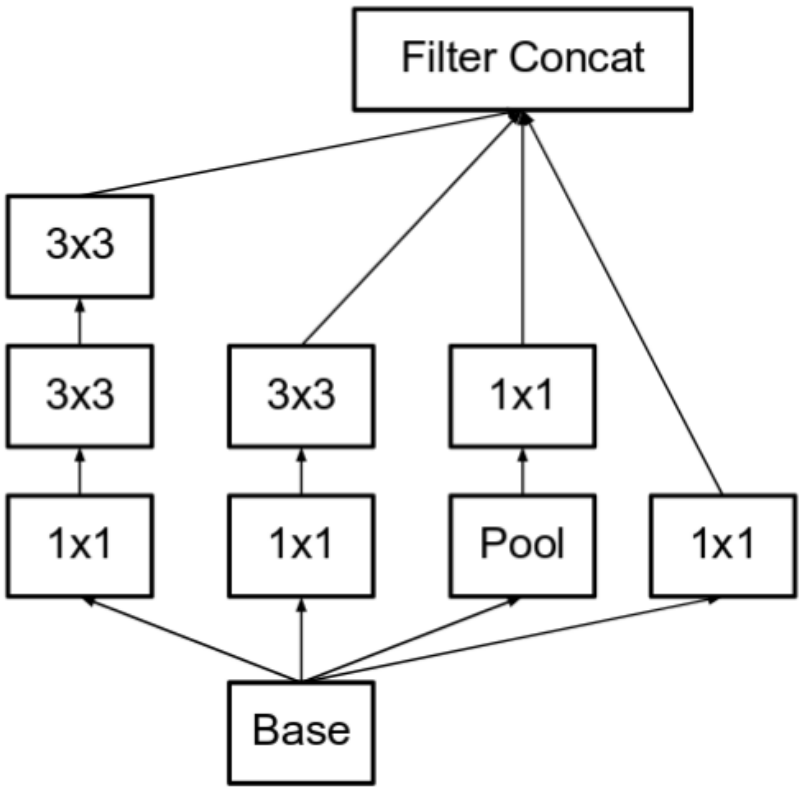


Figure 2.18: Inception V2 module replaced a 5x5 filter with two 3x3 filters as they were less computationally expensive. Image from Szegedy et al. 2015.

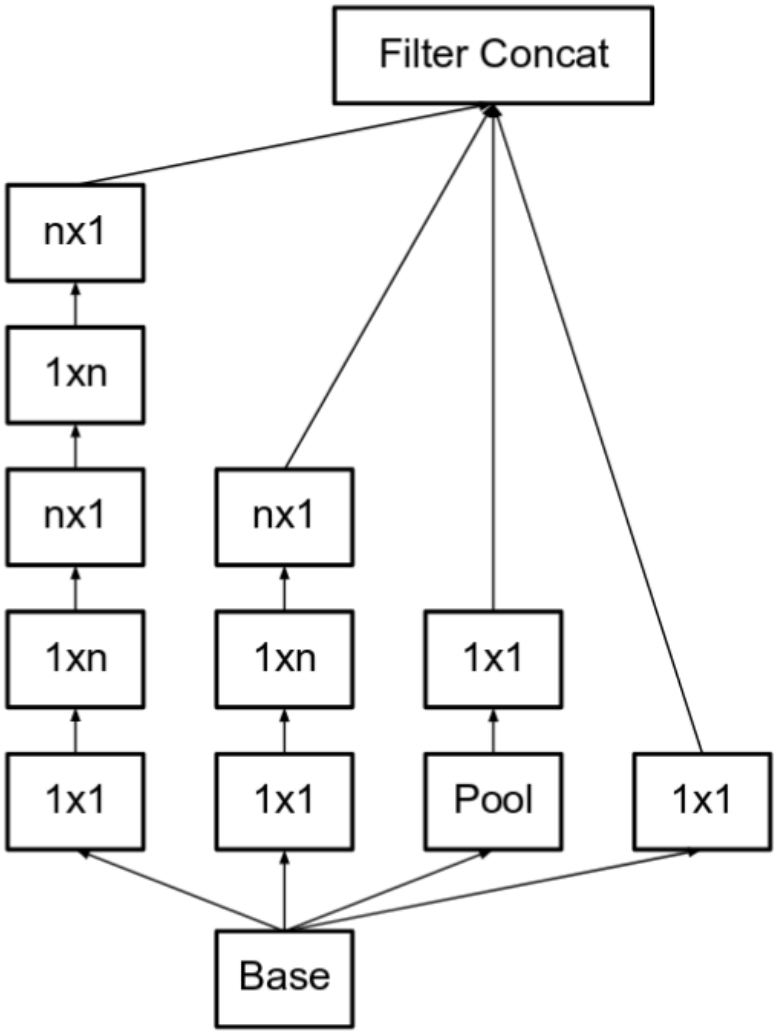


Figure 2.19: Inception V2 module replaced an nxn filter with a 1xn and nx1 filter. From Figure 2.18, n=3 here. Image from Szegedy et al. 2015.

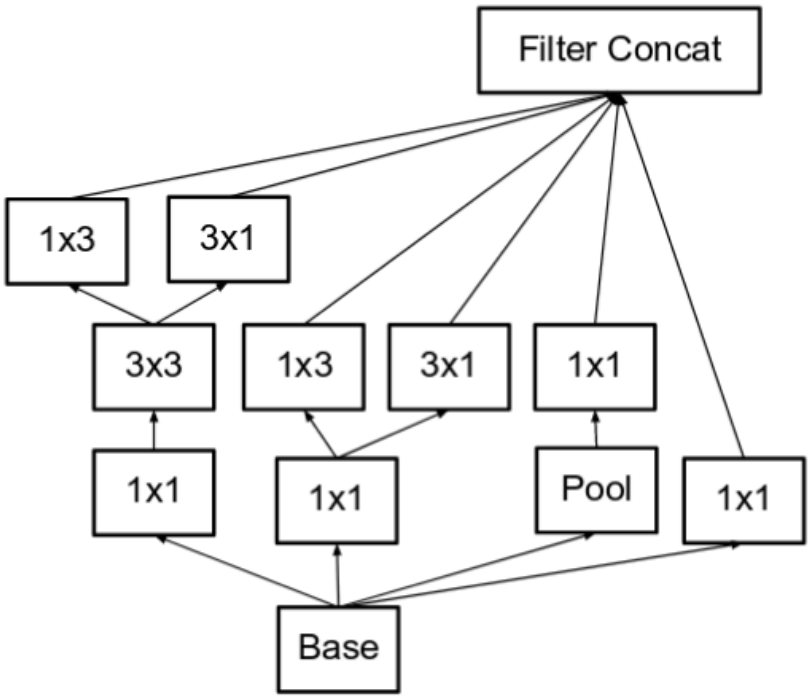


Figure 2.20: Inception V2 replaced the two ending 3x3 filters in Figure 2.18 with a 1x3 and 3x1 next to each other, rather than in sequence. Image from Szegedy et al. 2015.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 2.21: Complete architecture of Inception V2. Figures 2.18, 2.19, and 2.20 from this paper are from their figures 5, 6, and 7 from their paper in the architecture table. Image from Szegedy et al. 2015.

Figure 2.22: The Parrot Mambo Fly drone used in this thesis. Here, it has the added camera attachment.

TCP stream from the Mambo's attached camera. From here, it was possible to interact with a direct feed of images without any additional processing or classifications as seen in Figure 2.23. This feed reader is a wrapper around a method where another program may run in parallel. This method is where the object detection runs. A live feed can be seen as well as an additional window with a image marked up with bounding boxes and classification probabilities.

The pyparrot package was developed to teach STEM concepts by programming an autonomous flying drone (McGovern 2018). Pyparrot currently supports Mambo and Bebop drones, but for this research only Mambo drones are used.

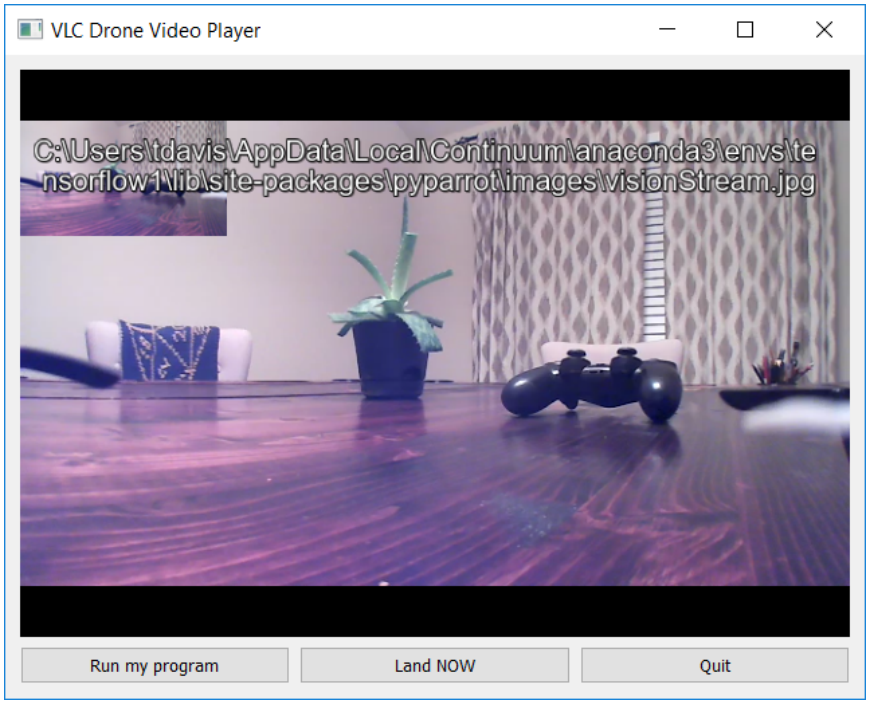


Figure 2.23: Pyparrot python script that streams the raw TCP feed from the Mambo camera.

2.8 Gestures

Gestures are nonverbal forms of communication, that can accentuate what is being said, or communicate complete thoughts without a verbal component. Gesture detection focuses on tracking and recognizing gestures to create semantically meaningful commands (Rautaray and Agrawal 2015). This thesis focuses on classifying visual hand gestures, specifically a hand pointing up, pointing down, a peace sign, and a flat, opened hand. All of these are static gestures (Figure 2.24) - gestures that do not require time or displacement to communicate the intended idea, and are based on a single posture (Kaâniche 2009). Compared to dynamic, moving gestures such as a waving hand (similar to a gesture accompanying the verbal "hi"), static gestures are easier to detect as a single frame is sufficient for classification. Dynamic gestures require additional algorithms and models such as dynamic time warping (Müller 2007) and Hidden Markov Models (Marasovi et al. 2015) for gesture detection. Classifying dynamic gestures require time and locations to communicate a coherent thought (Rautaray and Agrawal 2015).

The gestures are further classified as being 2-Dimensional static models, which are different than motion based models (Rautaray and Agrawal 2015). Motion-based models require a field of activators that detect when some object has passed through it or is currently activating it, and/or local motion histograms (Luo et al. 2010). The gestures for this thesis are free to move during detection, and do not require a specific movement set, or any movement at all, to be understood. When performing the gesture detection, previous predictions will not influence a frame's classification, though there are models today that rely on tracking positions and previous frames (Rautaray and Agrawal 2015).

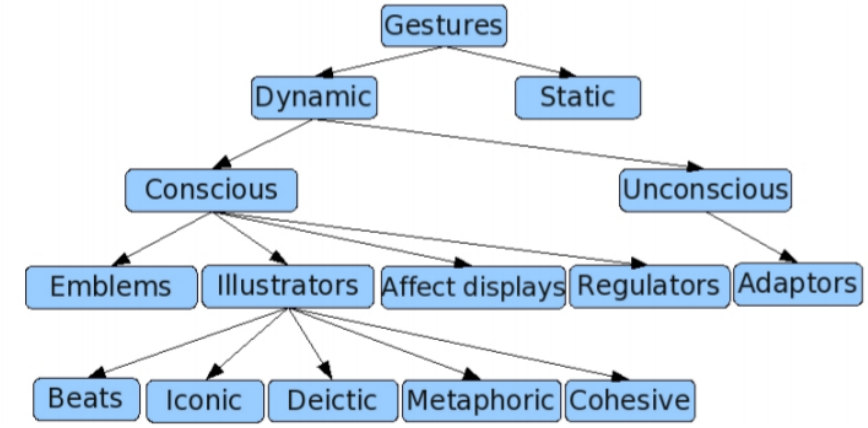


Figure 2.24: Taxonomy of Gestures for Gesture Detection from (Kaâniche 2009). The GIFT dataset consists entirely of static gestures.

This thesis focuses entirely on streaming video to classify gestures instead of tracking devices on hands or special monitors (Kaâniche 2009).

Finally, gestures are considered completely classified when the gesture is first detected and bounded with the appropriate classification as seen in Figure 2.25. A gesture is classified and printed on the image when the probability of the gestures surpasses 85%. This percentage is arbitrary and was chosen empirically. Raising that percentage will cause fewer gestures to be classified on the image, but gestures will be much more reliably identified. Lowering the percentage will allow for more abstract interpretations of gestures to be identified, but will also afford possible noise and erroneous identifications on in-between frames when a gesture may be beginning, but is not fully formed. The location of the gesture does not validate or harm the probability of the gesture, nor does the preceding or proceeding gestures in the stream. When dealing with more than one gesture in the frame (Figure 2.26), the terminating gestures are considered first, namely the "land" command, followed by highest

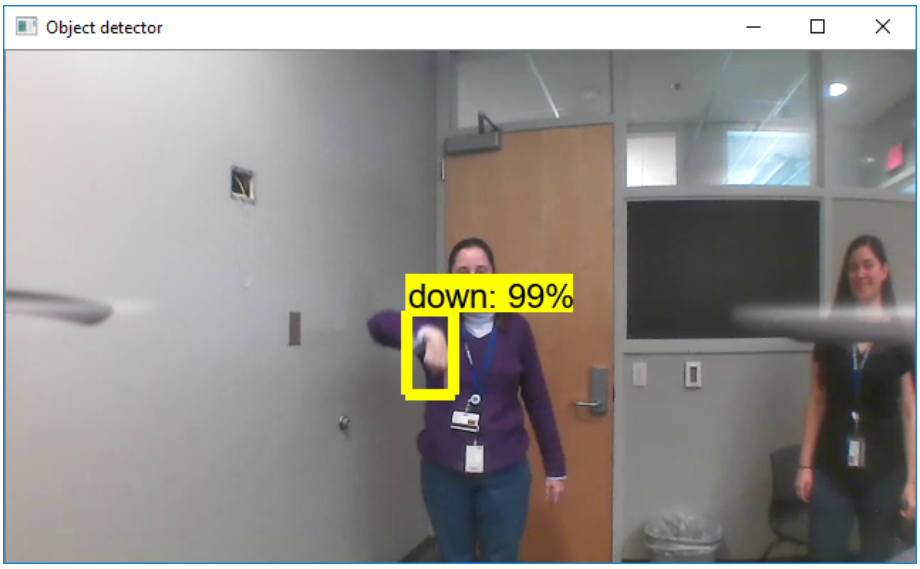


Figure 2.25: A correctly labeled, bounded "down" gesture.

probability gestures. As a last resort if multiple non-terminating gestures are detected with equal percentages, the gesture whose bounding box is closest to the origin will be chosen. This translates to the gesturing stage right and the streaming camera's left. This ensures safe-guessing a killswitch, terminating the program and safely landing the drone rather than performing a non-terminating action.

Figure 2.26: A frame with two classified gestures. The protocol would interpret this frame as a terminating frame since "land" is a termination action and is always be considered first for safety rather than the higher percentage "peace" gesture.

Chapter 3

GIFT Data Set

"Gesture Images For Training," or GIFT, is a contribution of this thesis. GIFT is an image dataset collected by image capture with the mini-drone's attachable camera (Figure 3.1). This camera produced 640 X 360 colored jpg images, and sent about 30 images a second to a laptop through a TCP connection.

The dataset consists of four gestures: a single finger pointing up, a single finger pointing down, two fingers in a "V" shape, and a flat open hand with the palm facing upwards. These are referred to as "up", "down", "peace", and "land" respectively (Figure 3.2). These gestures were collected from 23 volunteers across different months. We obtained permission from each volunteer to use their images in the GIFT dataset. Each volunteer produced the four gestures in differing environments (e.g. once against a wall with little noise, once against a brightly lit projector screen - both seen in Figure 3.3.) Frequently, the volunteers were not the only object visible in the image, and in a few cases, were not the only gestures present (Figure 3.4) - some participants waiting their turn were unintentionally giving some of the gestures in the image. However, most of the images gathered per individual consisted of a single gesture given by the volunteer with plenty of noise in the background such as posters, tables, backpacks, and people (Figure 3.5).

Every frame from the TCP camera during each session was saved as an individual image to a directory for future viewing and labeling. As the images

Figure 3.1: The mini-drone camera attachment, both separated and installed on the mini-drone.

were taken from a TCP stream, a single gesture resulted in more than one image. Let us consider the "up" gesture. The normal order of events was that the volunteer raised their arm and hand into the air, raise a single finger to the sky, and then lowered their hand. In that small span of a few seconds, there were about 90 images of the up gesture in different positions from waist to sky. This included the finger being initially raised from the hand, not quite fully straight up in the air but clear enough to be seen as up (Figure 3.6), and then the finger and hand going back down but not so far as to be considered not-up. This is to say that when viewing the collection of images for each person, some subset of the photos represents the life cycle of the gesture. GIFT contains these stages of each gesture as they contain many orientations of the hand that still display a valid gesture.

The light source was varied when collecting gestures from the volunteers. This allowed the agent to better abstract gestures in different environments such as a classroom versus a shadowed patio. The three main types of environments were "well-lit room from over head lights," "outside with direct sunlight," and "high-contrast light and shadows," with examples shown in Figure 3.7.

Figure 3.2: Gestures from top to bottom: “up”, “down”, “peace”, and “land”.

Figure 3.3: Two “up” gestures: one with no harsh light and a clear background, and one with a brightly lit projector screen.

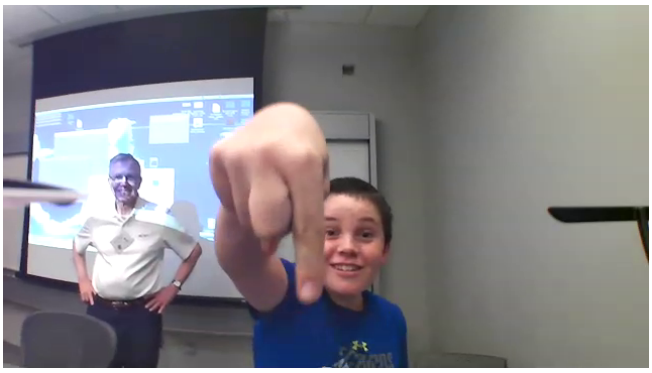


Figure 3.4: Both individuals are giving the “down” sign: one prominently up front, and one in the background near their pocket.

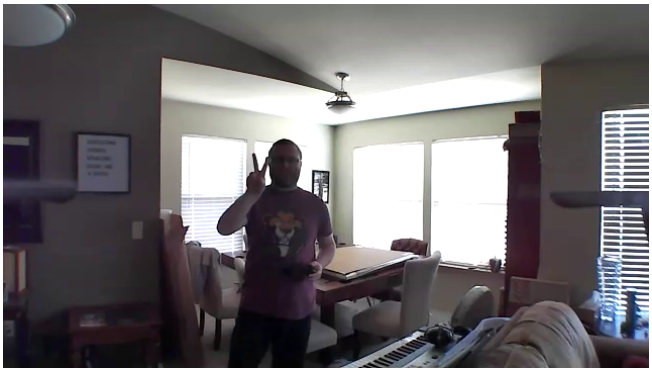


Figure 3.5: A “peace” gesture given with a high contrast background and a noisy environment with furniture and lighting.

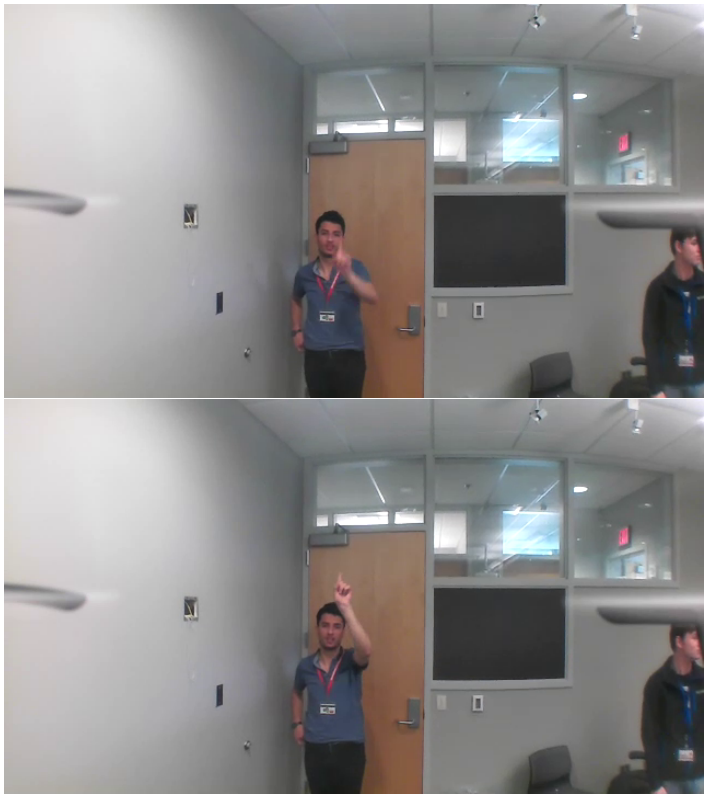
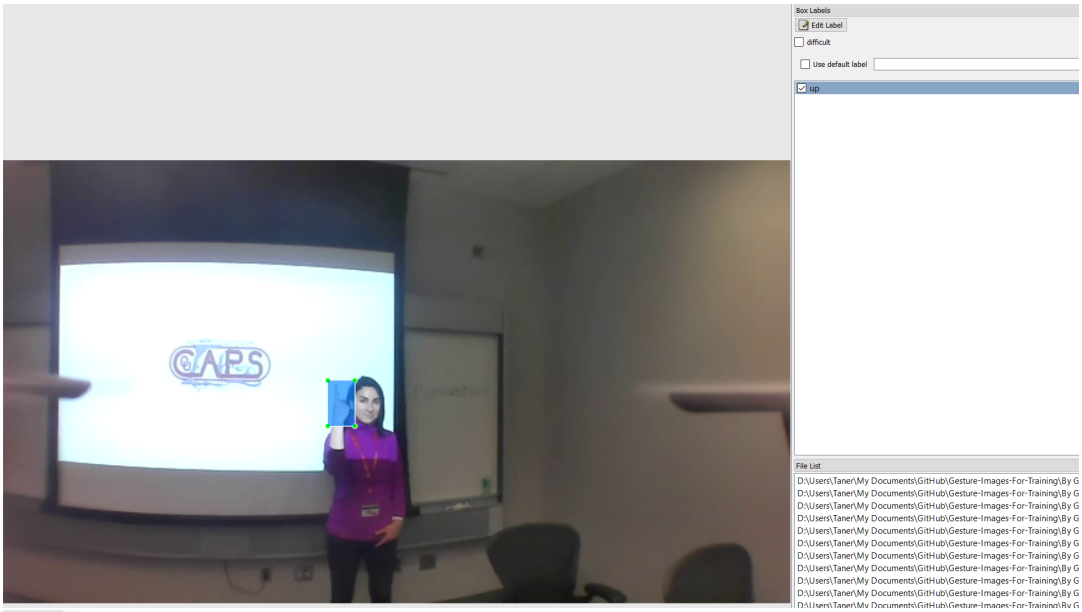


Figure 3.6: An “up” gesture coming from the chest to the sky. Both images were weighted equally as an “up” gesture during training.

Figure 3.7: Up gestures in three common environments: well-lit (top-left), direct sunlight (top-right), and high-contrast (bottom).

The data was collected through many sessions and was labeled for training by using Tzutalin's python application "LabelImg" (Tzutalin 2015). The application loads each image into an editor where one can draw bounding boxes on any and all desired objects and give them appropriate labels. After an image has been deemed by the user as fully labeled, the program creates an associated xml file with metadata such as the classification label and locations of the bounding box edges. LabelImg currently supports creating YOLO text files and PascalVOC xmls. For this research I used PacalVOC xml. Figure 3.8 is an image with the corresponding boundary box and PascalVOC file. The PascalVOC xml is the file that the agent uses to compare the truth bounding box and label versus the predicted bounding box and label.




```xml
<annotation>
    <folder>up</folder>
    <filename>large2_594.jpg</filename>
    <path>D:\Users\Taner\Desktop\Drone_Data_Combined\up\large2_594.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>0</width>
        <height>0</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>up</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>264</xmin>
            <ymin>179</ymin>
            <xmax>286</xmax>
            <ymax>216</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3.8: “Up” gesture in LabelImg application with associated PascalVOC xml file. Program by (Tzutalin 2015).

	Down	Land	Peace	Up	
A	61	102	63	203	429
B	85	99	93	102	379
C	126	230	283	162	801
D	183	229	229	210	851
E	189	231	209	163	792
F	13	36	44	57	150
G	90	148	184	156	578
H	278	336	572	341	1527
I	142	142	195	118	597
J	198	93	178	186	655
K	126	138	145	172	581
L	9	44	32	29	114
M	236	205	173	335	949
N	66	150	94	62	372
O	19	40	115	84	258
P	229	410	210	434	1283
Q	46	96	62	59	263
R	73	79	64	104	320
S	130	406	390	146	1072
T	131	117	142	104	494
U	74	71	84	58	287
V	362	425	406	393	1586
W	271	289	282	296	1138
	3137	4116	4249	3974	

Figure 3.9: Each volunteer (A through W) gave the corresponding gestures in different environments.

	Male	Female
Total	12	11
% of Images	56.49	43.51
% Down	58.46	41.54
% Land	56.66	43.34
% Peace	53.26	46.74
%Up	58.23	41.77
Middle/Dark Skin Color	25.00	27.27
Light Skin Color	75.00	72.73

Figure 3.10: Female and Male breakdown with relation to the GIFT data set. Genders were determined by appearance.

3.1 Data in Numbers

In total, 15,476 images from 23 volunteers were used to train the agents.

Females made up 43.51% of the gesture database with males making up 56.49%. Figure 3.10 is the break up of gender and gesture relations to the database as a whole, and Figure 3.11 is each volunteer's contribution. Given that the division of female/male phenotypes is 44% and 56%, each gesture is represented by the same division +/- 3%. The average contribution of an individual is 4.35% (100% of of the data/23 volunteers), though the contribution of the 23 volunteers is either much larger or smaller than 4.35%. This is because given that most photo-recording sessions were about the same length of time, most of the volunteers were either present for only one session, or present for most sessions - the volunteers present for most sessions were either the author or the participants in a lab section where many of these sessions were held.

The distribution of apparent skin color is shown in Table 3.1 . Each image's volunteer was evaluated by the author at having a light complexion, a dark

	% Total
A	2.772
B	2.449
C	5.176
D	5.499
E	5.118
F	0.9692
G	3.735
H	9.867
I	3.858
J	4.232
K	3.754
L	0.7366
M	6.132
N	2.404
O	1.667
P	8.29
Q	1.699
R	2.068
S	6.927
T	3.192
U	1.854
V	10.25
W	7.353

Figure 3.11: Each Volunteer's individual contribution with their associated observed gender. A is female, and B is male, for example.

	Percent of Total Images
Light	66.3091%
Middle	20.2249%
Dark	13.466%

Table 3.1: Apparent Skin Color Representation in the GIFT dataset.

	Number of Images	Percent of Total Images
High Contrast	2027	13.28%
Outside	1505	9.86%
Well-Lit	11735	76.86%

Table 3.2: Environment Representation in GIFT Dataset.

complexion, or a middle complexion. The percentage of middle and dark skin complexions are just over a third of the database.

The distribution of the three environment types is shown in Table 3.2. The largest portion is the well-lit room with overhead lights, as these describe meeting rooms and classrooms, both of which were where image sessions commonly took place. These classrooms and meeting rooms, though containing windows, were internal rooms and were not connected to an outside-facing wall. The smallest portion is the direct sunlight category as only one session was held outside with weather permitting a clear day and sunshine.

Chapter 4

Experimental Design

As a proof of concept that deep learning can be used in real-time to do gesture recognition on drones, I chose four gestures to identify. Each gesture was treated as a single request, meaning the drone would not need to consider multiple differing gestures before and after the current parsed image. I designated an "up", "down", "peace", and "land" gesture as a simple language from the drones. Each gesture in itself can be understood with a single image and can be given with either hand.

To record data, the drone was flown in front of each volunteer as they gestured, and recorded the video stream at 30 frames per second from the drone's on-board camera. This stream was saved with each frame as an image onto a computer. The agents were initially trained on images taken in a well-lit room with no windows, with the volunteer about 4 meters away from the hovering drone. I did not control the clothes worn, or which hand was used to gesture. More images were taken later in difficult lighting such as in front of a projector screen with the projector on, and in front of a bright window with room well lit. The last environment in which images were taken was outside in direct sunlight and direct shade.

After the raw data was collected, I went through all of the saved images frame by frame and selected every image where a gesture was present. I then used LabelImg (Tzutalin 2015) to create metadata about each image. I detailed

	Well-Lit	**Well-Lit and Complex**
SSD MobileNet V2	MobileNetV2_Clear	MobileNetV2_All
Faster-RCNN InceptionNet V2	InceptionV2_Clear	InceptionV2_All

Table 4.1: The four agents trained on the GIFT data set. The cross-sections define the names by which the agents are known. "Clear" agents are trained on gestures in a well-lit room. "All" agents are trained on all data, including outside images and high-contrast images.

the width and height of the images, x and y min/max of the bounding box, and the gesture classification of each bounding box in the image. Though most of the images all contain single gestures, there are a few images that contain multiple gestures.

Next, I trained two COCO-trained models on two different subsets of the previous data set, making four total agents. COCO (Common Objects in Context) is a dataset focusing on scene awareness in addition to 91 object types (Lin et al. 2014). Each of the models were selected from Tensorflow's model zoo (Tensorflow 2018a). In the zoo, each model has a speed-of-detection in milliseconds, and a COCO mean average precision (Microsoft 2018). I used "ssd_mobilenet_v2_coco" with a 1x depth multiplier, and "faster_rcnn_inception_v2_coco".

For training, each agent was trained on a GPU with four gigabytes of dedicated memory. Each model was trained in Python 3.6 with Tensorflow 1.9.0 (Tensorflow 2018b). For each model I trained the "Clean" model on data where the gesture was clearly visible in a well-lit room, and then trained the "All" model with every picture from the GIFT dataset. The reason is that the agent should be able to identify an unobstructed gesture with ease, but a gesture that

is present in multiple types of lighting and a noisy background should also be identified as well as the unobstructed gesture. All agents had a learning rate of 0.0002 and were trained for 30,000 steps.

After training all four agents, I then tested the classifiers on new images where each gesture was given at three, five, and seven meters from the drone. Each gesture was given with both hands, and in three environments: well-lit room, high contrast, and outside. These 72 images made up the test set.

Finally, after the agents were trained, I created a python script to fly the drone using gestures. Once the drone takes off, it stays stationary until it reads one of the four gestures:

- "Up" moves the drone towards the user very slowly
- "Down" move the drone away from the user slowly
- "Peace" moves the drone so that the gesture is in the middle of the frame
- "Land" cause the drone to land

Chapter 5

Results

The first focus is on the Tensorflow loss scores of the four trained agents: MobileNetV2_Clear, MobileNetV2_All, InceptionV2_Clear, and InceptionV2_All. Then, as well as identifying the 72 gestures from both hands in all environments. All agents have a classification loss (Figure 5.1), a localization loss (Figure 5.2), and a total loss (Figure 5.3) - all of which are lower for better agents. A lower classification loss reflects the agent's ability to accurately classify the bound object as the correct gesture. With this, a lower localization loss means the agent proposed a bounding box close to the true gesture's bounding box. The total loss is the sum of all losses for an agent, which includes two additional losses for InceptionV2 agents. These added losses are from the Region Proposal Networks (RPN), which SSDs such as MobileNets do not have. Both Inception agents have a RPN localization loss (Figure 5.4) and an objectness loss (Figure 5.5). The localization loss is the CNN's regressor's loss of the bounding boxes it proposes for objects in an image compared to the RoIs. The objectness loss, when lower, means that the agent is able to pick proposals as objects instead of the background in the region of interest. The accuracy of these models is tested with the 72 gestures.

Overall, the four agents were able to correctly learn and identify the images in their bounded boxes with diminishing errors (Figure 5.1). Empirically,

MobileNetsV2 is considered trained when the network has a consistent classification loss of around 2, and InceptionV2 is trained when the network has a consistent classification loss of 0.05. In Figure 5.1 the Clear agents classified just as well as the All agents. However, testing against the 72 test images at three, five, and seven meters (Table 5.1, 5.2, and 5.3) the ALL agents identify more gestures correctly across all distances and environments. Further, InceptionV2 agents out perform their MobileNetV2 counter parts across the board with identifying more testing gestures correctly. MobileNetsV2 do trade lower accuracy for higher speed which is preferable for Mobile devices, but with something as volatile and possibly dangerous as a drone, one should prefer the agents which correctly identify more gestures correctly over speed.

As an additional test to prove InceptionV2 outperforms MobileNetsV2 on identifying gestures in the GIFT dataset, I performed a paired t-test. New InceptionV2 and MobileNet agents, two in total, were trained on a 10-fold cross validation of the GIFT dataset. Each split was chosen across the volunteers instead of on environments. The splits are outlined in Figure 5.6 and the average contingency tables for both models are found in Figure 5.7. With each split, each volunteer is exclusively in the test, validation, or training set. Each volunteer appears only once in the test sets and once in validation sets in the ten folds. Our hypothesis is that the InceptionV2 model and the MobileNetsV2 model are the same. On average, the InceptionV2 model had a 34.2% higher accuracy than the MobileNetV2 model (paired t-test, $p < 0.01$ and $df = 18$). The intersection of bounding boxes of the predicted gesture and the truth gesture were measured as well in Figure 5.8. Over the 10-fold cross validation test folds, a bounding boxes were considered overlapping if the boxes shared at least one pixel in common, and do not take the predicted gesture into account. Measuring

	Inside_right	Inside_left	High_Contrast_right	High_Contrast_left	Outside_right	Outside_left
MobileNetV2_Clear	U D L	U D L	U D L P	U D L P	U D L	U D L
MobileNetV2_All	-	D	P L	P D L	U D	U D
InceptionV2_Clear	-	-	-	U	U D	D
InceptionV2_All	-	-	-	-	U	U L

Table 5.1: Gestures not identified at three meters from drone. U = Up, D = Down, L = Land, P = Peace.

the intersection of bounding boxes in Figure 5.8 lends evidence to InceptionV2 identifying gesturing hands better than MobileNetV2.

The InceptionV2 agents had high RPN localization losses, but low classification losses. This is likely due in part to human error when labeling the GIFT data. Some gestures truths may include small parts of the wrist or have a bit too much space around the hand as dead air, versus some truths that are exactly tight around the hand with no wrist in the bounding box. This would produce low classification losses as it would still guess the correct gesture but may box too much of the wrist or too little of the wrist, making the localization loss bounce back and forth fight for wrists or no wrists. Bad RPN guesses due to this wrist issue would lead to higher box localization losses with the InceptionV2 nets. Comparing the classification loss of all agents in Figure 5.1 with the total losses in Figure 5.3, MobileNetV2's classification loss is over two-thirds of the total loss, while InceptionV2's classification loss is about half of the total loss.

However, whether the box is too large or too small, one must consider the classification loss the important identifier as it is more important to be correct on what is gestured vs exactly how boxed-in the gesture is. It is for this reason and the low p-value from the paired t-test that the InceptionV2 agent model should be chosen over a MobileNetsV2 model to control an intelligent mini drone.

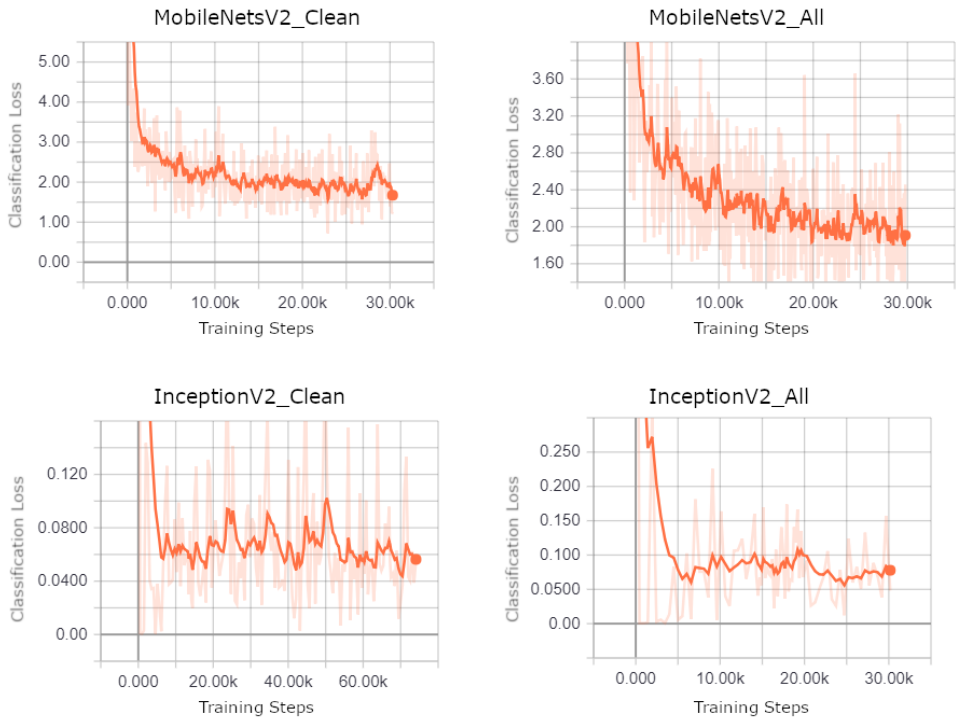


Figure 5.1: Box Classification Losses of the four agents on the GIFT training data. MobileNets should be trained to report a consistent classification loss of 2, and Inception should be trained to report a consistent classification loss of 0.05.

	Inside_right	Inside_left	High_Contrast_right	High_Contrast_left	Outside_right	Outside_left
MobileNetV2_Clear	D	D	D L	D L	U D	U D
MobileNetV2_All	-	-	L	D	D	D
InceptionV2_Clear	-	-	-	U	U D	D
InceptionV2_All	-	-	-	-	L	D

Table 5.2: Gestures not identified at five meters from drone. U = Up, D = Down, L = Land, P = Peace.

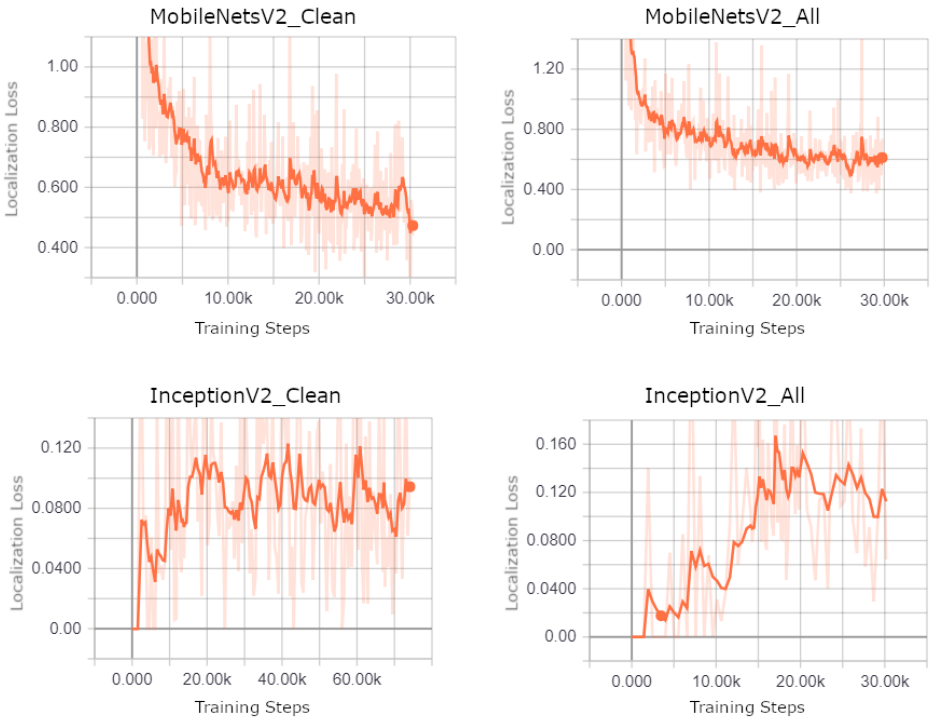


Figure 5.2: Box Localization Losses of the four agents on the GIFT training data. A low localization loss means the agent identified a gesture whose bounding box was close to the truth's bounding box.

	Inside_right	Inside_left	High_Contrast_right	High_Contrast_left	Outside_right	Outside_left
MobileNetV2_Clear	D L	D L	D L	U L P	U L	U L
MobileNetV2_All	-	-	P L	D L P	U D	U D
InceptionV2_Clear	-	-	-	U	-	U D
InceptionV2_All	-	-	-	-	-	U

Table 5.3: Gestures not identified at seven meters from drone. U = Up, D = Down, L = Land, P = Peace.

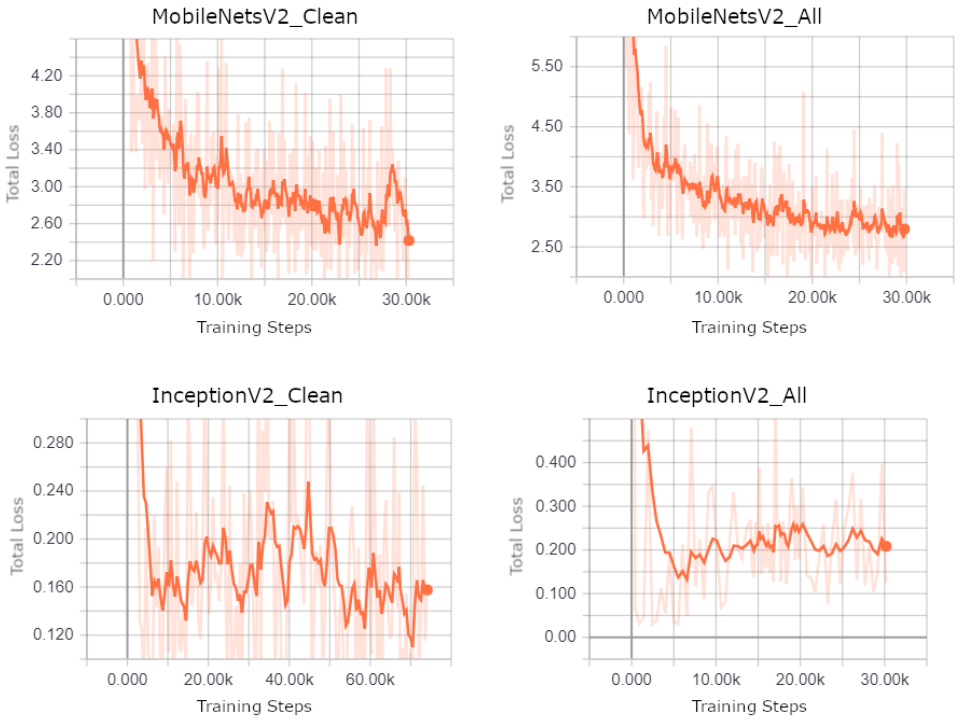


Figure 5.3: Total Losses of the agents trained on the GIFT dataset.

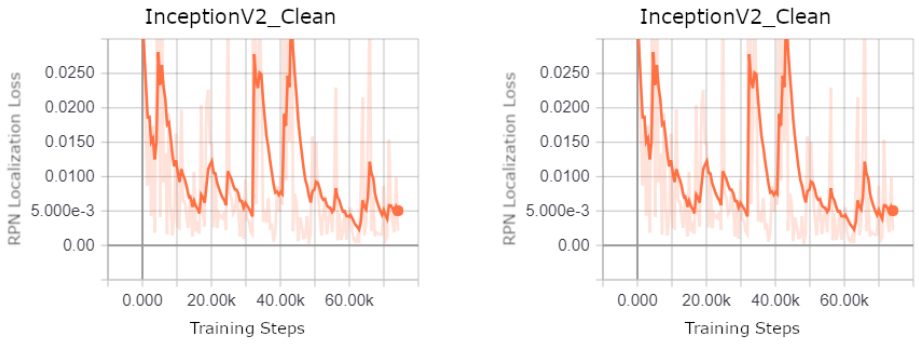


Figure 5.4: Region Proposal Network Localization Losses of both InceptionNet agents.

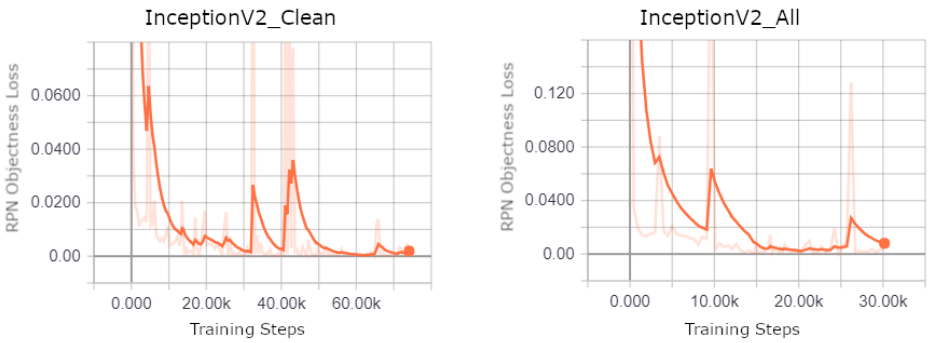


Figure 5.5: Region Proposal Network Objectness Losses of both InceptionNet agents.

Test and Validation Data Splits

Test	Test %	Validation	Validation %
V	10.3	H	9.9
ST	10.1	AW	10.2
H	9.9	MQR	9.9
ABE	10.3	CFG	9.9
DFI	10.4	BKNO	10.3
JKR	10.1	PU	10.2
LUW	10	V	10.3
PQ	10	EJL	10
CNO	9.3	ST	10.1
GM	9.8	DI	9.4

Figure 5.6: The 23 volunteers (labeled A to W) were split into 10 different sets for the two-sample t-test. Every volunteer appeared in every split, and only appeared once in the test, validation, or training set. Each split was made to be close to 80% training, 10% validation, and 10% test. The volunteers' letters not appearing in a row's test or validation column are in the training set for that split.

Inception V2 Average Across 10 Folds

		Truth					
		NG	Up	Down	Peace	Land	Total
Predicted	NG	0	0	0	0	0	0
	Up	33.5	270.2	6.6	73.5	8.5	392.3
	Down	71.9	6.5	221	7	7.1	313.5
	Peace	39.4	67.4	7.9	296.7	6.6	418
	Land	100.2	6.9	31.6	13.6	271.5	423.8
	Total	245	351	267.1	390.8	293.7	1547.6

MobileNet V2 Average Across 10 Folds

		Truth					
		NG	Up	Down	Peace	Land	Total
Predicted	NG	0	0	0	0	0	0
	Up	172.1	149.6	4.7	54.8	11.1	392.3
	Down	193.4	16.7	54.1	10.8	38.5	313.5
	Peace	151.6	97.8	6.1	155.9	6.6	418
	Land	217.7	6.5	5.8	25.1	168.7	423.8
	Total	734.8	270.6	70.7	246.6	224.9	1547.6

Figure 5.7: The average contingency tables for the ten InceptionNet and MobileNet agents on the splits from Figure 5.6.

Average Prediction/Truth Bounding Box Overlap

	InceptionV2	MobilenetV2
Overlapped Bounding Box	1202.6	696.7
Not Overlapped Bounding Box	100	116.1
No Gesture Detected	245	734.8

Figure 5.8: This figure shows the average amount of gesture bounding boxes that shared at least one pixel with the truth bounding box of an image during the 10-fold cross validation tests. The gesture guessed is not considered in this table. Only the intersection of the predicted gesture's bounding box and the truth bounding box is considered. If no gesture was guessed for an image, then no bounding box was proposed.

Chapter 6

Conclusions and Future Work

My contributions are as follows:

1. Creating and curating the GIFT data set
2. Developing a program that lets a mini-drone react to hand gestures
3. Evaluating the ability of Faster RCNN Inception V2 and MobileNets to identify gestures from a video feed
4. Training multiple agents with different subsets of GIFT data to address lighting and environment issues

My research demonstrates the success and process of training real-time agents on minidrones to detect hand gestures and react appropriately. The agents were successfully able to capture an image in real time of the environment, detect all gestures in the image, and then react to the gesture as programmed. This will open up the possibility of future programmers to have agents already trained on gesture data to develop agents that respond to their custom gestures.

Currently the data used for this research can be found at https://github.com/TanerDavis/Gesture-Images-For-Training with an MIT License included. I am releasing the GIFT dataset of hand gestures for future gesture-related research, and to function as a benchmark for intelligent gesture recognition. Currently GIFT only has four gestures: the "up," "down," "peace,"

and "land." The collection of images is currently sorted by gesture and by person, presenting the same data set twice with different categorizations. With every image, there is an accompanying Pascal VOC XML metadata file (Aytar 2012). Each file gives the class id for each gesture in the photo as well as the bounding coordinates of the gesture in the image.

To extend the use of language in gesture recognition, I'd like to create an agent that learns a command, not based solely on the presence of some gesture, but rather the distance between two hand gestures in two points of time like a hello-wave motion. The wave could be abstracted as two non-bent wrists with an open palm displaced by a few inches and be some number of frames apart. This could then be interpreted as a welcoming gesture, and the drone shakes to wave back. This consideration of where the gesture is in the image and when it was given compared to other gestures would afford the ability to build sentences with gestures to create a non-verbal lexicon with drones.

Reference List

Aytar, Y., 2012: The PASCAL Visual Object Classes Homepage. Visited 2018-11-13.
URL http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Chilson, C., 2017: Automated Detection of Bird Roosts Using NEXRAD Radar Data and Convolutional Neural Networks.

Chollet, F., 2018: *Deep learning with Python*. Manning.

Ciresan, D. C., U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, 2011: Flexible, high performance convolutional neural networks for image classification. *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Barcelona, Spain, volume 22, 1237.

Cuban, M., J. Reitman, and J. Pritchard, 2018: Object detection and analysis via unmanned aerial vehicle. Visited 2018-12-11.
URL https://patents.google.com/patent/US9989965B2/en

Davis, T., 2018: Gesture Images For Training. Visited 2018-11-13.
URL https://github.com/TanerDavis/Gesture-Images-For-Training

EdjeElectronics, 2018: How To Train an Object Detection Classifier for Multiple Objects Using TensorFlow (GPU) on Windows 10. Visited 2018-11-13.
URL https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

Felzenszwalb, P. F. and D. P. Huttenlocher, 2004: Efficient graph-based image segmentation. *International Journal of Computer Vision*, **59**, 167–181.

Girshick, R., 2015: Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 1440–1448.

Girshick, R., J. Donahue, T. Darrell, and J. Malik, 2013: Rich feature hierarchies for accurate object detection and semantic segmentation. 2013. *arXiv preprint arXiv:1311.2524*.

Heliguy, 2018: DJI Spark Gesture Control Tutorial. Visited 2018-11-13.
URL https://www.heliguy.com/blog/dji-spark-gesture-control-tutorial/

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, 2017: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Visited 12-14-18.
URL https://arxiv.org/pdf/1704.04861.pdf

Jantzen, J., 1998: Introduction to perceptron networks. *Technical University of Denmark, Lyngby, Denmark, Technical Report*.

Kaâniche, M., 2009: *Gesture recognition from video sequences*. Ph.D. thesis, Université Nice Sophia Antipolis.

Karpathy, A., 2016: Convolutional Neural Networks for Visual Recognition. Visited 2018-11-13.
URL `http://cs231n.github.io/convolutional-networks/`

Krizhevsky, A. and G. Hinton, 2010: Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, **40**.

Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, 2014: Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision*, Springer, 740–755.

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, 2016: SSD: Single shot multibox detector. *European Conference on Computer Vision*, Springer, 21–37.

Luo, Q., X. Kong, G. Zeng, and J. Fan, 2010: Human action detection via boosted local motion histograms. *Machine Vision and Applications*, **21**, 377–389.

Marasovi, T., V. Papi, and J. Marasovi, 2015: Motion-Based Gesture Recognition Algorithms for Robot Manipulation. *International Journal of Advanced Robotic Systems*, **12**, 51.
URL `https://doi.org/10.5772/60077`

McGovern, A., 2018: Pyparrot. Visited 2018-11-13.
URL `https://github.com/amymcgovern/pyparrot`

Microsoft, 2018: COCO Evaluation Metrics. Visited 2018-11-13.
URL `http://cocodataset.org/#detection-eval`

Müller, M., 2007: *Information Retrieval for Music and Motion*. Springer-Verlag, Berlin, Heidelberg.

Rautaray, S. S. and A. Agrawal, 2015: Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review*, **43**, 1–54.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi, 2016: You Only Look Once: Unified, Real-Time Object Detection. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
URL `http://dx.doi.org/10.1109/CVPR.2016.91`

Ren, S., K. He, R. Girshick, and J. Sun, 2015: Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 91–99.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1985: Learning internal representations by error propagation. Technical report, California Univ San Diego, La Jolla Inst for Cognitive Science.

Russell, S. J. and P. Norvig, 2016: *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 2018: MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2015: Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tensorflow, 2018a: Tensorflow Detection Model Zoo. Visited 2018-11-13.
URL `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`

—, 2018b: Tensorflow Object Detection. Visited 2018-11-13.
URL `https://github.com/tensorflow/models/tree/master/research/object_detection`

Tzutalin, 2015: Visited 2018-11-13.
URL `https://github.com/tzutalin/labelImg`

Uijlings, J. R., K. E. Van De Sande, T. Gevers, and A. W. Smeulders, 2013: Selective search for object recognition. *International Journal of Computer Vision*, **104**, 154–171.

Viola, P. and M. J. Jones, 2004: Robust real-time face detection. *International Journal of Computer Vision*, **57**, 137–154.