Bayesian Augmentation of Deep Learning to Improve Video Classification

Emmie Swize
Air Force Institute of Technology, USA
Emmie.Swize@afit.edu

Lance Champagne
Air Force Institute of Technology, USA
Lance.Champagne@afit.edu

Bruce Cox
Air Force Institute of Technology, USA
Bruce.Cox@afit.edu

Trevor J. Bihl
Air Force Research Laboratory, USA
Trevor.Bihl.2@us.af.mil

Abstract

Traditional automated video classification methods lack measures of uncertainty, meaning the network is unable to identify those cases in which its predictions are made with significant uncertainty. This leads to misclassification, as the traditional network classifies each observation with same amount of certainty, no matter what the observation is. Bayesian neural networks are a remedy to this issue by leveraging Bayesian inference to construct uncertainty measures for each prediction. Because exact Bayesian inference is typically intractable due to the large number of parameters in a neural network, Bayesian inference is approximated by utilizing dropout in a convolutional neural network. This research compared a traditional video classification neural network to its Bayesian equivalent based on performance and capabilities. The Bayesian network achieves higher accuracy than a comparable non-Bayesian video network and it further provides uncertainty measures for each classification.

1. Introduction

Video classification involves detecting and identifying objects and activities autonomously. This builds upon image classification, which is commonly performed by neural networks. The combination of convolutional neural networks (CNN), often used for image classification, and recurrent neural networks (RNN), which handle time series data, allow for the classification of videos whereby videos broken into a series of images for analysis.

However, the current technology is not flexible enough in nature to handle novelty data. For example, currently available classification algorithms are largely static in nature once trained and context from recently observed data is not considered in making decisions, meaning such algorithms are unequipped to handle uncertain and unexpected situations. The remedy for this is to create classification algorithms that are aware of their own uncertainty and therefore able to identify unexpected data [1].

As mentioned in [2], there are in general 5 "tribes" of machine learning algorithms: symbolists (e.g. decision trees), Bayesians (e.g. Naïve Bayes), Connectionists (e.g. Neural Networks), Evolutionaries (e.g. Genetic Programs), and Analogizers (e.g. Support Vector Machines). While great advances have been made from these approaches, it is hypothesized that much more is capable by combinations of approaches [2]; however, this is difficult due to these "tribes" having very different philosophies and terminologies. The general video and image based recognition systems fall into the connectionists tribe; however, they are inefficient in that they do not remember prior results and often treat each frame in an image as an independent observation.

This research aims to explore blending of the connectionist and Bayesian tribes to classify videos and images with the goal of allowing the classification model to measure its uncertainty in each prediction. The approach taken herein consists of a CNN for image classification, an RNN for sequences of images (video) classification, with a Bayesian neural network (BNN) incorporated to measure uncertainty. Comparisons are made against the baseline, non-Bayesian equivalent, algorithm. Results show a significant improvement in classification accuracy when using the hybrid approach.

This paper is organized as follows: a background on artificial neural networks (ANN) is presented, uncertainty is discussed, and the video dataset under analysis is presented. Then the paper presents the implemented algorithm. Finally, results and conclusions end the paper.

2. Background

ANNs are machine learning models loosely based on the structure of the brain's biological network, in which biological neurons pass information through connections when triggered [3]. ANNs are versatile

URI: https://hdl.handle.net/10125/79596
978-0-9981331-5-7

machine learning tools that can handle large and complex tasks, including image recognition. Furthermore, constructed appropriately, ANNs are a provably optimal approach to learning patterns in data [3].

One basic unit of ANNs are threshold logic units (TLU) [4], which are neurons activated by the inputs passed to them where numerical value are both input and produced as an output. As illustrated in Figure 1, the connections around TLUs are weighted, meaning each unit computes the weighted sum of the inputs passed to it according to the connection weights.

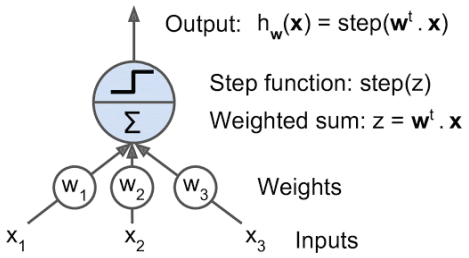

Figure 1. Threshold Logic Unit [4]

The output of each TLU is the result of a step function applied to the weighted sum of the inputs. To provide inferential capabilities, the Perceptron was developed as single layer of TLUs [5]; Figure 2 demonstrates the structure of a Perceptron with two inputs and three TLUs. While the output of the TLU presented in Figure 2 is the result of step functions, modern ANNs build off the work of [6] and use a weight-training method called backpropagation which employs gradient descent and differentiable activation functions, such as the sigmoid activation function.

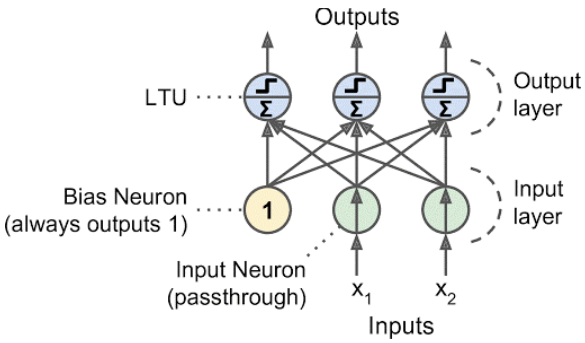

Figure 2. The Perceptron [4]

The capabilities of such a model improve with breadth and depth, whereby multiple TLUs and their interconnection enable increasing powers of inference. Multilayer Perceptron (MLP) are the result which are ANNs composed of multiple layers of TLUs [4]. In an MLP, the middle layers are referred to as the hidden layers and the last layer is referred to as the output layer. A Deep Neural Network (DNN) builds on the structure of the MLP, consisting of a deep stack of hidden layers, typically three or more.

2.1. Convolutional Neural Networks

CNNs are a variant of DNNs that are particularly useful in processing and categorizing 2-dimensional visual data, such as images and handwriting [4]. The earliest stage of the CNN, the Neocognitron, was proposed by [7], and it contained simple cell operations for the feature extraction of an image and complex cell operations that pool the simple cell results to provide spatial invariance. This CNN model inspired further development, particularly of the LeNet-5 model in 1998 [8], which introduced convolutional layers and pooling layers, the backbones of the modern-day CNN.

In a general CNN, convolutional layers apply a learned filter, called the kernel, to the input arrays to detect co-occurrences and spatial information in the input [4]. In doing so, small ``neighborhoods'' of the image are examined, revealing each neuron's area of influence based on its location. This lends well to image and video classification, in which pixels closer together are typically more correlated than pixels farther apart. Pooling layers create a summarized version of the features identified by the preceding convolutional layer, reducing the dimensionality. With convolutional and pooling layers, CNNs assemble simple features into increasingly more complex features with each hidden layer [4].

Table 1. LeNet-5 Architecture of [8]

Layer	Type	Description	Kern. Size
in	Input	1 map of size 32x32	
1	Convolutional	6 maps of 28x28 neurons	5x5
2	Avg pooling	6 maps of 14x14 neurons	2x2
3	Convolutional	16 maps of 10x10 neurons	5x5
4	Avg pooling	16 maps of 5x5 neurons	2x2
5	Convolutional	120 maps of 1x1 neurons	5x5
6	Fully connect.	84 neurons	
Out	Fully connect.	10 softmax neurons	

Table 1 demonstrates the layout of the LeNet-5 architecture. Each convolutional layer outputs one feature map for each filter, each of which emphasizes the image locations that activate the respective filter the most. By applying multiple filters to the inputs, a convolutional layer is able to extract multiple features at each location. The sub-sampling layers represent the pooling layers, which reduce the sensitivity of the outputs to shifts and distortions in the image. This gives the CNN the powerful capability of recognizing a learned pattern in any location in the image, not just

where the original pattern instance occurred [8]. As a final classifier, the LeNet-5 architecture includes an MLP at the end consisting of fully connected layers and an output layer.

2.2. Recurrent Neural Networks

While CNNs are useful at processing and categorizing individual images, they are not good at processing sequential data, such as a video, which includes a temporal component. RNNs possess a type of memory in the form of a hidden state, which passes previous output information as additional inputs to future time steps in the network [4]. This extends from the associative memory concepts instantiated in Hopfield Networks [9], and build upon the Backpropagation Through Time (BPTT) methods developed by Rumelhart [9]. Backpropagation is a procedure that repeatedly updates a network's connection weights according to the error of the network's output. BPTT is the application of backpropagation to each time step of an RNN or RNN variant [9]. Figure 3 demonstrates a layer of recurrent neurons unrolled through time. At each step, the layer receives the input x_i and the output of the previous time step y_{t-1} as inputs.

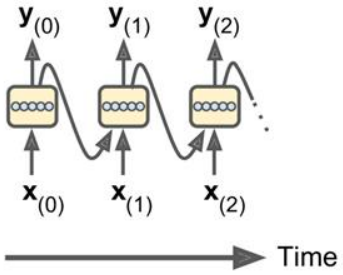

Figure 3. RNN Layer Unrolled through Time [9]

Standard RNNs, although able to process sequential data, can suffer from short-term memory, meaning the network's memory is limited in the number of past outputs it can represent clearly. An evolved version of the standard RNN, the Long Short-Term Memory (LSTM) cell, was developed as a solution [10]. LSTM cells outperform standard RNNs by converging more quickly and by detecting long-term dependencies in sequential data. Figure 4 shows that LSTMs possess not only a hidden state, but a cell state as well. The hidden state is similar to that of the standard RNN and represents the short-term information, while the cell state represents the long-term information. LSTM cells also utilize three gate controllers that are responsible for adding information to the stored memory or erasing information from the stored memory [4]. These controllers are the forget gate, the input gate, and the output gate. With the use of these gates, the LSTM cell is able to discern which content should be stored in its memory and which content it should forget.

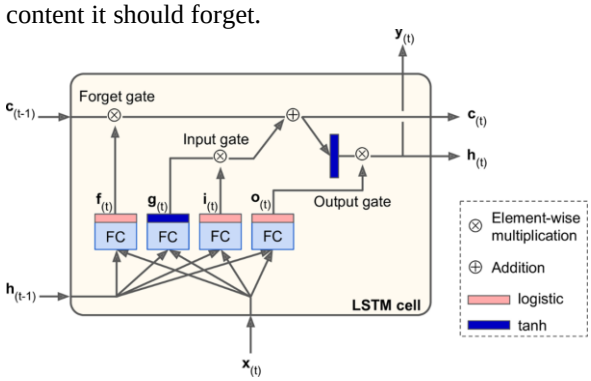

Figure 4. LSTM Cell [4]

2.3. Bayesian Neural Networks

Standard ANNs and DNNs learn point estimates for network weights and produce point estimates for predictions. These networks do not have a measure of certainty or confidence in the parameters and predictions, making the results potentially difficult to trust. In other words, standard DNNs output point estimate predictions, but no measure of respective uncertainty. Using Bayesian inference, a BNN incorporates a measure of uncertainty by learning parameters as distributions instead of point estimates [9] [11]. Bayesian inference is a type of statistical inference that uses Bayes' theorem to update the inferred weight distributions as more information becomes available to the network. Any kind of network can become a BNN by treating the parameters in a Bayesian manner [9]. Figure 5 presents a visual of the weight-learning difference between a standard CNN (left) and BNN (right).

Neal [12] expanded his contribution to the growth of BNNs the following year by establishing a link between BNNs and Gaussian processes, which are stochastic processes in which every finite linear combination of random variables is normally distributed. Although the Gaussian process properties do not translate easily to finite neural networks, Bayesian inference can be approximated for finite neural networks that have Gaussian priors for the weights. As pointed out by Shridhar et al. [13], even for a network with few parameters, performing exact Bayesian inference to determine a network's posterior is a lengthy and difficult task. For this reason, Bayesian inference is often approximated using variational inference, a method that fits a Gaussian distribution as closely as possible to the true posterior

distribution [9]. This is done by minimizing the Kullback-Leibler (KL) divergence, a measure of how much information is lost in the approximation. However, because variational inference significantly increases the number of model parameters, it comes at a high computational cost.

A typical dropout procedure is only implemented during the training stage, causing predictions during the testing stage to be deterministic. Monte Carlo dropout (MCD), proposed by [14], applies dropout to both the training and testing stages of a network. Implementing dropout during the testing stage means

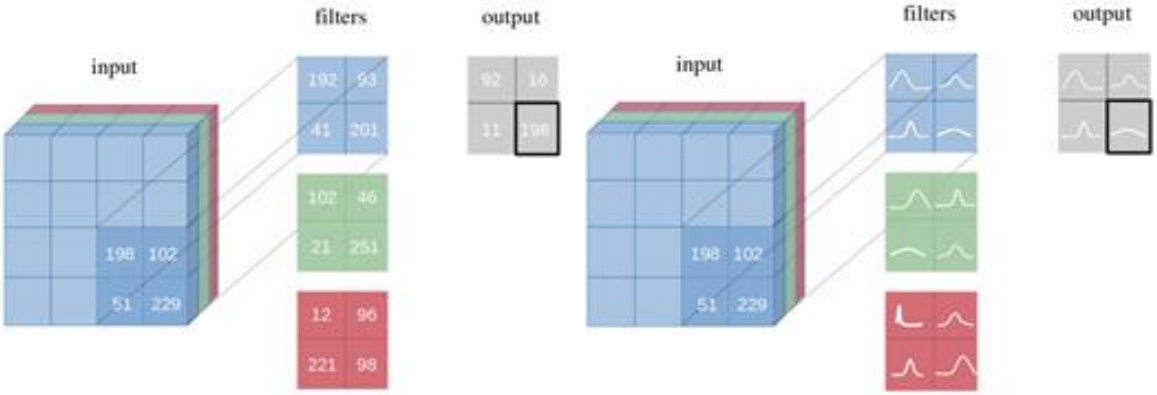

Figure 5. Example of a BNN with an input image with exemplary pixel values, filters, and corresponding output with point estimates (left) and probability distributions (right) over weights [13]

2.4. Dropout as a Bayesian Approximation

A simple method of Bayesian inference approximation that does not sacrifice computational complexity is dropout. In 2016, Gal and Ghahramani [14] demonstrated that applying dropout before every weighted layer in a network is mathematically equivalent to a Bayesian approximation of a Gaussian process. Their work shows that the application of dropout minimizes the KL divergence between an approximate distribution and a Gaussian process posterior distribution.

Dropout, a popular regularization technique that prevents a model from overfitting training data, is a process that randomly omits neurons and their associated connections from the neural network according to a fixed probability *p*. In other words, at each step, each neuron will be retained in the network at that step with probability *p*. Figure 6, created by Srivastava et al., demonstrates the difference between standard neural network layers and neural network layers with dropout. This technique essentially samples a thinned version of the full network for each training case. For a neural network with *n* neurons, applying dropout to training amounts to a collection of 2^n possible networks [15]. Additionally, dropout prevents individual neurons from relying on other specific neurons to supplement their contributions to the network [16]. This causes the contribution of each neuron to become more helpful regarding correct network predictions.

that the model output can be treated as a random sample generated from the posterior predictive distribution. The model uncertainty can therefore be estimated with the distribution of repeated predictions for an instance, constructing a distribution of probabilities for every class. With this distribution of multiple predictions, the average and the variation can reveal the networks uncertainty in its predictions. Gal and Ghahramani [14] show that not only is this procedure simple in execution, but that it has no negative impact on model performance.

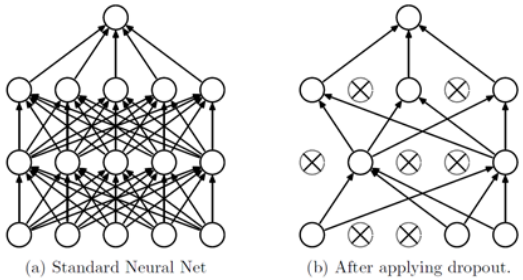

Figure 6. Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [15]

2.5. Rejecting Uncertain Classifications

A known limitation of machine learning classifiers is that possessing a fixed set of classification

categories they attempt to emplace all test data into these categories. When the test data either does not come from the closed training domain (e.g., a CNN trained on MNIST which is then fed test data from the not-MNIST data set), is sufficiently noisy to fall outside the classification boundaries, or the resulting confusion matrix is unbalanced (indicating some categories were poorly trained or have fuzzier boundaries). Since at least 1969 researchers (see e.g., [18], [19], [20], [21],) have proposed various techniques for either enabling classifiers to refuse to classify an image that falls outside the closed domain of the training set, or to explicitly place uncertain test data into an "I don't know" catchall category.

Broadly these techniques fall into the following categories. 'First, the test data may be eliminated via preprocessing. Second, after the classifier is trained the output for the test data may be examined and discarded as appropriate. Third, the researcher can build the ability to classify a test point as unknown into the neural network' [22]. Our technique is a hybrid of the first and third approaches. Specifically, we utilize a novel hybrid Bayesian Augmented CNN-RNN. Similar to Chows seminal work [20] we establish rejection rules, such that a trained Bayesian Neural Network acts as a preprocessor and eliminates test data that meets any of the rejection rules. The remaining "certain" test data is then fed into the RNN and is classified. In this fashion we create a classifier which exhibits high accuracy for the test data it is certain of and notifies a human operator when it is uncertain of a video's classification. Such a risk adverse approach is particularly appropriate when high negative consequences are associated with false positives.

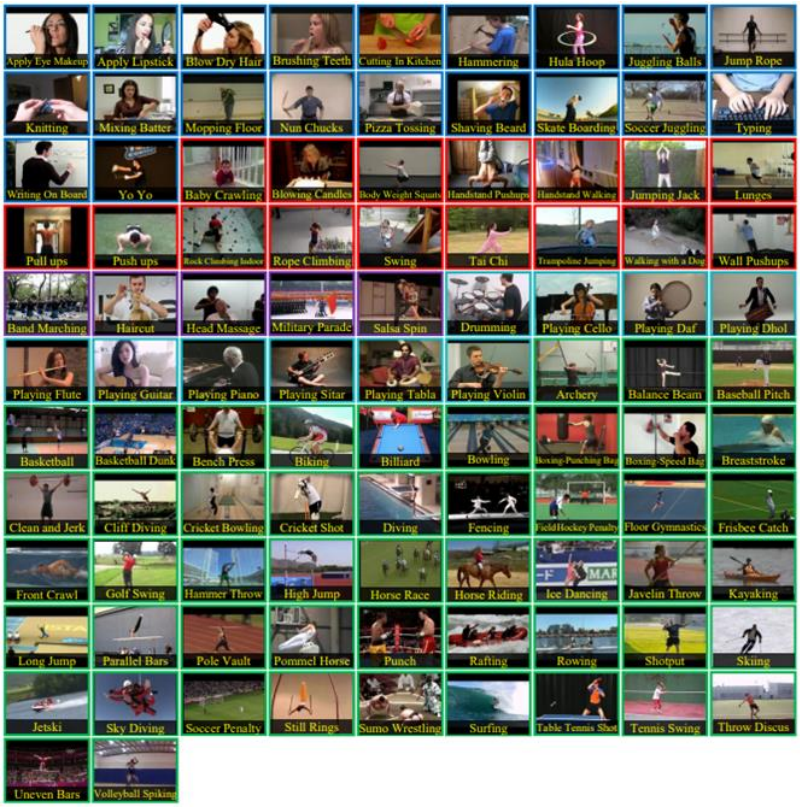

Figure 7. Sample frames of each of the 101 action classes in UCF101. The color of each frame border corresponds to the respective action type: blue for Human-Object Interaction, red for Body-Motion Only, violet for Human-Human Interaction, cyan for Playing Musical Instruments, and green for Sports [17].

2.6. Representative Video Data

The data set used in this research is the UCF101 Action Recognition Data Set [17]. The University of Central Florida (UCF) introduced this dataset in 2012 as the largest to date compilation of human action videos, comprising of 13,320 videos that span 101 classes of actions in total. UCF101 offers a large amount of variety regarding actions, camera motion and viewpoint, object pose and scale, background appearance, and lighting conditions. The 101 action classes are divided into five types: Human-Object

Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments, and Sports.

Table 2 provides the summary statistics of the data set structure and Figure 7 provides visual representations of the data set. Figure 7 contains frames of an example video from each of the 101 action classes along with corresponding class and class type labels.

Each class of actions is also subsequently divided into 25 groups based on common features, such as background or viewpoint. For all classes, the 25 groups contain anywhere from 4 to 7 videos each. Although the videos of roughly half of the action categories also contain audio, this feature is not taken into consideration in this research. As preparation for training and testing, each video is partitioned into individual frames sized 224 x 224 pixels, with each pixel containing values for 3 color channels. The images are converted to three-dimensional arrays, sized 224 x 224 x 3. These arrays are normalized by dividing all values by 255 to have all pixel values range from 0 to 1.

Table 2. UCF101 Summary Statistics [17]

Statistic	Values
Actions	101
Clips	13,320
Groups per action	25
Clips per Group	4-7
Mean clip length	7.21
Total duration	1600 mins
Min Clip Length	1.06
Max Clip Length	71.04
Frame Rate	25 fps
Resolution	320 x 240
Audio	Yes (51 actions)

3. Bayesian Augmented CNN-LSTM

This research explores the effects of utilizing a blend of the best of breed of the networks discussed above to gain synergistic effects. The CNN is responsible for classifying individual frames of a video, which is then passed to the RNN as a sequence of frames for classification of the video as a whole. However, this doesn't involve any memory of what was previously seen. Thus, the main contribution is in treating the CNN parameters in a Bayesian manner, and making it a BCNN. The front-end network is a BCNN and the back-end network is an RNN. The goal of this network architecture is to classify videos and provide a measure of uncertainty in each video's frame predictions.

3.1. Front-end BCNN

The front-end BCNN is constructed from a simple CNN consisting of two weighted layers with dropout applied before every weighted layer. This is following Gal's finding that applying dropout before every weighted layer is an approximate Bayesian inference. Both dropout layers utilize an identical dropout rate of 0.5 according to the precedent set by [14]. This reduction in the front-end model's size does not impact this research, as its aim is to show a proof of concept.

Table 3. Front-end BCNN

Layer	Type	Description	Kern. Size
in	Input	1 image: 224x224x3	
1	Convolutional	32 maps of 222x222 ReLu neurons	3x3
2	Convolutional	32 maps of 220x220 ReLu neurons	3x3
3	Max pooling	32 maps of 73x73 ReLu neurons	3x3
4	Flatten	32 ReLu neurons	
5	Dropout	170528 parameters	
6	Fully connect.	170528 neurons	
7	Dropout	50 parameters	
8	Fully connect.	50 ReLu neurons	
Out	Fully connect.	101 neurons: classification	

Table 3 presents the architecture of the front-end model. The input consists of three colored images (red, green, blue) which then passes through two convolutional layers, stride of 1, zero-padding of 1. The kernel size used in the convolutional layers is the smallest kernel size that can capture the concept of left and right, up and down, and center. This is followed by a max-pooling layer and two fully connected layers, one with 50 units and one with 101 units, which is the number of classes in the UCF101 dataset. The activation function of all the layers, aside from the final fully connected layer, is the rectified linear unit activation function (ReLU). The final fully connected layer uses a softmax activation function, which converts the input to a vector of categorical probabilities, each between 0 and 1, that sum to 1. However, as shown by [14], the output vector of probabilities from the softmax function alone cannot be interpreted as model confidence or uncertainty.

The output of the BCNN is a list of matrices, one for each video that is classified by the front-end network. Initially, each matrix contains as many rows as a video has frames and as many columns as the data set has classes.

3.2. Back-end RNN

Given a video index, the matrix corresponding to that index in the list contains the MCD predicted

probabilities for each of the 101 classes for each frame of the video. However, the Keras LSTM layers used to build the back-end network require that all sequences within each batch have the same number of timesteps. To standardize the number of frames across the data passed to the back-end network, all matrices in the list outputted by the front-end network are padded with arrays of zeros to match the highest number of frames that occurs in the data. Using zero padding allows for the list to preserve the original content of the data.

This list of matrices, the BCNN output, is fed into an RNN consisting of two LSTM layers, each with 50 units and a softmax activation function. These two LSTM layers are followed by two fully connected layers that contain 50 units and 101 units, respectively. The architecture of the back-end network is represented in Table 4.

Table 4. Back-end RNN

Layer	Type	Description
in	Input	1 vector of 101x1
1	LSTM	50 units, softmax
2	LSTM	50 units, softmax
3	Fully connect.	50 softmax neurons
Out	Fully connect.	101 softmax neurons

3.3. Hyperparameter Tuning

Because the performance and required training time of a model depend on the specified hyperparameter values [23]; thus tuning the hyperparameters to find optimal or near-optimal values is of interest. The approach used herein was cross validation to exhaustively consider all parameter combinations from a grid search to determine the optimal parameter values for a model, with a value of 3 for K-fold cross validation [4].

For the models developed herein, the tunable hyperparameters included the batch size, epochs, optimizer, weight initialization method, and size of hidden layers. The batch size is the number of data observations that are shown to the network before updating the weights [4]. RNNs and CNNs are particularly sensitive to the batch size. The number of epochs is the number of times that the entire training data set is shown to the network during training [4]. The optimizer is the algorithm used to update the model weights in response to the loss function results [4]. The weight initialization method determines with which random distribution, if any, the network weights are initialized, which heavily affects network training time [4] }. The size of a hidden layer refers to the number of neurons it contains, which controls the representational capacity of the network at that layer [4].

4. Comparative Assessment and Evaluation

The performance and capabilities of the above Bayesian model will be compared to a non-Bayesian, baseline model consisting of the same front-end CNN and back-end RNN structure, but with the dropout layers removed from the front-end network during the testing phase. Thus both the Bayesian front-end network and the baseline front-end network are identical other than the employment of dropout during the testing phase for the Bayesian front-end network. The same back-end RNN is used after each of these front-end networks.

First, we compared the baseline and Bayesian models on their performance in classifying the entire test set without the ability to leave images non-classified. Next, both models are evaluated on the measures of uncertainty that they can each provide for their predictions. This includes a sensitivity analysis on the non-classified thresholds for each model. Finally, both models are evaluated on their performance on the subsection of the test set they choose to classify based on the uncertainty thresholds. The test set used for evaluation spans all 101 classes and contains 3,782 videos, which collectively contain 28,890 images.

All model training and evaluation were conducted on a Windows 10 Professional PC with an AMD Ryzen 5 5600X CPU, 32 GB RAM, and Sapphire Nitro+ RX 5700XT, as well as the Python 3.7 packages Tensorflow 2.1.0, Keras 2.3.1, and all necessary dependencies.

4.1. Network Performance without Uncertainty

Without accounting for uncertainty, both the baseline and Bayesian neural network configurations classified the video data set. The baseline front-end model achieves 25.2% accuracy on the whole test set. Feeding the baseline front-end network's predictions to the back-end network, the combined model achieves 1.3% accuracy. The Bayesian front-end network achieves 20.9% accuracy on the whole test set. With the Bayesian front-end network predictions as input for the back-end network, the Bayesian RNN combined model achieves 1.3% accuracy.

4.2. Uncertainty Thresholds

The appeal of measuring the network's uncertainty lies in the network's ability to know what it does not know. However, this ability is not useful unless the

network is also able to request clarification for those cases about which it is uncertain. To address this, both front-end networks, baseline and Bayesian, will 'flag' any image and its associated video that corresponds to high uncertainty based on respective network thresholds.

For the Bayesian front-end network, the distribution of MCD predictions for an image is used to determine whether the network will classify the image or leave it non-classified. There are many ways to set the network uncertainty threshold for flagging an image, some of which will suit certain data and situations more than others. For the purposes of this research, the uncertainty thresholds are set as follows:

1. An image and its video are flagged if the maximum mean predicted class probability is less than a determined cutoff value.
2. An image and its video are flagged if there are two or more mean predicted class probabilities greater than another determined value.
3. An image and its video are flagged if the maximum standard deviation for any of the predicted class probabilities is greater than a determined value.

These values and their respective sensitivity analyses are specified in the following section. These flag

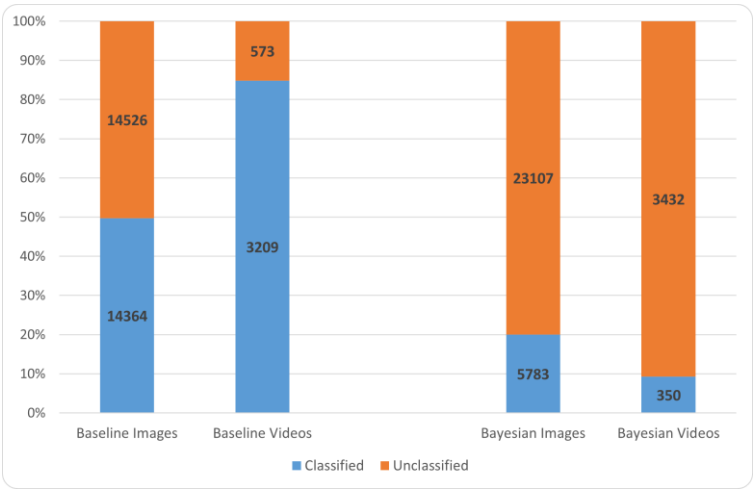

Figure 8. Classified to non-classified ratio for each front-end network.

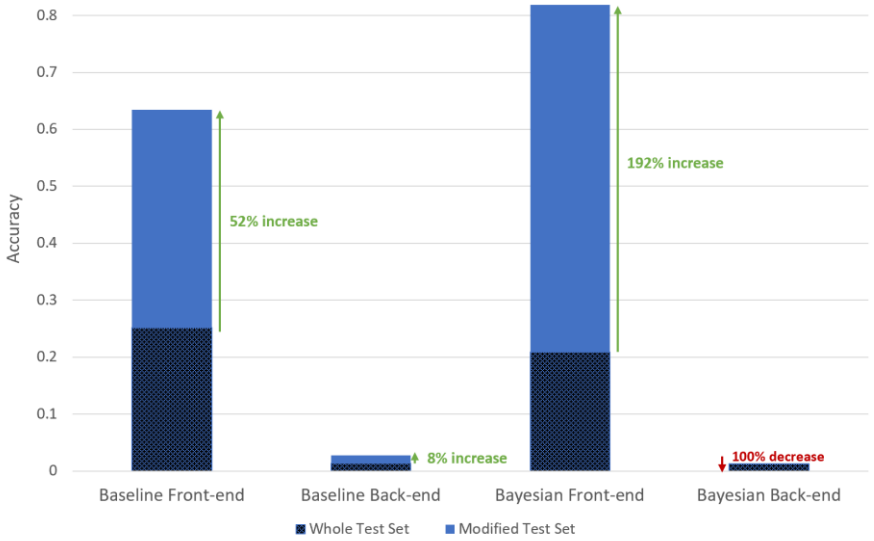

Figure 9. Model performances on whole test set and on modified test set. Here the entire bar represents the accuracy of a model on the modified test set, while the shaded portion of the bar represents the accuracy of a model on the whole test set.

thresholds are chosen to ensure reasonable network certainty in the unflagged images.

The baseline front-end network cannot provide the same distribution of predictions that the Bayesian front-end network can. For this reason, in order to gain a semblance of the baseline front-end network's uncertainty, this research applies a frequentist methodology, which treats uncertainty as a probability that is the limit of the relative frequency of an event after many trials [24]. To accomplish this, augmented data is created from the images and used to construct a distribution of correct predictions and incorrect predictions for each class [25]. The data is augmented using the Image Data Generator class in Keras [26]. The baseline `non-classified' threshold for each class is found using the same determined value as mentioned in the first Bayesian threshold as a percentile cutoff in the correct predictions distribution for that class. During the testing stage, if the softmax output for the predicted class is below the threshold of that class, then the image is rendered 'non-classified.'

4.3. Determining Uncertainty Thresholds

Experimental runs were conducted to determine the sensitivity of the three threshold uncertainty values. In order for the Bayesian front-end network to classify at least half of the test set videos (14,445), the standard deviation threshold (criterion 3) must be greater than 0.32; below this value, the other two threshold values do not allow the network to classify even half of the test set. In fact, even with a standard deviation threshold of 0.32, the Bayesian network only classifies half of the test set with a cutoff value of 0.10 for criterion 1 and a value of 0.5 for criterion 2. With these values, the Bayesian network classifies all images that have maximum mean predicted class probability greater than 0.10, no more than one class with a mean predicted probability greater than 0.50 (which could not occur regardless), and with no standard deviations greater than 0.32 for the predicted class probabilities.

For this research, we chose the final thresholds for all three Bayesian uncertainty thresholds and for the baseline uncertainty threshold to enable the Bayesian front-end network to classify at least 20% of the whole test set (min 5,778 videos). This will enable a more thorough analysis of the network's capabilities. To accomplish this, the criterion 1 threshold value is 0.6, the value for Bayesian uncertainty (criterion 2) is 0.25, and the Bayesian standard deviation threshold (criterion 3) value is 0.4. For the baseline network, this means that for an image to be classified, the softmax output for the predicted class must be at least the value of the 60th percentile of that class's correctly predicted augmented data softmax probabilities. For the Bayesian network, then the mean predicted class probability must be at least 0.6, no more than one mean class probability can be over 0.25, and no standard deviation of the class probabilities can exceed 0.4.

4.4. Network Results Including Non-Classification Due to Uncertainty

Using the threshold settings outlined in section 4.2, the baseline and Bayesian networks again attempted to classify the test data. Under these settings, each network configuration either categorized the image/video or flagged it as "non-classified." Figure 8 shows the ratio of classified to non-classified for both images and videos for the networks with the chosen threshold values. The baseline network classifies significantly more images and videos than the Bayesian network, but at a significant cost in accuracy.

On the modified test set, the baseline front-end network achieves 38.2% accuracy. The Bayesian front-end network achieves 61.0% accuracy. Figure 9 provides a visual of the difference between whole test set performances and modified test set performances. As shown in Figure 9, the baseline network with the CNN front-end and RNN back-end experienced a 52% and 8% increase in accuracy, respectively. The Bayesian front-end experienced a 192% increase in accuracy, while its RNN back-end had a 100% decrease in accuracy. It should be noted that both the increase and decrease in RNN accuracy represents a large percentage change that is due to the small denominator. That is, the RNN performed poorly in all cases. Improving the RNN performance through architecture or data structure changes is left to future study.

5. Conclusions

The Bayesian model construct for video classification provided significant accuracy improvements in video classification over a baseline convolutional neural network with similar criterion for flagging videos as "unclassified." With estimation for Bayesian priors calculated through network node dropout, three criteria were developed to flag certain videos as "non-classified."

The Bayesian CNN's success was shown for this particular data set over a traditional data set. With careful threshold selection, the Bayesian network was able to leave troublesome images/videos non-classified, rather than forcing (or allowing) an incorrect classification.

However, the results also show significant challenges that must be overcome. First, as implemented in this research, the subsequent recurrent neural network did not improve the accuracy of the video classification, and in some cases showed worse accuracy. Second, the uncertainty can leave a large portion of the data set non-classified. This may be acceptable in cases where the cost of incorrect classification is catastrophic, but the cost of human intervention may still be quite significant.

Finally, this research demonstrated success in the comparison of a baseline CNN to a Bayesian CNN in an AR context and, therefore, has only been tested in a limited sense. To broadly compare these algorithms, it is necessary to both find applicable data and comparison metrics.

6. Bibliography

[1] T. Bihl and M. Talbert, "Analytics for autonomous c4isr within e-government: a research agenda.," *Hawaii International Conference on System Sciences*, pp. 2218-2227, 2020.

[2] P. Domingos, *The Master Algorithm*, Basic Books, 2015.

[3] T. Bihl, et al., "Artificial neural networks and their applications in business," *Encyclopedia of Information Science and Technology*, 4th Edition, pp. 6642-6657, 2018.

[4] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2019.

[5] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton*, Cornell, 1957.

[6] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavior science*., Unpublished Doctoral Dissertation: Harvard University., 1974.

[7] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," *Competition and cooperation in neural nets*, pp. 267-285, 1982.

[8] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[9] Y. Gal, *Uncertainty in deep learning*, University of Cambridge, 2016.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[11] N. Park,et al., "Vector Quantized Bayesian Neural Network Inference for Data Streams," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 9322-9330, 2021.

[12] R. Neal, *Bayesian learning for neural networks*, vol. 118, Springer Science & Business Media., 2012.

[13] K. Shridhar, et al., "A comprehensive guide to bayesian convolutional neural network with variational inference.," arXiv preprint arXiv:1901.02731., 2019.

[14] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *International Conference on Machine Learning*, pp. 1050-1059, 2016.

[15] N. Srivastava, et al., "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[16] G. Hinton, et al., "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580., 2012.

[17] K. Soomro, A. Zamir and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," arXiv preprint arXiv:1212.0402, 2012..

[18] B. Dubuisson and M. Masson, "A statistical decision rule with incomplete knowledge about classes,," *Pattern Recognition*, vol. 26, no. 1, pp. 155-165, 1993.

[19] D. Chakraborty and N. R. Pal, "Making a multilayered perceptron network say - "Don't Know" when it should," in *9th International Conference on Neural Information Processing*, 2002.

[20] C. Chow, "On Optimum Recognition Error and Reject Tradeoff," *IEEE Transactions on Information Theory*, vol. 16, no. 1, pp. 41-46, 1970.

[21] W. J. Scheirer, L. P. Jain and T. E. Boult, "Probability models for open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2317-2324, 2014.

[22] B. Karmakar and N. R. Pal, ""How to make a neural network say “don’t know”," *Information Sciences*, vol. 430, pp. 444-466, 2018.

[23] T. Bihl, et al., "Easy and Efficient Hyperparameter Optimization to Address Some Artificial Intelligence “ilities”," *Hawaii International Conference on System Sciences*, pp. 943-952, 2020.

[24] M. Ambaum, "Frequentist vs Bayesian statistics-a non-statisticians view," arXiv preprint arXiv:1208.2141, 2012.

[25] M. Cerliani, "When your Neural Net doesn't know: a bayesian approach with Keras," Towards Data Science, Jun. 2020. [Online]. Available: https://towardsdatascience.com/when-your-neural-net-doesnt-know-a-bayesian-approach-with-keras-4782c0818624. [Accessed 2 Feb. 2021].

[26] F. Chollet, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras. [Accessed 9 Feb. 2021].