BAYESIAN CONVOLUTIONAL NEURAL NETWORK WITH PREDICTION SMOOTHING AND ADVERSARIAL CLASS THRESHOLDS

THESIS

Noah Miller, Second Lieutenant, USAF

AFIT-ENS-MS-22-M-154

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-22-M-154

BAYESIAN CONVOLUTIONAL NEURAL NETWORK WITH PREDICTION SMOOTHING AND ADVERSARIAL CLASS THRESHOLDS

THESIS

Presented to the Faculty

Department of Operations Research

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Noah Miller, B.S.

Second Lieutenant, USAF

March 24, 2022

AFIT-ENS-MS-22-M-154

BAYESIAN CONVOLUTIONAL NEURAL NETWORK WITH PREDICTION SMOOTHING AND ADVERSARIAL CLASS THRESHOLDS

THESIS

Noah Miller, B.S.
Second Lieutenant, USAF

Committee Membership:

Bruce Cox, Ph.D
Chair

Lance Champagne, Ph.D
Member

Trevor Bihl, Ph. D
Member

AFIT-ENS-MS-22-M-154

Abstract

Using convolutional neural networks (CNNs) for image classification for each frame in a video is a very common technique. Unfortunately, CNNs are very brittle and have a tendency to be over confident in their predictions. This can lead to what we will refer to as "flickering," which is when the predictions between frames jump back and forth between classes. In this paper, new methods are proposed to combat these shortcomings. This paper utilizes a Bayesian CNN which allows for a distribution of outputs on each data point instead of just a point estimate. These distributions are then smoothed over multiple frames to generate a final distribution and classification which reduces flickering. Our technique is able to reduce flickering by 67%. We also propose a second method to combat False Positive predictions of certain adversarial classes, or classes that have some cost if predicted incorrectly. This is accomplished by increasing the confidence threshold the adversarial class must meet in order to be the final predicted class. This technique is able to reduce false positives by 5.43%, while maintaining accuracy.

To my incredible wife. I could not have done this without you.

Acknowledgements

I would like to thank my advisor, Dr. Cox, who supported me even in the midst of numerous setbacks. I would also like to thank Dr. Champagne as well as Dr. Bihl from AFRL for their support of thesis. Without the support of these individuals, as well as others, this thesis would not have been possible.

Noah Miller

Table of Contents

Page

Abstract iv

Dedication v

Acknowledgements vi

List of Figures ix

List of Tables x

I. Introduction 1

1.1 Problem Statement 1
1.2 Background and Motivation 1
1.3 Organization of Thesis 2

II. Background and Literature Review 3

2.1 Artificial Neural Networks 3
2.2 Convolutional Neural Networks 4
2.3 Object Detection using Convolutional Neural Networks 8
2.4 Bayesian Neural Networks 13
2.5 Previous Work on Model Uncertainty 16

III. Methodology 19

3.1 Original Investigative Question 19
3.1.1 Description of Data 19
3.1.2 Model Overview 20
3.1.3 Front End Model Architecture 20
3.1.4 Back End Model Architecture 24
3.1.5 Hyperparameter Tuning 26
3.1.6 Model Evaluation 28
3.2 Revised Investigative Question 29
3.2.1 Description of Data 29
3.3 Revised Investigative Question 34
3.3.1 Front End Model Architecture 34
3.3.2 Back End Model Architecture 35
3.3.3 Hyperparameter tuning 35
3.3.4 Model Evaluation 36

Page

IV. Results and Analysis 37

4.1 Original Investigation Question Results 37
4.2 Revised Investigation Question Results 39
4.2.1 Model Training 40
4.2.2 Prediction Smoothing 40
4.2.3 Adversarial Threshold 42
4.2.4 Effects of the Number of Predictions per Frame 44

V. Conclusions 46

5.1 Summary 46
5.2 Future Work 46
Appendix: Glossary 48
Bibliography 49

List of Figures

Figure Page

1. An Example of a Perceptron 4
2. Neocognitron Architecture [1] 5
3. LeNet-5 Architecture [2] 6
4. VGG16 Architecture [3] 7
5. YOLOv1 Model [4] 9
6. YOLOv3 Model [5] 11
7. YOLOv3 Detector [5] 12
8. Point Estimates Compared to Distributions on Weights [6] 15
9. YOLOv3-tiny Model [7] 22
10. Bounding Box for YOLOv3 [8] 23
11. Distance Between Frame n and Frame $n-1$ 25
12. An Example of a Precision Recall Curve for mAP 29
13. Durations for UCF101 Classes 31
14. Number of Videos with Certain Durations for UCF101 Classes 32
15. Examples of UCF101 Classes 33
16. Training and Validation Loss During Final 100 Epochs 38
17. Predictions per Class for Standard Model 41
18. Predictions per Class for Bayesian Model with Smoothing 42
19. The Effect of Number of Predictions per Frame on Accuracy and Speed 45

List of Tables

Table Page

1. Affect of α Value on Frame Weight 27
2. Affect of Hyperparameters on mAP 37
3. Optimizer Testing 40
4. Alpha Values 40
5. Adversarial Threshold of 0.0 43
6. Adversarial Threshold of 0.6 43
7. Adversarial Threshold of 0.7 43
8. Adversarial Threshold of 0.8 44
9. Adversarial Threshold of 0.9 44

BAYESIAN CONVOLUTIONAL NEURAL NETWORK WITH PREDICTION SMOOTHING AND ADVERSARIAL CLASS THRESHOLDS

I. Introduction

Object identification is a key element in successful Intelligence, Surveillance and Reconnaissance (ISR) missions. This allows for identification of important targets in an automated setting. This has the potential to increase accuracy of target identification and to reduce man hours. Convolutional Neural Networks are a key component of object identification in combination with other techniques to find the location of said object in the image. Bayesian Neural Networks (BNNs) help quantify the uncertainty that can often be found when utilizing neural networks.

1.1 Problem Statement

The goal of this research is to successfully blend Bayesian Neural Networks with a one step object detection algorithm to more successfully classify images. An attempt will be made to add a new technique that will track a given object from frame to frame in an attempt to utilize information from prior frames to help with the prediction of the current frame.

1.2 Background and Motivation

A prominent shortcoming of object detection and image classification models is what is often referred to as "flickering" in the literature [9] [10] [11]. This phenomena occurs when the object detection or image classification model correctly predicts on

an object a number of times, switches to a false prediction for a frame or two, and then switches back to the correct classification. This can happen for a number of reasons including occlusion, poor lighting, a weird camera angle or simply a failure of the neural network. In some scenarios, this "flickering" may not be problematic, but in any scenario in which automation occurs and would take some action, which it can be assumed would have some cost, this amounts to wasted resources due to the shortcomings of the model. This paper suggests a solution to this flickering problem by creating not only point estimates for each frame, but distributions to help model uncertainty and to not only use these distributions on the current frame, but to carry them into future frames to help increase the accuracy by providing increased information.

1.3 Organization of Thesis

Chapter 2 will discuss the rise of neural networks, the advent of convolutional neural networks, object detection utilizing neural networks, and Bayesian neural networks. Chapter 3 will provide an overview of the methodology utilized to create the proposed model. Chapter 4 will give insight to the results and provide analysis on said results. Chapter 5 will provide a conclusion to the work, as well as suggesting future work that could be done to expand upon the work done here.

II. Background and Literature Review

In this chapter, previous works addressing topics and techniques utilized in this thesis are discussed. This includes the advent of the perceptron and artifical neural networks, convolutional neural networks, object detection and Bayesian neural networks.

2.1 Artificial Neural Networks

The first neural networks were designed by McCulloch and Pitts in 1943 [12]. This paper introduced the first idea of neurons and addressed their connection to the neurons in the brain. Much like neurons in the brain, these neurons would"fire" in response to some "stimulus" where in this case the stimulus is whatever the input is. These neurons however possessed no trainable weights and could only take binary inputs leading to binary outputs and in this way, their use was drastically limited. This simple neural network failed to even represent simple, although non linear, problems such as XOR and NOR. The next major improvement to the neural network was the work of Rosenblatt in 1958 when he published his paper on the perceptron [13]. An example of the perceptron can be seen in Figure 1. Rosenblatt's improvements to the work of McCullough and Pitts included two key factors. First was the ability to handle non-binary inputs. This allowed whole new classes of data to be fed into neural networks greatly improving their use. The second improvement was the addition of weights to the inputs. This allowed greater control of the output of the neural network, also adding to their ability to accurately predict.

However single layer networks still failed to predict on nonlinear problems such as the XOR problem. This problem was solved by the idea of hidden layers in conjunction with backpropagation. While there is some debate regarding who invented the

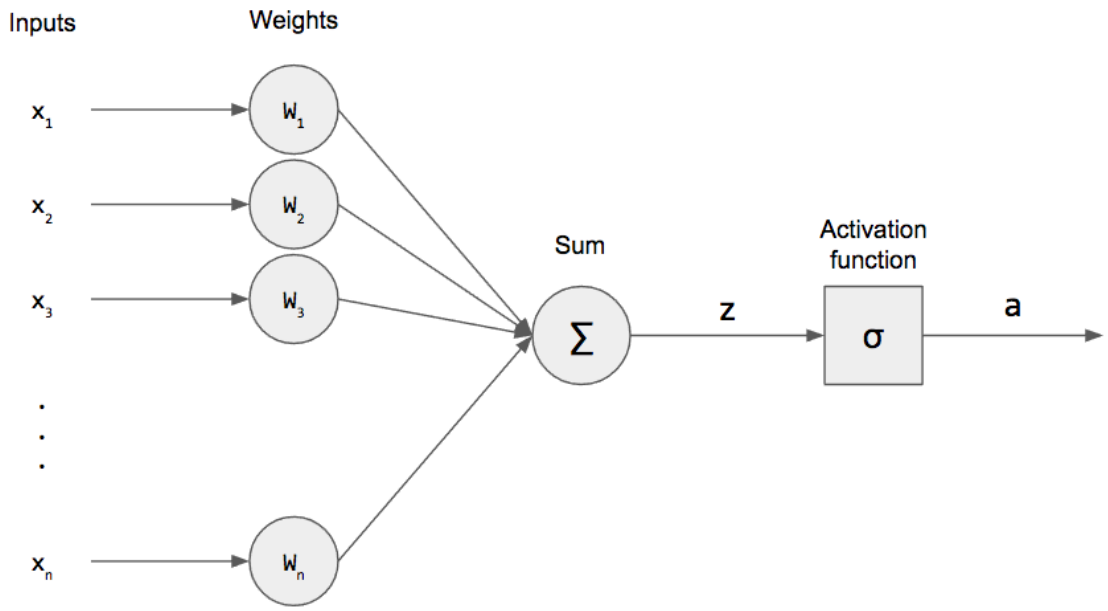

Figure 1: An Example of a Perceptron

idea, it was the 1986 paper written by Rumelhart, Hinton and Williams which popularized it [14]. By adding hidden layers, neural networks were now able to accurately predict on non linear problems as the neurons between layers could interact to model more complex interactions between inputs. It was in this paper as well that back-propagation was introduced which allowed the weights on neurons to be trained by the model instead of having to be chosen manually which allowed for more accurate predictions. These multilayered perceptrons were to first true examples of artificial neural networks (ANNs).

2.2 Convolutional Neural Networks

One of the most obvious uses for neural networks is image classification. However, this is challenging using normal ANN structures. It is not only the values of each individual pixel that matters in image classification but also its location relative to the other pixels in the image. This is where the convolutional neural network is

most effective. The first concept of the convolutional neural network was published in 1980 by Fukushima who invented the neocognitron [1]. In this paper, Fukushima introduces the idea of S cells and C cells. His idea is that a complex image could be predicted accurately by first using simple cells, his S cells, to detect simple patterns and then using complex cells, his C cells, after that to find the more complex patterns which would be combined to accurately predict on an image. A visual representation of the neocognitron can be seen in Figure 2.

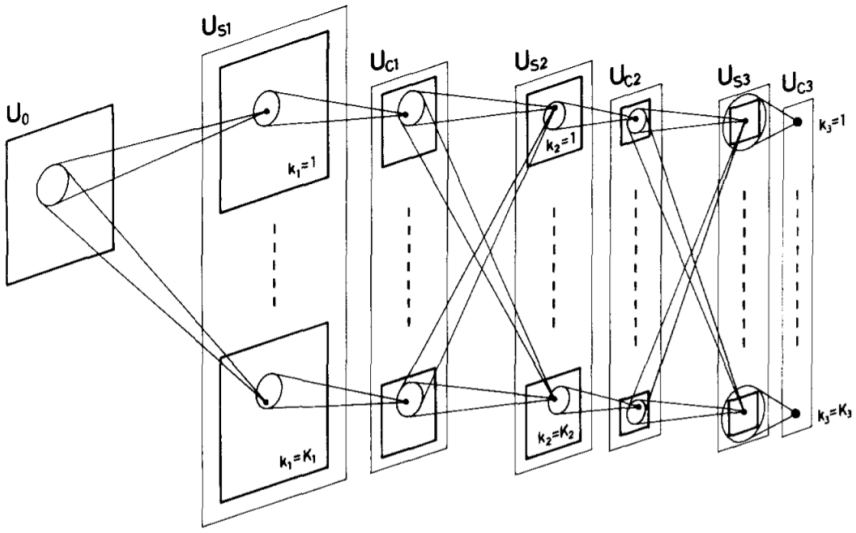

Figure 2: Neocognitron Architecture [1]

The invention of the neocognitron lead to the development of a number of new CNNs designed by LeCun which eventually culminated in LeNet-5, the architecture of which can be seen in Figure 3 [2]. This paper was the first paper to actually use the phrase "Convolutional Neural Network" and to use some of the other terminology utilized today, such as convolutional layers and subsampling, although these had both been roughly invented in the Fukushima paper. The big emphasis in the LeNet-5 paper was the ability for the model to train all of its filters using backpropagation. This meant that the model itself was able to determine what the filters should look like, leading to a more accurate, more efficient neural network. Since LeNet-5 there

has been an explosion of new CNN architectures which have lead to increasing performance on different data sets. One example of the success of CNNs vs other deep neural networks is in the ImageNet large scale visual recognition challenge (ILSVRC). The first two years of the challenge did not have CNNs and the best error rate from those two years was 25.8%. However in just the first year with CNNs, the error rate dropped to only 16.4%. Since then, CNNs of increasing depth have continued to see improvements with the 2017 (and final) winner recording an error rate of only 2.25% [15].

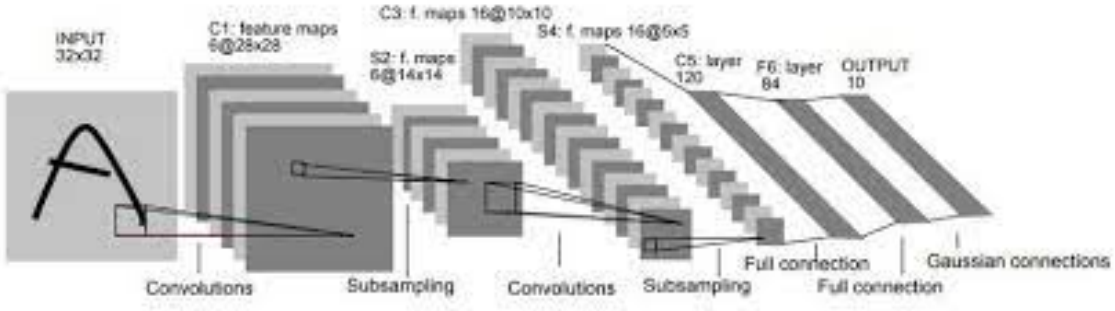

Figure 3: LeNet-5 Architecture [2]

The model that will be utilized in the secondary investigation of this paper is a version of the VGG16 model, created in 2014. This model also competed in the 2014 ILSVRC competition and had an error rate of 7.32% [16]. Although it received a slightly worse accuracy when compared to GoogleNet, what is impressive about this model is its simplicity. VGG16 simply has 16 convolution layers with maxpooling layers every 2-3 layers. This is compared to GoogleNet which created an entire new kind of layer called inception layers. The powerful simplicity of VGG16 makes it a very usable model and it is very easy to do transfer learning with this model. The architecture of VGG16 can be seen in figure 4 [17]. The input for this model is a 224 pixel by 224 pixel image in color, which is a 224 x 224 x 3 array, the 3 being for the RGB color coding. The first layer of this model takes in the 224 x 224 pixel

image and passes it through 64 different filters twice. The image is then downsized in a maxpooling layer to be 112 x 112 pixels and passed through two more layers each with 128 filters. This is then downsized again in another maxpooling layer so that the image is 56 x 56 and passed through three layers with 256 filters each. The image is then downsized again in a maxpooling layer to 28 x 28 pixels and passed through three layers with 512 filters each and then the same again after downsizing to 14 x 14 pixels. After this, the image is downsized one last time to 7 x 7 pixels and passed through one layer with 512 filters. The output from this layer is then flattened and then passed into a dense layer with 4096 neurons twice and then into the output layer which has 1000 neurons, one for each of the output classes. These dense layers are denoted with 1 x 1 x neurons in Figure 4.

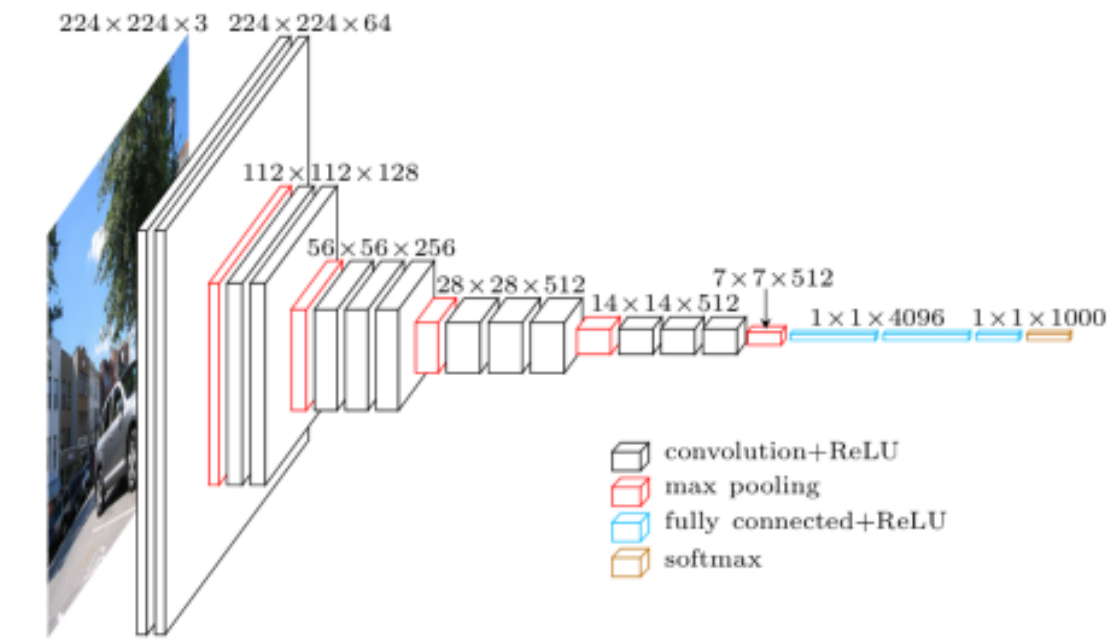

Figure 4: VGG16 Architecture [3]

2.3 Object Detection using Convolutional Neural Networks

The convolutional neural networks discussed so far in this paper have all focused on the section of computer vision known as image classification. That is, the model's goal is to output a classification for the image as a whole. This is not always useful in real life however, as often images contain multiple salient objects. This is where object detection becomes most useful. The goal of object detection is to identify both what the salient objects are and where they are in the image. For example, given an image of a busy city street, image classification would simply aim to predict that the image is of a city street. Object detection however, would aim to put what is known as a bounding box around each object of interest, for example, a person or a car, and to predict specifically what is in that one bounding box. One of the first object detection algorithms was invented by Paul Viola and Michael Jones and the goal of this algorithm was to detect human faces in real time [18]. This paper introduces three new techniques. The first of these, the authors call integral image. The idea behind integral image is that image intensities are not used, but instead a series of features which can be calculated in constant time and make prediction faster. The second is to select a small number of features using AdaBoost. The number of features that can be created by the first technique is far too many, when in reality, the authors state that only a small set of critical features are needed. The third contribution of this paper was detection cascades which the authors describe as "a method for combining successively more complex classifiers in a cascade structure which dramatically increases the speed of the detector by focusing attention on promising regions of the image." [18]

The next major improvement in object detection was the advent of the two stage detector. The first of these was designed by Ross Girshick [19]. When they invented the Regions with CNN method or as it is now known, R-CNN. The reason that this

model was called a two stage detector is because there are two steps in producing a prediction. The first step is to generate a set of object proposals, which are regions where the model believes an object of interest may be and then this image is resized and then passed into a CNN model to extract features. The second step is that a Support Vector Machine (SVM) classifiers are used to predict which object is in each region. R-CNN showed a huge improvement over previous object detection models in regards to accuracy, however it is incredibly slow due to some of its redundancies and the need to have multiple steps.

There were a number of other two stage models that increased in speed and accuracy such as SPPNet, Fast R-CNN and Faster R-CNN, however the event that lead to significantly faster recognition speeds was the advent of the one stage detector. The first singe stage detector was the YOLO algorithm [4]. YOLO stands for You Only Look Once as this algorithm does not predict bounding boxes and classes separately, but rather at the same time. It does this by breaking the image up into a grid, where each grid cell is responsible for predicting some number of bounding boxes in its cell. The boxes with a score over a certain threshold are then selected, however this often still leaves a large number of boxes, so a technique call non-maximum suppression (NMS) is utilized. NMS will get rid of boxes that have too much overlap with other boxes to help ensure that one object isn't being predicted on repeatedly.

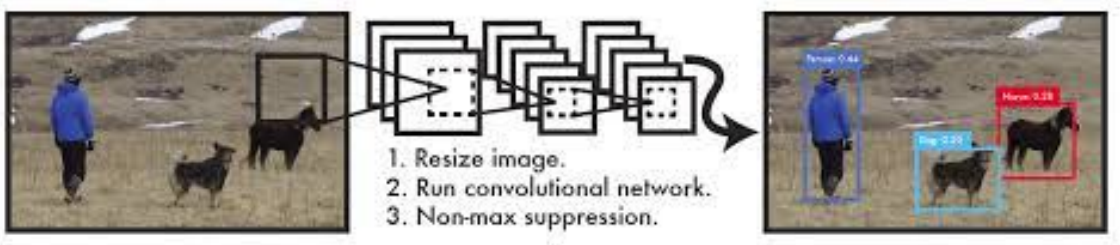

Figure 5: YOLOv1 Model [4]

The model that is utilized in this paper is YOLOv3-tiny [8]. This is a condensed

version of the third edition of the YOLO model. This model, while less accurate runs at a fraction of the time, which can be extremely important in object detection models, where the goal is often to have real time predictions. YOLOv3 has two main parts. First, a feature extractor. The job of the feature extractor is similar to that of a normal CNN, attempting to find patterns in the given image. The second part is a feature detector. The goal of the detector is to output the final images with bounding boxes around the salient objects. The feature extractor for YOLOv3 is the Darknet-53 CNN. This model has many repeated residual blocks, which are blocks of convolutional layers where the original input to the block is set aside and concatenated with the output of the block at the end. These residual blocks were first implemented by ResNet and were shown to greatly help with the effectiveness of very deep neural networks such as Darknet-53 [20].

After the Darknet-53 backbone, another 53 convolutional layers are added onto the model. Part of what makes YOLOv3 unique is its ability to make detections at 3 different scales. As can be seen in Figure 6, for the first 81 layers the model is downsampled repeatedly until it has been downsized by a factor of 32. Here, the first feature detector is utilized. The same is done as layer 94, except the model will be upsampled by 2 so that the image is downsized by a factor of 16 from the original image. This is done a third and final time at layer 106 where the image is upsampled one more time to be only a factor of 8 smaller than the original image. At each of these feature detectors, a detection kernel is applied, the shape of which is 1 x 1 x (B x (5 + C)), where B is the umber of bounding boxes to be predicted for the given cell and C is the possible number of classes. The 5 is in the formula for the 4 bounding box attributes and the object confidence. An example of one of these kernels can be seen in Figure 7

These kernels are then passed to the back end like the one described for YOLOv1,

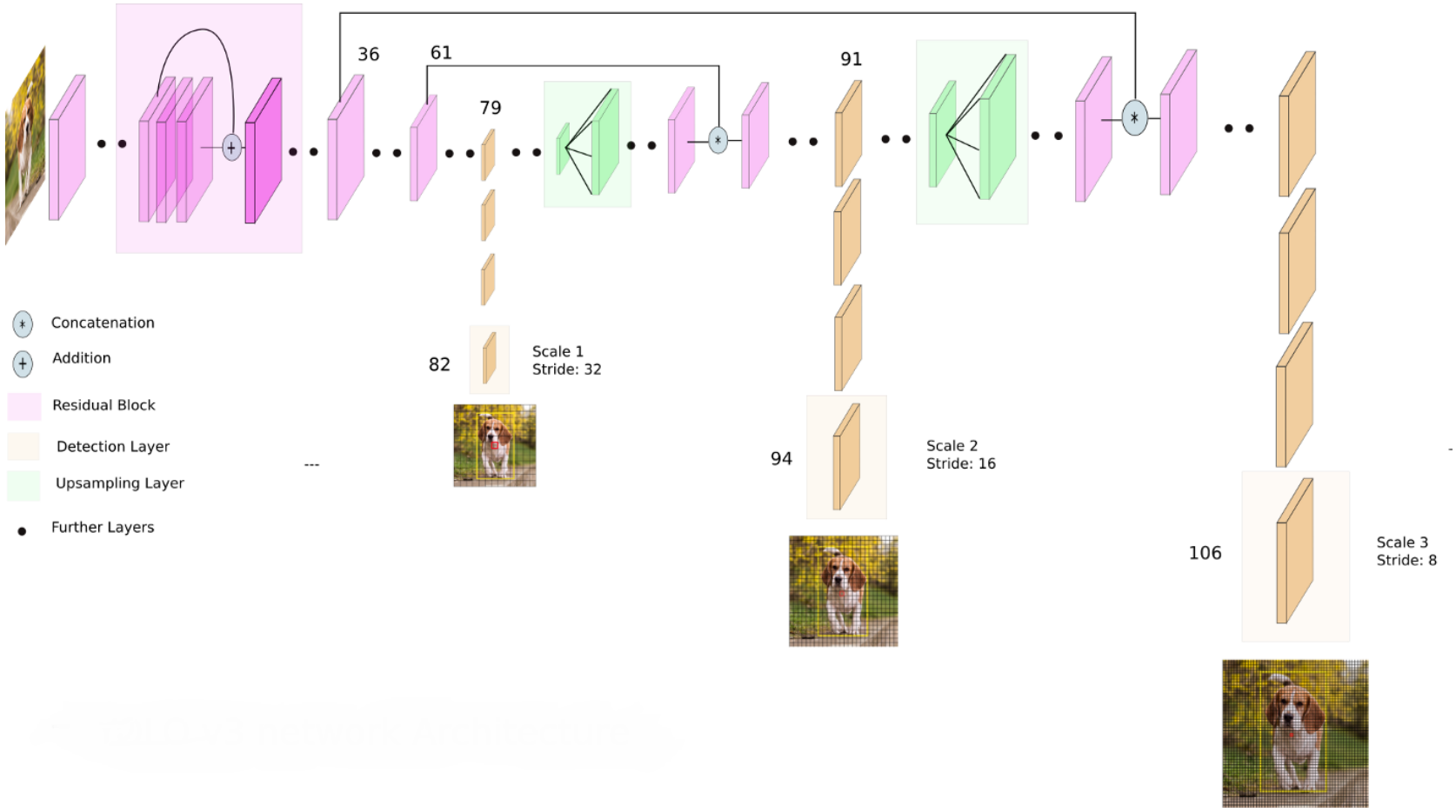

Figure 6: YOLOv3 Model [5]

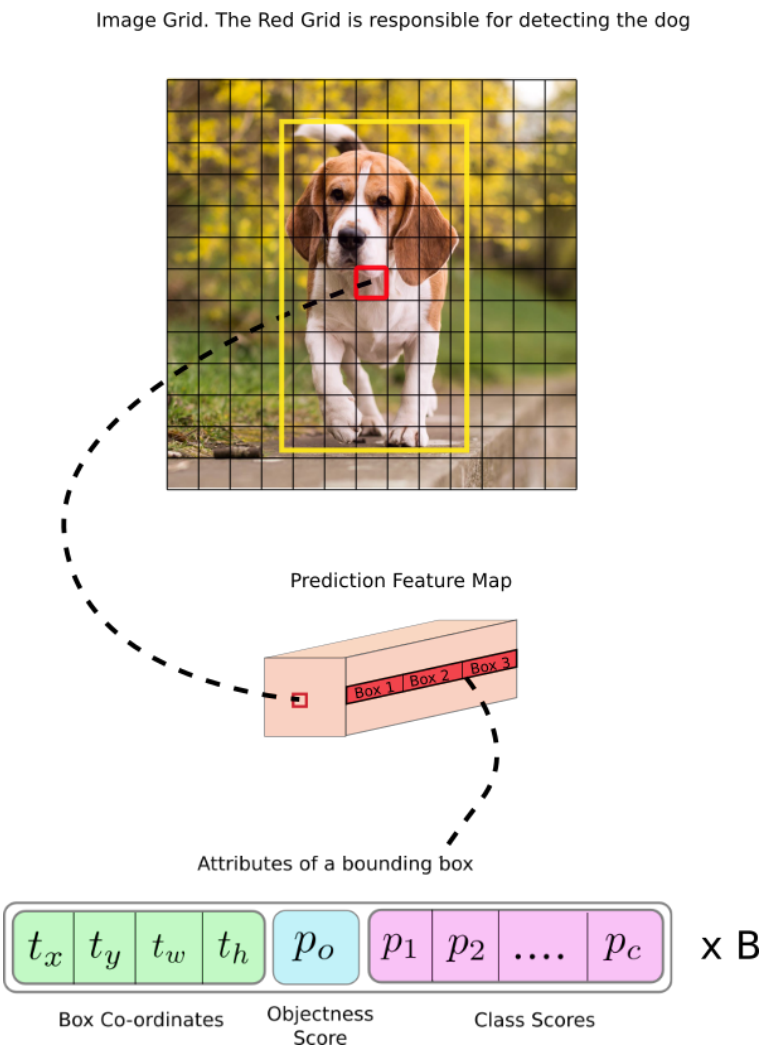

Figure 7: YOLOv3 Detector [5]

where the boxes are first filtered by an object threshold, where their "objectness" meet the given threshold and are then passed through the NMS threshold to ensure there are not multiple boxes surrounding the same salient object. Then the object with the highest class softmax score is chosen, given it meets the class threshold. This is the final output for the given box.

2.4 Bayesian Neural Networks

Bayesian neural networks (BNNs) seek to overcome a pivotal shortcoming of traditional neural networks which is that they have no way to express their level of uncertainty, often making them seem overly confident in predictions in which there may actually be little confidence. They do this by implementing uncertainty onto the weights themselves in the neural network, replacing typical point estimates with distributions which are randomly sampled from. The first paper to introduce a Bayesian neural network was written in 1989 by Tishby [21]. The focus of Tishby's paper was to address the issue that often neural networks perform well on training data, but perform poorly on test data that are not from the exact same distribution as the training data. This paper developed a method to measure the performance outside of the training set, using only the training set by utilizing the average statistical prediction error. This information was then used to develop an optimal network architecture. The next paper to make a significant impact using BNNs was from Denker and LeCun [22]. This paper proposed a method to take the output vector from a traditional neural network and apply traditional statistics methods to it to transform it into a meaningful output that reflects the level of certainty in the response.

For Bayesian neural networks, it is often useful to consider Variational Inference (VI) when approaching the concept of backpropagation. Variational Inference is able to transform backpropagation into an optimization problem for Bayesian neural networks utilizing, most frequently the Kullback-Liebler (KL) Divergance which measures the difference between the variational distribution and the true posterior. The equation for KL-Divergence can be seen in Equation 1 where $q_\theta(\omega)$ is the density over the set of parameters ω. This distribution is limited to a certain family of functions which can be parameterized by θ. The true posterior is given by $p(\omega|D)$.

$$KL(q_\theta(\omega)||p(\omega|D)) = \int q_\theta(\omega) log \frac{q_\theta(\omega)}{p(\omega|D)} d\omega \tag{1}$$

The first paper to utilize variational inference for Bayesian neural networks was written by Hinton and Van Camp who were attempting to use Bayesian neural networks to solve the problem of over fitting in traditional neural networks [23]. Unfortunately, the authors were unable to adequately "capture the posterior correlations between parameters," as mentioned by Barber and Bishop's 1998 paper [24]. Part of this issue was the fact that they assumed the distribution of the model's weights were independent, when this is actually fairly uncommon in neural networks. Barber and Bishop attempted to rectify this by using a full-covariance Gaussian distribution to capture the covariances between the weights. By doing this, however, they found that the the number of weights scales quadratically to the number of weight distribution parameters, so more constraints need to be added to limit this explosion of parameters. They do manage to find success doing this, however, their model was extremely small at only four hidden layers.

After these first papers addressing VI, there was not widespread use of this method for almost two decades, which was, according to Graves, "due to the difficulty of deriving analytic solutions to the required integrals over the variational posteriors" [25]. In his 2011 paper, Graves posits that the correct manner to tackle this problem is to forgo looking into the true posterior and to instead analyze the variational posterior as it is easier to utilize numerical methods on due to the fact that it is easier to draw samples from. Graves finds a result that can be utilized on any log-loss model, which he notes applies to most neural networks. The result is a stochastic method for variational inference with a diagonal Gaussian posterior. After this revival of the Bayesian neural network, one of the most promising lines of research was started by Blundell, called "Bayes by Backprop" [6]. This algorithm shows how to

find unbiased estimates of the derivative of an expectation. It does this by using a reparameterisation trick to find the expected lower bound of a given function. Using this trick, Bayes by Backprop is able to minimize the KL divergence between the approximate and true posterior. An example of the distributions on the weights can be seen in Figure 8 from Blundell's paper.

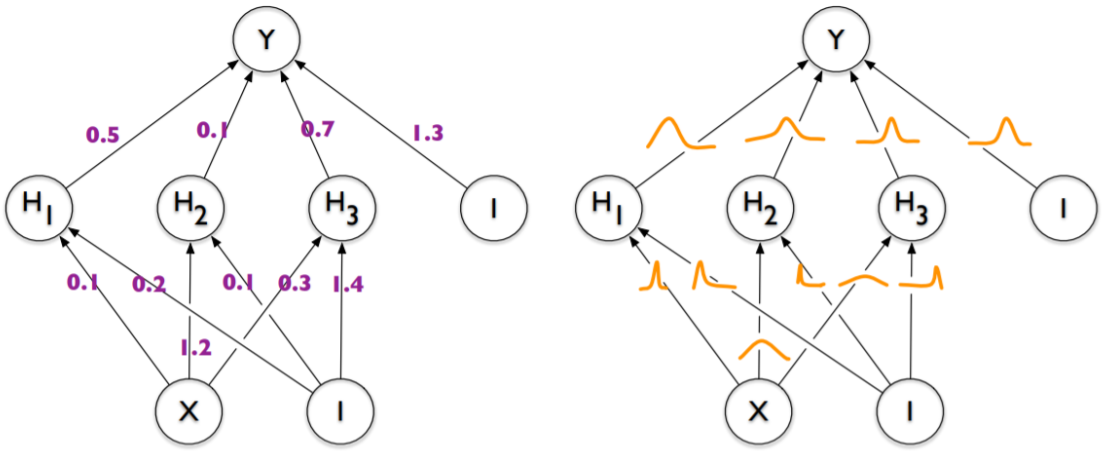

Figure 8: Point Estimates Compared to Distributions on Weights [6]

One of the first papers to implement Bayesian thinking into convolutional neural networks was written by Gal and Ghahramani in 2016 [26]. They found that the posterior they wished to utilize was intractable and therefore decided to utilize dropout in the forward pass which can be an appropriate substitution for placing uncertainty on weights. They find that this method shows a considerable improvement when compared to traditional neural networks in regards to classification accuracy. Another example of adding Bayesian thinking to neural networks can be seen in the paper by Goan and Fookes written in 2020, where they implemented the Bayes by Backprop algorithm to the kernels of a CNN, specifically LeNet-5 [27]. This was compared to regular LeNet-5 and found to have only a marginal drop in accuracy, but with the benefit of having uncertainty levels on predictions, which is the entire point of applying Bayesian thinking to neural networks.

The final paper to mention regarding Bayesian neural networks is written by Wen, et all. and it developed the method utilized in this paper, called Flipout [28]. The authors state that weight perturbation algorithms, such as Bayes by Backprop suffer from high variance of gradient estimates due to the fact that all of the mini batch samples will have the same perturbation, which causes correlation between gradients. The authors use Flipout as a method to be able to perturb these weights independently within each mini batch. Flipout does this by multiplying a base perturbation by a different rank-one sign matrix for each sample. This leads to decorrelated gradients that still have a marginal distribution that is the same as the distribution that would have been calculated using weight perturbations. The second benefit of this method is it means that mini batches can be more effectively utilized, meaning that this method is significantly faster than similar techniques. This is in part because Flipout can be expressed in terms of matrix multiplications, meaning it can be more efficiently implemented by a graphics processing unit GPU.

2.5 Previous Work on Model Uncertainty

There are a number of papers which have attempted to tackle various aspects of uncertainty regarding the output of neural networks. The two novel methods proposed in this paper stem from two different techniques, prediction smoothing and class threshold variance. In *Deep learning based surgical workflow recognition from laparoscopic videos*, Kurian, et. al. utilize a rolling average of the outputs to help get rid of flickering for the outputs of a CNN predicting on videos of surgery [11]. Kurian, et. al. utilize a standard CNN, meaning that there is only one prediction per frame throughout the video. These predictions are averaged over the previous 16 frames to give the final prediction. This thesis builds on the ideas presented in this paper by implementing Bayesian layers so that the outputs from each frame being

averaged are distributions instead of point estimates. The authors do find success utilizing their model, which is promising for the use of prediction smoothing.

In *A comparison of the performance of 2D and 3D convolutional neural networks for Subsea Survey Video Classification* by Stamoulakatos, et. al., the authors show that utilizing a simple rolling average of CNN predictions can outperform a 3-Dimensional CNNs [29]. Utilizing a 3-Dimensional neural network or other similar techniques such as a bi-directional long short-term model (LSTM) require an entire video as an input. This means that the model is unable to be utilized in real time as it needs all frames at once. It would seem intuitive that these methods would perform better than a rolling average, however this paper shows the opposite. This paper gives reason to believe that despite the simplicity of a rolling average on a standard neural network, there is still reason to utilize them. Our research takes this one step further to utilize a BNN to generate a weighted average of output distributions instead of simply utilizing the averaging of point estimates.

Regarding varying class thresholds, *A threshold-varying artificial neural network approach for classification and its application to bankruptcy prediction problem* by Pendharkar discusses changing the class threshold in the context of binary classes [30]. This paper looked at moving the threshold for which to classify an instance as bankruptcy in order to increase accuracy. This thesis expands on this first and most importantly by changing the model from a standard neural network to a BNN. In the standard neural network, the output being compared to the threshold is that from a softmax output which is a poor approximation for actual confidence. In the BNN, the output is a distribution much more closely resembling the model's actual level of confidence on its prediction. Second, the model looks at changing class thresholds not simply in a binary classification, but in the multi-class case. Third, this thesis looks not simply at changing class thresholds for the sake of an increase in accuracy,

but also for the sake of avoiding certain adversarial classes.

In *Training cost-sensitive neural networks with methods addressing the class imbalance problem*, Zhou, et. al. address the effect of threshold moving in training cost-sensitive models [31]. This paper does this by moving the threshold for each class so that the model is more likely to select less expensive options. The results of this paper show that the authors struggled to show any significant improvement of the baseline with their techniques. This paper again falls short in that it utilizes the softmax output of a traditional neural network, which is not a satisfactory approximation for confidence. It is in large part due to this difference that we believe this thesis has merit when compared to previous work studying similar problems.

III. Methodology

This chapter provides a description of the data utilized, a detailed explanation of the front and back end architectures utilized, a summary of the hyperparameter tuning and an overview of how the model is evaluated for both the original proposed model and the secondary proposed model.

3.1 Original Investigative Question

This section discusses the original investigative question. Unfortunately, there was some difficulty with the training of the front end YOLOv3-tiny model that caused this model to not find the desired success. That being said, even though the front end was unable to train adequately, we still feel that the proposed back end has merit and is worthy of discussion.

3.1.1 Description of Data

The data utilized to train the original front end vanilla model is from the ImageNet Large Scale Visual Recognition Challenge 2015 (ILSVRC2015). The full ILSVRC2015 training data set includes 3862 snippets, or videos, yielding 866,870 bounding boxes for 1,122,397 frames. There are 30 total classes. For each class, there are between 56 and 458 snippets that include it with a median of 116. For each snippet there are between 6 and 5,493 frames with a median of 180. For the validation data set there are 555 snippets that yield 135,949 bounding boxes in 176,126 frames. There are between 4 and 64 snippets per class with a median of 17 and between 11 and 2,898 frames per snippet with an average of 232. The videos have different sized frames, however all were resized to be 416 by 416 pixels for the model. For the sake of training, there were simply too many training and validation images to be practical

for the hardware utilized.

The videos were first split into training, validation and testing sets to ensure that there was no data leakage between them. This is because from frame to frame, there is not much change, so including frames from the same video in multiple sets would likely indicate the model is performing better than it truly is. From the videos set aside for the training set, 2,000 frames were taken randomly for each class from the training set for training the model and 200 frames from each class randomly from the validation set for validation of the model. It is safe to take randomly from these videos now as the train test split has already occurred. This ensures that images from more different videos are selected, instead of just taking all of the frames from the first video, then the second video and so on. All classes had the same number of frames for both the training and validation sets. The rest of the videos were left for the test set.

3.1.2 Model Overview

For analysis of the original investigative question, two models are utilized. The first is YOLOv3-tiny as mentioned in the background section of this paper. It is exactly the same as the original. The second is a novel model developed in this paper. For this model there is a distinct front end based on YOLOv3 and a distinct back end that transfers information from frame to frame. Both the model front end and back end are explained in the following sections.

3.1.3 Front End Model Architecture

The front end of the initial proposed model is exactly like the YOLOv3-tiny model until the two final output layers. The model architecture of YOLOv3-tiny can be seen in Figure 9. The model has 22 layers including the two output layers for the two

different sized grids, which for YOLOv3-tiny are 13 by 13 and 26 by 26. The model is primarily repetitions of a 3x3 convolutional layer, a maxpooling layer, and then the leaky ReLu activation function, with extra convolutional layers added before each output for the two grid sizes. The number of filters for the convolutional layer increases by a factor of 2 for each of the first 6 repetitions of this grouping of layers starting at 16 filters. The difference is in the last layer before each of the two predictions. The last two convolutional layers are Bayesian layers, meaning that instead of their weights being a point estimate, they are distributions that are randomly drawn for each prediction. This is done by utilizing the Convolution2DFlipout layer from tensorflow's probability package (tfp) which uses the same method from Wen, et al. [28] as discussed in the background section. The route layers seen in Figure 9 are the same as the residual blocks mentioned in subsection 2.3. These layers directly pass information from the layer mentioned to the current layer. The YOLO layers mentioned are the feature detectors mentioned in subsection 2.3. On each image, a certain number of predictions are made to generate a distribution of predictions of the classes. For the bounding boxes, the x and y coordinates are simply be averaged to generate the final bounding box. This distribution of classes connected to the given bounding box is what is passed to the back end of the model.

The reason for using YOLOv3-tiny instead of the full sized model is because it trains and predicts significantly faster. While YOLOv3 can make predictions at 35 frames per second on 416x416 images, YOLOv3-tiny can make 220 predictions per second [32]. This is 6.28 times more frames per second. A comparison of a number of different models trained and tested on the COCO data set show that YOLOv3 tiny is the only model to attain more than 100 FPS, besides another tiny YOLO model. This comparison includes models such as the SSD models, Retinanet and R-FCN. It is especially crucial for the model to be fast given that for the Bayesian

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figure 9: YOLOv3-tiny Model [7]

version of this network, multiple predictions have to be made per frame to generate the distribution. Unfortunately, this does come with a loss in accuracy dropping from 55.3 mAP for the full sized model to only 33.1 mAP for the tiny model [32], however as the focus of this paper is on comparing a vanilla object detection model vs a BNN object detection model, this trade off was determined to be acceptable.

The YOLOv3-tiny loss function is the same loss function utilized in the full sized YOLOv3 model. The loss function can be seen in Equation 2. For training, the sum of the squared error loss is utilized. $loss_{xy}$ is the loss due to the error in the prediction of the location of the bounding box. $loss_{wh}$ is the error in determining the size of the bounding box. $loss_{class}$ is the error in determining the class of the object in the bounding box. $loss_{confidence}$ is the loss due to the confidence score of the prediction.

$$loss = \frac{1}{n}\sum_{i=1}^{n} loss_{xy} + \frac{1}{n}\sum_{i=1}^{n} loss_{wh} + \frac{1}{n}\sum_{i=1}^{n} loss_{class} + \frac{1}{n}\sum_{i=1}^{n} loss_{confidence} \quad (2)$$

The output of this model is a prediction for the bounding box as well as an objectness score. The format for the bounding box can be seen in Figure 10 and the variables in this figure are as follows. t_x and t_y are the outputs from the model corresponding to the location of the box with respect to the cell it is centered in. t_w and t_h are the outputs from the model corresponding to the size of the bounding box, refering to width and height respectively. The last output from the model is t_o which is the model's output regarding the level of certainty regarding the existence of an object in the bounding box. These are all transformed according to the formulas in Figure 10 to be usable by the backend model.

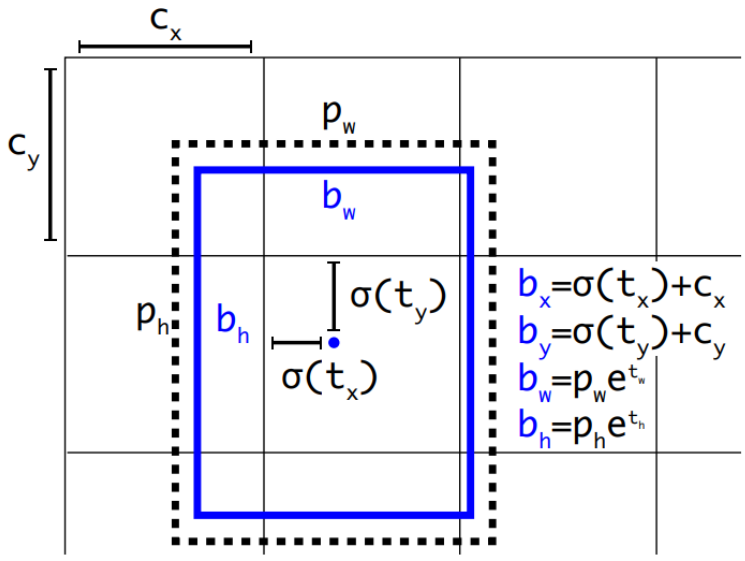

Figure 10: Bounding Box for YOLOv3 [8]

3.1.4 Back End Model Architecture

For the control model, the regular YOLOv3-tiny is utilized without a back end. The prediction on that image is simply the output from the model. However, for our novel proposed model, we add a back end to incorporate the Bayesian aspects to the model. As mentioned in the Front End section, the information passed to the back end for each frame is the average bounding box for each predicted object in the frame as well as the distribution of predicted classes for each bounding box. The prediction for each bounding box can be seen in Equation 3

$$\boldsymbol{Y_{n_i}} = \alpha * \boldsymbol{Y_{n-1_i}} + (1 - \alpha) * \boldsymbol{X_{n_i}} \tag{3}$$

Where $\mathbf{Y}_{n_i}$ is the final prediction for the current frame for box i, $\mathbf{X}_{n_i}$ is the output from the model for the current frame for box i, $\mathbf{Y}_{n-1_i}$ is the final prediction from the previous frame for box i and α is a constant that determines how influential previous frames are. This means that the current frame is not only taking the predictions form the current frame to make the final prediction, but also the predictions from frames before the previous frame as well. This is done under the assumption that an object will not suddenly become another object during a video, so it is safe to continue using the previous predictions to build an even stronger idea of what the true distribution should look like for the given object. The tuning for α is discussed in the section regarding back end hyperparameter training.

In order to get the distributions from the prior frame, each box from frame $n-1$ must be connected to a box in frame n or must be intentionally ignored. This is done by calculating the distance from the four corners of a bounding box in frame n to the four corners of each bounding box in frame $n-1$, summing them and then repeating for each box in frame n. The bounding boxes in the current frame are

matched to those in the previous frame such that the sum of the squared differences are minimized. This of course has the potential to incorrectly match boxes. The most obvious scenario in which this may occur is if there is an object in the middle of the image and an object on one side in one frame and in the next frame the object in the middle is still there, however the object on the side has moved out of the frame and a new object has moved into the frame on the other side. In this event, the backend model may attempt to match the center box with the new object and the old object to the center box. In order to prevent this, there is a constraint that the centroid of the bounding boxes cannot be more than a certain percentage of pixels away from the previous bounding box. This adds another assumption that objects do not move too quickly, which for most data sets is likely true.

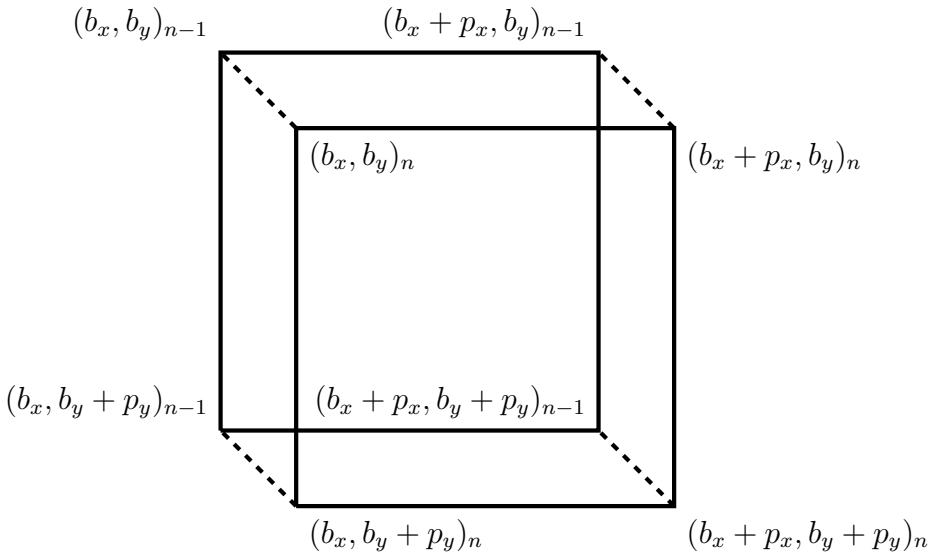

Figure 11: Distance Between Frame n and Frame $n-1$

The equation for the distance between each possible pair of bounding boxes in frames n and $n-1$ then can be seen in Equation 4.

$$
\begin{aligned}
distance = & \sqrt{(b_{x_n} - b_{x_{n+1}})^2 + (b_{y_n} - b_{y_{n+1}})^2} \\
& + \sqrt{((b_{x_n} + p_{x_n}) - (b_{x_{n+1}} + p_{x_{n+1}}))^2 + (b_{y_n} - b_{y_{n+1}})^2} \\
& + \sqrt{(b_{x_n} - b_{x_{n+1}})^2 + ((b_{y_n} + p_{y_n}) - (b_{y_{n+1}} + p_{y_{n+1}}))^2} \\
& + \sqrt{((b_{x_n} + p_{x_n}) - (b_{x_{n+1}} + p_{x_{n+1}}))^2 + ((b_{y_n} + p_{y_n}) - (b_{y_{n+1}} + p_{y_{n+1}}))^2}
\end{aligned}
\tag{4}
$$

This is calculated for each possible pair and frames are matched to minimize the sum of the differences. These are the final predictions for the given frame.

3.1.5 Hyperparameter Tuning

There are a number of hyperparameters to be tuned in both the front end and the back end of this model. The hyperparameters are tuned for the front end first, and then the resulting hyperparameters are what is utilized to train the model's back end.

In the front end of this model, the hyperparameters that are tuned include the object threshold and the class threshold. The object threshold determines which bounding boxes have a high enough objectness score to pass to the next step. The objectness score is reflective of how confident the model is that there is an object in the bounding box. The class threshold performs similarly but only allows through boxes that have specifically predicted that any class in the bounding box is present with above a certain threshold. The reason for this second threshold is that the model may be confident that an object is present in the box, however the object may not be one of the 30 possible classes. We do not want a bounding box in this location as it is not possible for the model to accurately predict what is in the box since it is not in the class list. This is tuned with the help of mean accuracy precision (mAP)

which is discussed in Section 3.6.

In the back end, the hyperparameter that is tuned is α, which is the weight given to the final prediction of the previous frame. When tuning α, it is important to note that it effects not only the weight given to the previous frame, but also how long previous frames stay relevant for. As can be seen in Table 1, when $\alpha = 0.1$, the weight on the previous frame is already below 0.01, whereas when $\alpha = 0.4$, even the frame 3 frames ago still has a weight greater than 0.01, meaning frames stay relevant for longer. The weight on a given frame n, w_n, can be seen in Equation 5

$$w_n = \begin{cases} 1 - \alpha & \text{if } n = 1 \\ w_{n-1}\alpha & \text{if } n > 1 \end{cases} \tag{5}$$

Table 1: Affect of α Value on Frame Weight

Frame	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$
n	0.7500	0.5000	0.2500
$n-1$	0.1875	0.2500	0.1875
$n-2$	0.0469	0.1250	0.1406
$n-3$	0.0117	0.0625	0.1055
$n-4$	0.0029	0.0313	0.0791
...	...	...	...

For the actual training of this model, the first important hyperparameter to mention is the optimizer. The optimizer utilized for training this model originally was stochastic gradient descent and as such, we use this optimizer as well with the same hyperparameters utilized in original YOLOv3 paper. These are a learning rate of 0.001, a momentum of 0.9 and a decay of 0.0005. This paper also tests the Adam optimizer, known for it's speed in training with different learning rates including 0.1, 0.01, 0.001 and 0.0001. The number of epochs was somewhat constrained by time as there are a large number of classes and samples for each class and was therefore set at 100 epochs. This is because each epoch on it's own takes nearly an hour to train.

The size of the mini batches was set to 128 as that was the largest number with a power of 2 to fit on our GPU. Powers of 2 are common number to select for neural network training as GPU and CPU memory normally stores memory in powers of two. This is a common practice seen in nearly all neural network papers. Using the largest possible batch size has two benefits. The first is speed. The larger the mini batch size, the less mini batches needed, the faster the model can train. Second, there is some evidence to show that larger batch sizes can have an increase in accuracy [33].

3.1.6 Model Evaluation

The model is primarily evaluated based on mAP. To calculate this, the first step is to create a precision/recall curve based on utilizing an Intersection over Union (IoU) of greater than 0.5 for every class. The IoU is a measure of the overlap of two bounding boxes. In this case, they are the predicted bounding box and the truth bounding box. This is done by matching predictions to the ground truths and sorting them by confidence. Here, precision and recall can be calculated for each prediction and ground truth combination. This curve is then altered slightly to be monotonically non-increasing by setting the precision value at least as high for a given point as any points with higher recall than it. For example, see Figure 12. The area under this curve is then computed numerically. After this, a mAP is calculated by taking a weighted average for each class based on how many times the given class occurred.

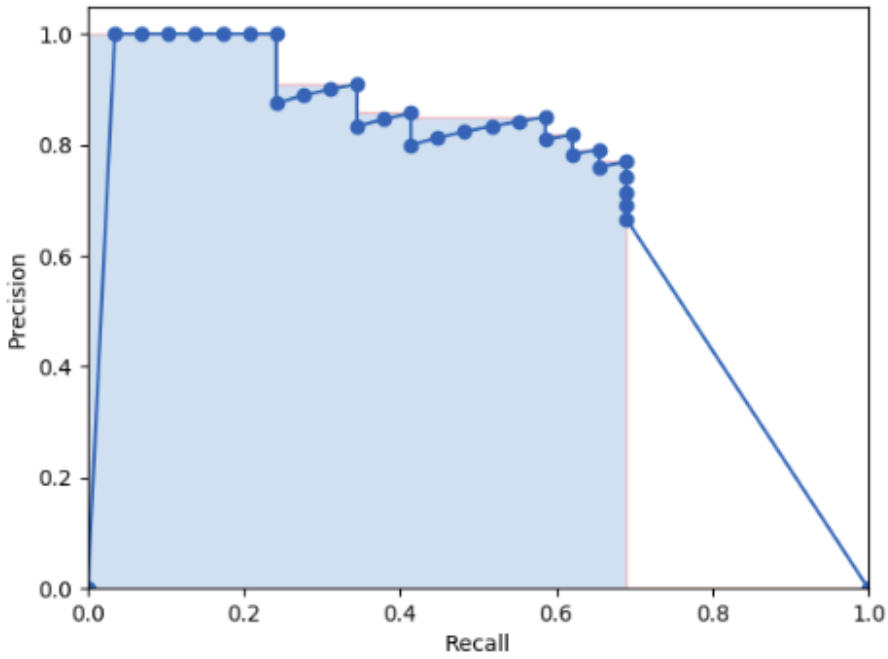

Figure 12: An Example of a Precision Recall Curve for mAP

3.2 Revised Investigative Question

Due to the difficulties training the original two models, two new models were examined. These revised models focus only on image classification, a simpler task than object detection.

3.2.1 Description of Data

The data utilized for this model is from the UCF101 data set. This data set is composed of 13,320 videos from 101 different action classes. These videos have average durations as can be seen in Figure 13 and the distribution of video lengths for the different classes can be seen in Figure 14.

The purpose of the UCF101 data set was to establish a data set that had videos of realistic actions. As mentioned on the UCF101 website, most available action recognition data sets are videos of staged actors performing actions, which is not a

realistic depiction of what an image classification model would expect to see when deployed.

For this model, only 10 classes were selected: brushing teeth, drumming, horse race, horse riding, ice dancing, jump rope, punch, rock climbing indoor, rowing, and shaving beard. Examples of each of these classes can be seen in Figure 15.

As mentioned for the original investigative question model, this data was split in such a manor to ensure that frames from a given video will only be in one given set to ensure there is no data leakage. The videos were split into training, validation and testing sets first and then frames were taken from those splits to make the final sets. For this data set, for the given classes, all frames were utilized. This did cause some imbalance in the number of frames per class, but it was not a large difference and as will be seen in Chapter 4, this did not cause any issues with a large class skewing the model predictions towards itself.

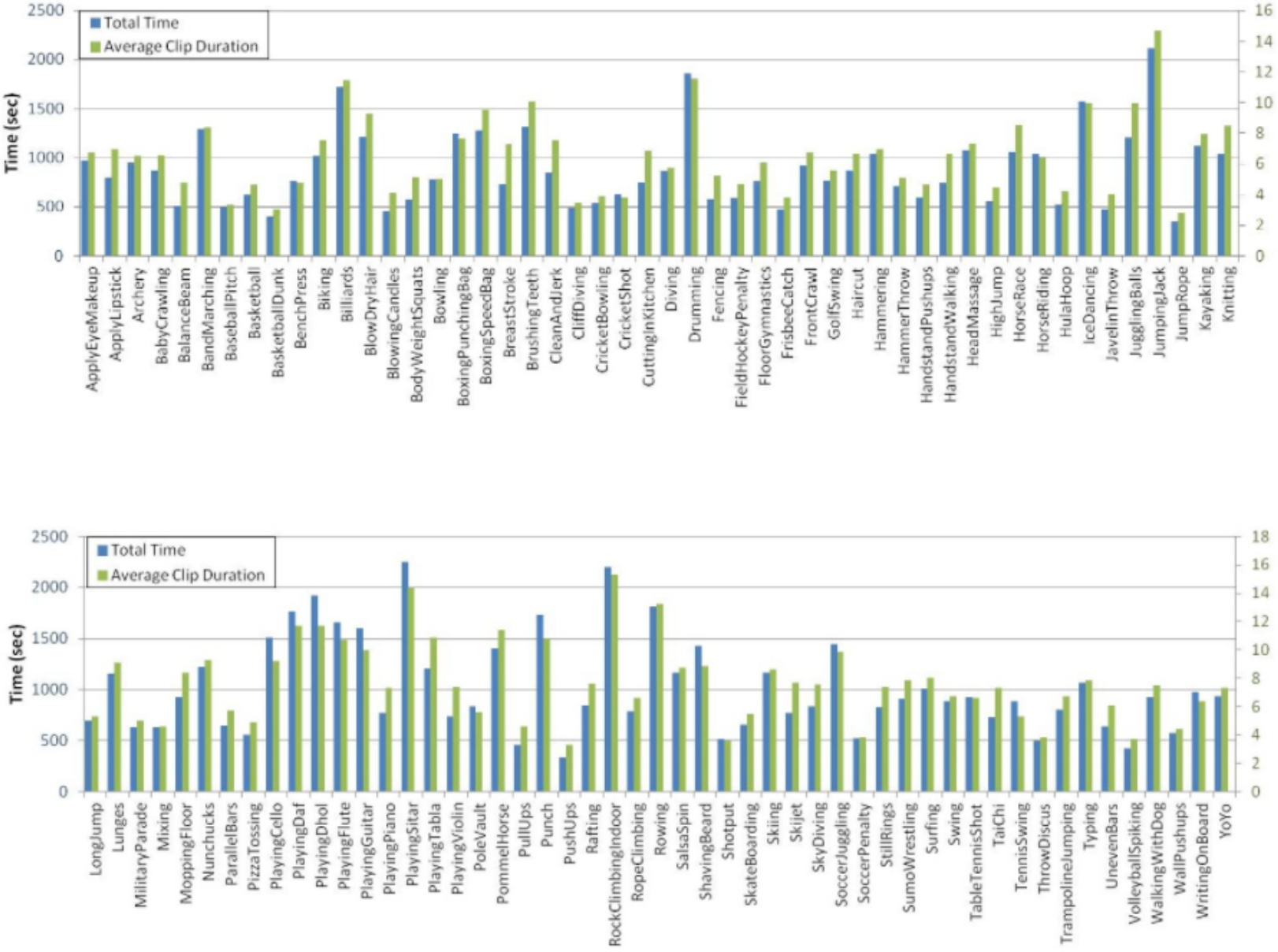

Figure 13: Durations for UCF101 Classes

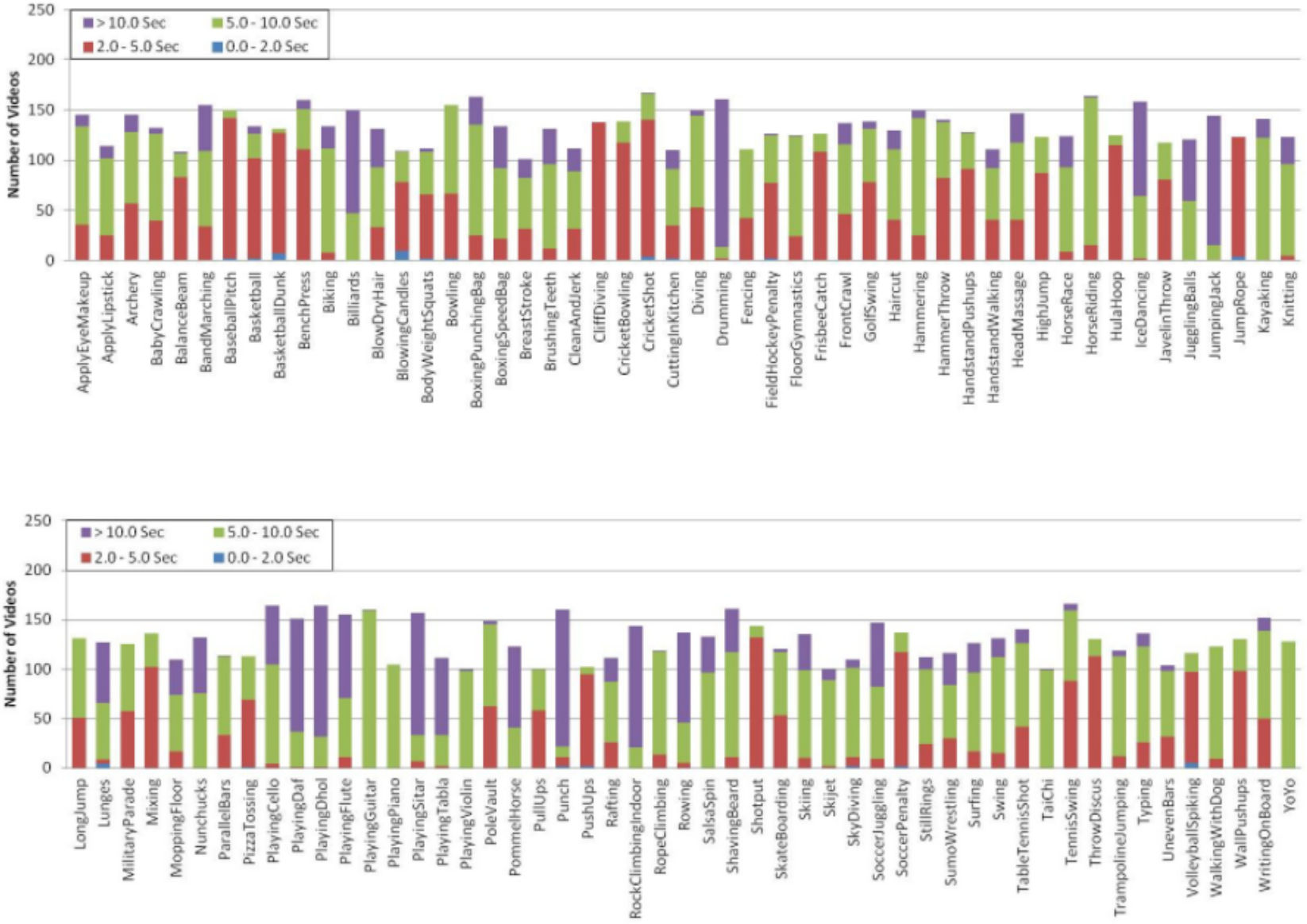

Figure 14: Number of Videos with Certain Durations for UCF101 Classes

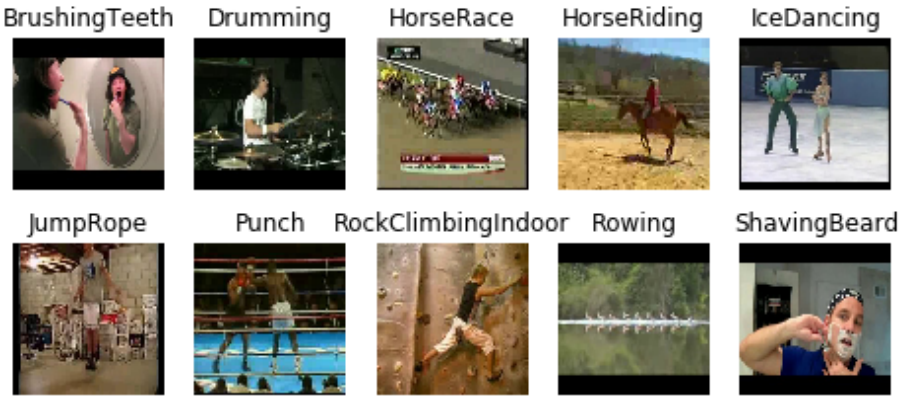

Figure 15: Examples of UCF101 Classes

3.3 Revised Investigative Question

For analysis of our revised investigative question, two models are utilized. The first is VGG16 as mentioned in the background section of this paper. It is nearly exactly the same as the original and deviations from the original are detailed in the following subsection. The second is the novel Bayesian model that is developed in this paper. For this model there is a distinct front end based on VGG16 but with a Bayesian layer and a distinct back end that transfers information from frame to frame that are explained in the following section.

3.3.1 Front End Model Architecture

The front end of this model is exactly the same as VGG16, except instead of the fully connect layers having 4096, 4096 and 1000 neurons respectively, they now have 512, 256, 128 and 10 neurons. This is due to the fact that the original VGG16 model was trained on the ImageNet data set which has 1000 classes, whereas this model was trained on only 10 classes, meaning therefore less neurons are needed. Including as many neurons as were seen in the original wold likely lead to overfitting. The VGG16 model is made of up repetitions of convolutional layers with a ReLu activation function with a kernel size of 3x3 and maxpooling layers which downsize the image as it moves through the model. Each time the image is scaled down from a maxpooling layer, the next set of convolutional layers have twice as many filters. The model start by taking in a 224x224x3 image and downsizes it until what is passed to the fully connected layers is a 7x7x512 array. This is then flattened and passed into the fully connected layers. The final prediction layer is passed through a softmax function for the final output. In our model, a dropout rate of 0.5 was added to these layers to help prevent overfitting.

Similarly to the novel model from our original investigative question, the novel

model for the revised investigative question is initially exactly the same as the above VGG16 model, but with a Flipout layer put in place of the last layer in order to generate the output distributions required for the backend. The output of this model is a a distribution of image classifications of one of the classes listed in the Data Description section, one prediction for each time the model is predicted on.

3.3.2 Back End Model Architecture

The simple VGG16 baseline model has no back end as the output of that model is simply the prediction. The secondary proposed model has a back end that is very similar to the original proposed model's back end. For the back end of this model, each frame is predicted on some number of times. These predictions were then be utilized in the same manor as seen in Equation 3. The difference is that instead of each prediction being object specific, they are frame specific which gets rid of the need for the distance function shown in Equation 4.

3.3.3 Hyperparameter tuning

There are a number of hyperparameters to be tuned in both the front end and the back end of this model. The hyperparameters are tuned for the front end first, and then the resulting best model is what is utilized to train the model's back end.

For the front end, the only hyperparameter that is tuned is the optimizer utilized. The optimizers that are analyzed are Adagrad, Adam, Adamax and NAdam. These optimizers are selected primarily for their speed when training models. For the backend, the hyperparameters that are tuned are α and the number of predictions per frame. When analyzing α, the primary concern is accuracy. When analyzing the number of predictions per frame, both accuracy and speed in frames per second (FPS) are analyzed.

3.3.4 Model Evaluation

The model is evaluated based on the average accuracy for all of the frames. The predicted answers are each compared to their corresponding truth label and are assigned a 1 if correct and a 0 if incorrect. These numbers are summed and then divided by the total number of frames to determine the overall accuracy of the model. This is the same for both the standard and the Bayesian model.

IV. Results and Analysis

In this chapter, the result of both the original and revised investigative questions are discussed, including the results of the hyperparameter tuning primarily for the revised investigative question.

4.1 Original Investigation Question Results

Unfortunately, the Yolov3-tiny model front end failed to successfully train, meaning that the back end was unable to be adequately tested. The results showing the mAPs for the different hyperparameter tunings for the base original model can be seen in Table 2 below. The best hyperparameter setting was an object threshold of 0.15 and a class threshold of 0.15. This setting still only managed a 2.73% mAP, whereas the yolov3 model, as mentioned in Chapter III, should be seeing a mAP in the low 30% range. Due to this poor performance, the backend model was unable to be adequately tested. The backend developed assumes a sufficiently accurate frontend with which to carry forward information. If the information from the previous frame is rarely accurate, there is no point in carrying it forward.

Table 2: Affect of Hyperparameters on mAP

Object Threshold	Class Threshold	mAP
15	15	2.73%
	25	0.49%
	35	0.49%
35	15	0.46%
	25	0.46%
	35	0.45%

A number of attempts were made to make alterations to this model to achieve better training accuracy, however none were successful. Different learning rates were tested as mentioned in the previous chapter and even with the largest learning rate,

the model was still improving, implying that it had not hit a local minima, but it was training so slowly that we simply did not have the computational power to make training this model feasible. After training the model for the initial 100 epochs, we trained it for 100 more and the results of this can be seen in Figure 16. Both the training and validation losses still appear to be decreasing, but at such a slow rate that reaching a low enough loss would have taken thousands of epochs more.

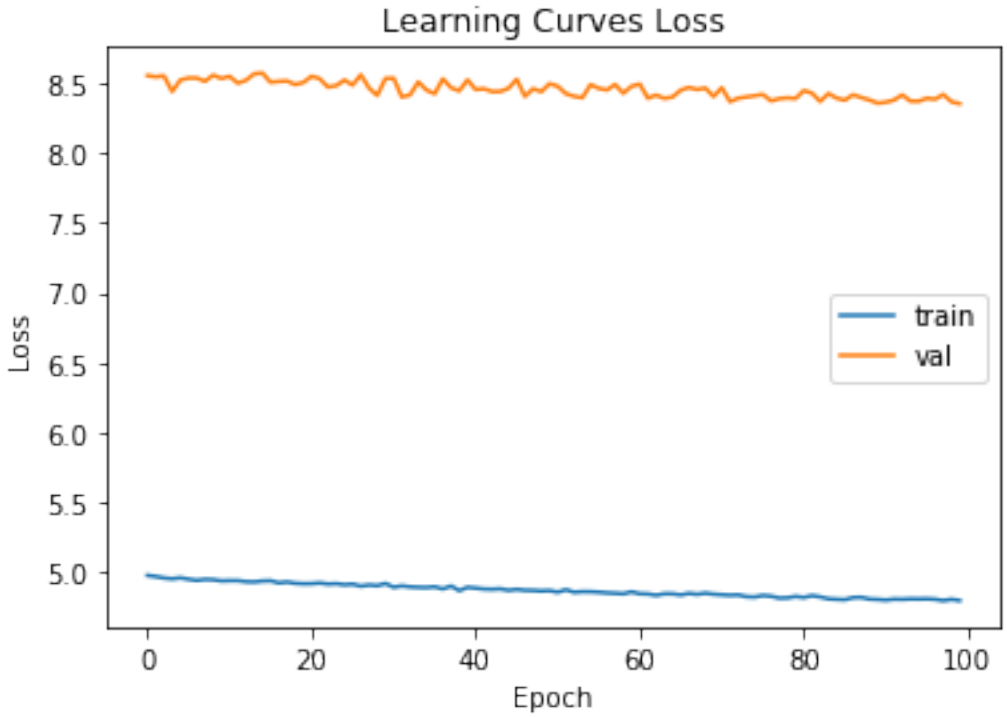

Figure 16: Training and Validation Loss During Final 100 Epochs

After seeing that the model with all 30 classes was not going to train with the computational power we had, we attempted to solve this problem by limiting the number of classes. It is inherently harder to train a neural network with a large number of classes. Also, with the original large number of classes, not many frames were utilized per class due to the limited space on our GPU. The number of classes was reduced from 30 to 6. This is one-fifth of the size and as such, 5 times more images were utilized for each class, meaning each class had 10,000 images. This

unfortunately had no impact on the accuracy of the model. The model continued to fail to achieve any mAP over 3%, rendering itself useless for attempting to show the benefits of the proposed back end. This model showed a nearly identical pattern in training and validation loss as the model with all 101 classes where the model was still learning, but at a rate that was too slow to continue with.

There are a few reasons this may have occurred. First, we simply did not have the computational power to train this neural network. This is certainly a possibility. The model was still appearing to train, although slowly even 200 epochs in. It is possible that to get to an accurate model thousands of epochs of training may have needed to occur. A second possibility rests on the specific anchor boxes utilized. For this paper, the pre-trained anchor boxes for YOLOv3-tiny were utilized. This was done with the knowledge that all of the given classes in the ILSVRC video data set are also in COCO, on which YOLOv3-tiny was originally trained, or have an object for which we would expect a very similar anchor box due to the shapes of the classes. For example, there is no antelope in the COCO data set, while there is in the ILSVRC data set, but COCO does have other animals which would be bounded by similarly proportioned boxes such as a horse or a zebra. This assumption may have lead to poor training, although given the reasoning mentioned above, this seems unlikely.

4.2 Revised Investigation Question Results

In this section, results for the revised investigation question will be discussed. The VGG16 model trained significantly better than the original model. Comparisons between the vanilla and Bayesian augmented VGG16 show that the techniques proposed for the original investigative question are solid theoretically.

4.2.1 Model Training

The first tuning was done by selecting the optimizer for training the neural network. The following four optimizers were chosen for their ability to train both accurately and quickly. The optimizers used were Adam, NAdam, Adamax and Adagrad. As can be seen in Table 3, the Adamax optimizer performed best on both the standard model and the Bayesian model and therefore was utilized for both.

Table 3: Optimizer Testing

	Standard Val Accuracy	Bayesian Val Accuracy
Adam	98.65%	98.65%
NAdam	98.78%	98.02%
Adamax	**98.82**%	**98.82**%
Adagrad	96.79%	95.48%

4.2.2 Prediction Smoothing

After training the neural network, the next step in hyperparameter tuning was to tune the value for α, which, as mentioned in Chapter III, is the weight given to information from previous frames. As can be seen in Table 4, an α value of 0.75 performed the best of the tested values. The steady increase in accuracy when increasing α shows the importance of previous information when predicting on the current frame.

Table 4: Alpha Values

α value	Accuracy
0.00	75.06%
0.25	75.10%
0.50	75.15%
0.75	**75.17%**

After tuning alpha and predicting on all test videos, the affect that the proposed model had on flickering was tested. The amount of flickering is determined by counting

the number of times in a given video the prediction changes from one class to another. The proposed model was able to successfully decrease the amount of flickering by 67.43%, which is a large improvement. On average, the standard model will change classes every 1.85 seconds, whereas the Bayesian model will only change classes on average every 5.68 seconds.

Accuracy on both the standard model and the Bayesian model with smoothing can be seen in Figures 17 and 18. These figures show that both models performed very similarly on all classes. This shows that the Bayesian model can still compete with the accuracy of the standard model, while also giving a more robust look at how certain the model is on any given prediction.

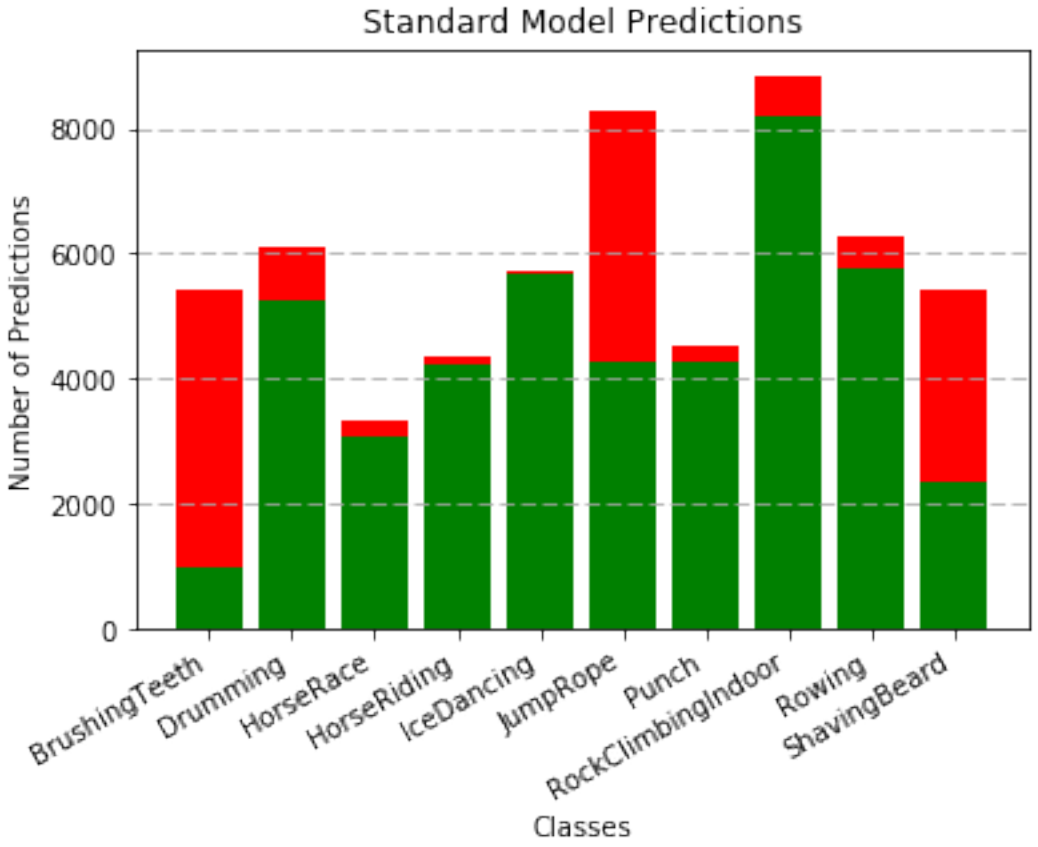

Figure 17: Predictions per Class for Standard Model

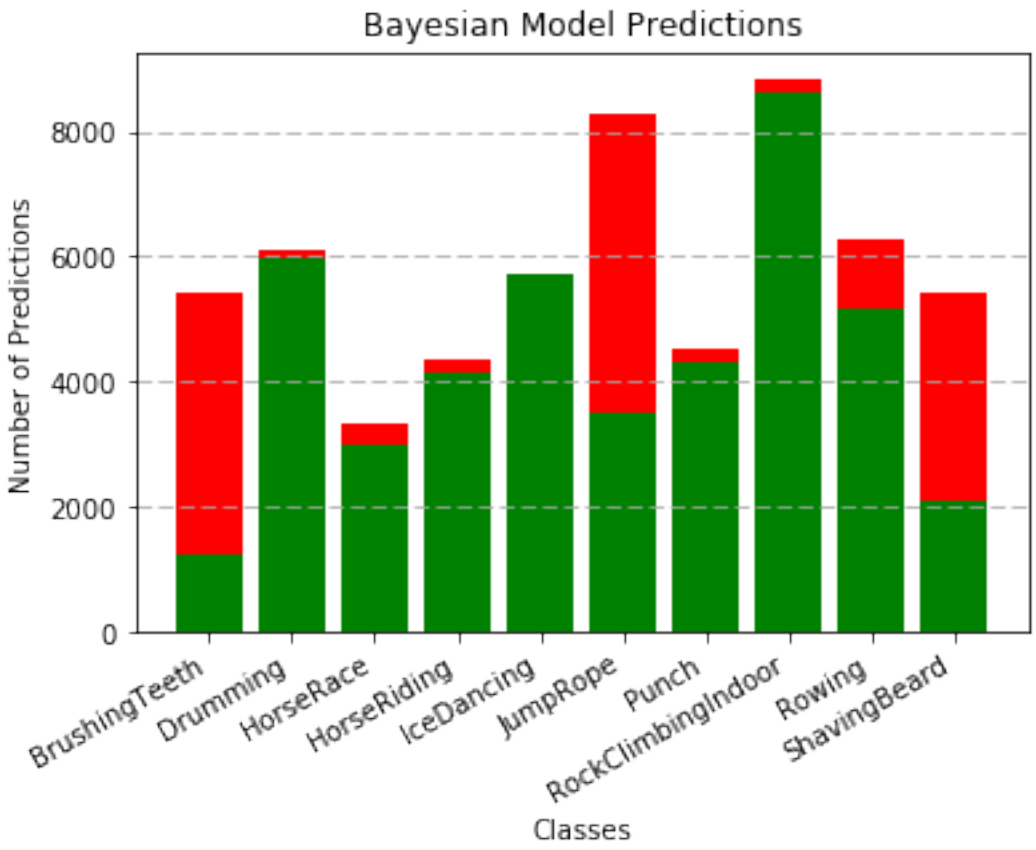

Figure 18: Predictions per Class for Bayesian Model with Smoothing

4.2.3 Adversarial Threshold

Because of this more accurate view of certainty, it becomes possible to test the model against adversarial classes as discussed in III. Higher adversarial threshold values make it harder for the adversarial class to be predicted, ensuring an increased level of confidence for the given prediction with each increase in β, if the given class is predicted. After selecting an adversarial class, in this case shaving, different adversarial threshold levels are tested. These tables are shown as confusion matrices with the brushing teeth class, as this was the pair of classes that the model struggled the most to accurately classify. The first, shown below in Table 5, shows the baseline where there is no threshold for the output to need to meet in order to predict the

adversarial class. In this instance, the False Positive rate is 76.60% and the True Positive rate is 43.91%.

Table 5: Adversarial Threshold of 0.0

		Truth	
		Brushing	Shaving
Pred	Brushing	12.29%	26.64%
	Shaving	40.21%	20.86%

By raising the adversarial class threshold to 0.6, as can be seen in Table 6 below, the False Positive rate is dropped to 75.70%, while the True Positive rate falls to 43.22%.

Table 6: Adversarial Threshold of 0.6

		Truth	
		Brushing	Shaving
Pred	Brushing	12.79%	26.89%
	Shaving	39.85%	20.47%

By raising the adversarial class threshold again to 0.7, as can be seen in Table 7, the False Positive rate is dropped to 74.54%, while the True Positive rate falls to 42.12%.

Table 7: Adversarial Threshold of 0.7

		Truth	
		Brushing	Shaving
Pred	Brushing	13.45%	27.30%
	Shaving	39.38%	19.87%

By raising the adversarial class threshold again to 0.8, as can be seen in Table 8, the False Positive rate is dropped to 73.09%, while the True Positive rate falls to 40.79%.

By raising the adversarial class threshold to the highest value tested here, 0.9, as can be seen in Table 9, the False Positive rate is dropped to 71.18%, while the True Positive rate falls to 39.10%.

Table 8: Adversarial Threshold of 0.8

		Truth	
		Brushing	Shaving
Pred	Brushing	14.27%	27.82%
	Shaving	38.76%	19.16%

Table 9: Adversarial Threshold of 0.9

		Truth	
		Brushing	Shaving
Pred	Brushing	15.33%	28.49%
	Shaving	37.88%	18.30%

By raising the adversarial class threshold level to 0.9 from 0, the False Positive rate drops by 5.42%. As with any improvement in modeling, however, it does come as a cost, with True Positives dropping by 4.80%. It is important for the end user of this model to understand the tradeoffs associated with implementing the adversarial back end. There are use cases like the ones mentioned in previous chapters in which this could be incredibly beneficial, however there are also use cases in which the trade off may not be worth it.

4.2.4 Effects of the Number of Predictions per Frame

Testing was also done to see the impact of the number of predictions per frame on the accuracy and speed of the model. The results of this can be seen in Figure 19. It can be seen here that when smoothing is utilized, the number of predictions per frame does not greatly impact the accuracy of the model, however decreasing from 20 predictions per frame to 1 prediction per frame results in the ability to predict on approximately 6 more frames per second. This is possible because the smoothing allows the model to still create an accurate distribution of predictions, even with only 1 prediction per frame. Context will be important to the end user when determining if a small decrease in accuracy is worth the additional prediction speed. While not

shown in Figure 19, the vanilla neural network with no smoothing predicted on 16.8 frames per second, meaning it is not significantly faster than utilizing the Bayesian model with smoothing.

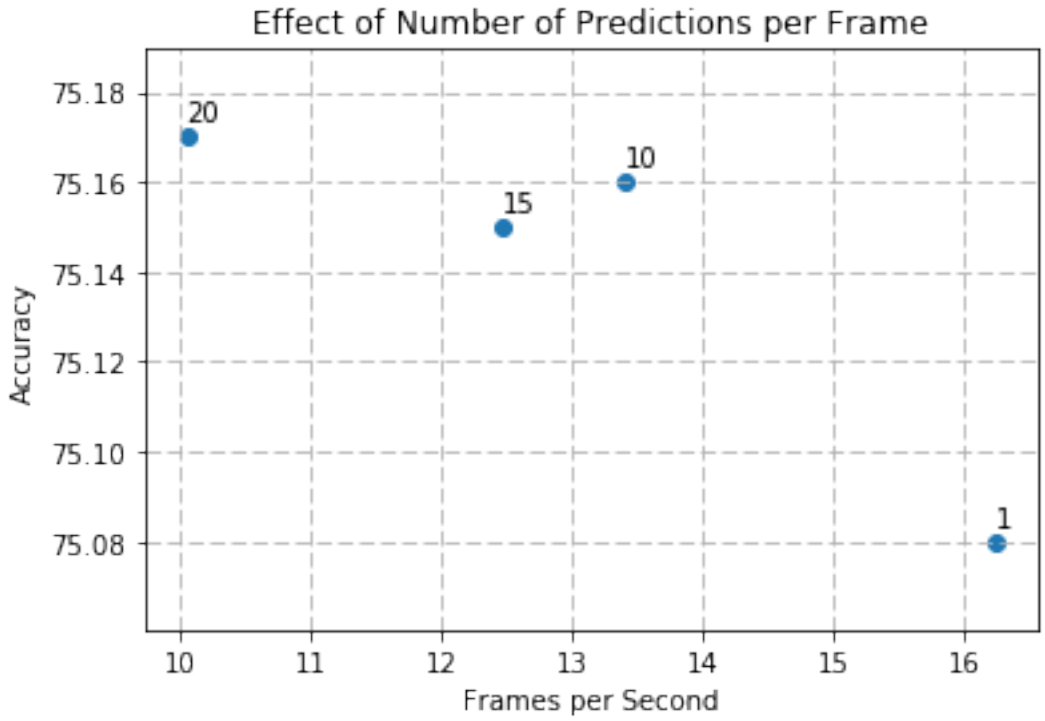

Figure 19: The Effect of Number of Predictions per Frame on Accuracy and Speed

Overall, the results shown in this chapter are extremely promising for the future of utilizing output distributions from Bayesian neural networks in tandem with smoothing functions instead of standard neural networks.

V. Conclusions

5.1 Summary

This thesis demonstrates that the passing forward of information from previous frames does lead to a significant improvement in the reduction of flickering. By reducing flickering by nearly 70%, the back end of the proposed model does exactly what it is intended to do and proves that prediction smoothing on the distribution outputs from Bayesian neural networks is effective. This thesis also successfully demonstrates that by using a Bayesian neural network, instead of a standard neural network, adversarial class thresholds can be successfully implemented due to the distributional outputs from the BNN.

5.2 Future Work

There a number of possible directions for future work to take. The first and most obvious of these is to continue the work to make the original proposed model work. While the training of the YOLOv3-tiny model was unsuccessful in this thesis, it has shown to be successful in the past and therefore may simply need to be trained for more epochs that computing power and time allowed on this thesis or it may take more finely tuned hyperparameters than explored in this work. The success of the secondary model show that the smoothing technique is effective and should be attempted on an object detection model.

The next direction future work could take on this thesis is to attempt to have an "anti-adversarial" class. That is, a class that should be predicted even if it is not the most voted class if it meets some certain lower threshold. This would be useful for use cases such as cancer detection, where False Negatives are extremely detrimental and the user would likely be willing to have a larger number of False Positives to get

a higher True Positive rate.

A final direction future work could take is to add more Bayesian layers in the BNN. The model did still see some over-fitting as would be expected for a standard neural network, meaning that only adding variation to the final layer may have not been enough to truly show the model's level of certainty as would be preferred.

Appendix: Glossary

ANN Artificial Neural Network. 4

BNN Bayesian Neural Networks. 1, 13, 17, 22, 46, 47

CNN Convolutional Neural Network. iv, 5, 6, 10, 16, 17, 1

CPU Central Processing Unit. 28

GPU Graphics Processing Unit. 16, 28, 38

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 6, 19, 39

IoU Intersection Over Union. 28

ISR Intelligence, Surveillance, and Reconnaissance. 1

KL Kullback-Liebler. 13

LSTM Long Short-Term Memory. 17

mAP Mean Accuracy Precision. 26, 28, 39

NMS Non Maximum Suppression. 9, 12

R-CNN Regions with Convolutional Neural Network. 8, 9

SVM Support Vector Machine. 9

tfp TensorFlow Probability. 21

VI Variational Inference. 13, 14

YOLO You Only Look Once. 9, 10, 19, 20, 21, 22, 24, 27, 39, 46

Bibliography

1. Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

2. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

3. Rohit Thakur. Step by step vgg16 implementation in keras for beginners, Nov 2020.

4. Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

5. Ayoosh Kathuria. What's new in yolo v3?, Apr 2018.

6. Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622. PMLR, 07–09 Jul 2015.

7. He, Chang-Wei Huang, Wei, Li, and Guo Anfu. Tf-yolo: An improved incremental network for real-time object detection. *Applied Sciences*, 9:3225, 08 2019.

8. Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694, 2020.

9. Kai Xu, Longyin Wen, Guorong Li, Honggang Qi, Liefeng Bo, and Qingming Huang. Learning self-supervised space-time cnn for fast video style transfer. *IEEE Transactions on Image Processing*, 30:2501–2512, 2021.

10. Rizwan Ali Shah, Odilbek Urmonov, and HyungWon Kim. Improving performance of cnn based vehicle detection and tracking by median algorithm. In *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–3, 2021.

11. Elizebeth Kurian, Jubilant J. Kizhakethottam, and Justin Mathew. Deep learning based surgical workflow recognition from laparoscopic videos. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 928–931, 2020.

12. Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

13. Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

14. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

15. Large scale visual recognition challenge 2017 (ilsvrc2017).

16. O. Russakovsky, J. Deng, H. Su, and et al. Imagenet large scale visual recognition challenge. *Int J Comput Vis*, April 2015.

17. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

18. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

19. Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

20. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

21. Tishby, Levin, and Solla. Consistent inference of probabilities in layered networks: predictions and generalizations. In *International 1989 Joint Conference on Neural Networks*, pages 403–409 vol.2, 1989.

22. John S. Denker and Yann LeCun. Transforming neural-net output levels to probability distributions. In *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, page 853–859. Morgan Kaufmann Publishers Inc., 1990.

23. Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, page 5–13. Association for Computing Machinery, 1993.

24. D. Barber and Christopher Bishop. Ensemble learning in bayesian neural networks. In *Generalization in Neural Networks and Machine Learning*, pages 215–237, January 1998.

25. Alex Graves. Practical variational inference for neural networks. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, page 2348–2356. Curran Associates Inc., 2011.

26. Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *ArXiv*, abs/1506.02158, 2015.

27. Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. *Lecture Notes in Mathematics*, page 45–87, 2020.

28. Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger B. Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *ArXiv*, abs/1803.04386, 2018.

29. Anastasios Stamoulakatos, Christos Tachtatzis, Javier Cardona, Xavier Bellekens, Robert Atkinson, I. Andonovic, Md Hossain, Craig Michie, and Pavlos Lazaridis. A comparison of the performance of 2d and 3d convolutional neural networks for subsea survey video classification. 02 2022.

30. Parag Pendharkar. A threshold-varying artificial neural network approach for classification and its application to bankruptcy prediction problem. *Computers Operations Research*, 32:2561–2582, 10 2005.

31. Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):63–77, 2006.

32. Joseph Redmon. Yolo: Real-time object detection. Available at `https://pjreddie.com/darknet/yolo/`.

33. Pavlo Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20:20–24, 12 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Bayesian Convolutional Neural Network with Prediction Smoothing and Adversarial Class Thresholds

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Miller, Noah, 2nd Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

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

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

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-22-M-154

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

Trevor Bihl, DAF, DR-III, PhD
Sensors Directorate
Air Force Research Laboratory
2242 Avionics Circle
Wright-Patterson AFB, OH 45431 Email: trevor.bihl.2@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL

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

12. DISTRIBUTION / AVAILABILITY STATEMENT

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

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Using convolutional neural networks (CNNs) for image classification for each frame in a video is a very common technique. Unfortunately, CNNs are very brittle and have a tendency to be over confident in their predictions. This can lead to what we will refer to as "flickering," which is when the predictions between frames jump back and forth between classes. In this paper, new methods are proposed to combat these shortcomings. This paper utilizes a Bayesian CNN which allows for a distribution of outputs on each data point instead of just a point estimate. These distributions are then smoothed over multiple frames to generate a final distribution and classification which reduces flickering. Our technique is able to reduce flickering by 67%. We also propose a second method to combat False Positive predictions of certain adversarial classes, or classes that have some cost if predicted incorrectly. This is accomplished by increasing the confidence threshold the adversarial class must meet in order to be the final predicted class. This technique is able to reduce false positives by 5.43%, while maintaining accuracy.

15. SUBJECT TERMS

artificial neural network (ANN), convolutional neural network (CNN), Bayesian neural networks (BNN), image classification, object detection, prediction smoothing, adversarial class thresholds

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Cox, AFIT/ENS
U	U	U	UU	64	19b. TELEPHONE NUMBER *(include area code)* (937) 255 3636 x4676; Bruce.Cox@afit.edu