Quantifying Uncertainty in Deep Learning for Safety-Critical Tasks

Avataran Sethi

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Artificial Intelligence
of the
University of Aberdeen.

Department of Computing Science

2020

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2020

Abstract

Deep Learning has been a key component in the immense increase in popularity of Artificial Intelligence. There are plenty of researches applying Deep Learning in various fields, include safety-critical applications like autonomous vehicles, medical science, manufacturing etc. It is very crucial to know model uncertainty in these fields. There are different methods to obtain uncertainty estimates in Deep Learning like Monte Carlo Dropout. We will demonstrate multiple ways to utilize insights obtained from uncertainty estimates, to develop a better deep learning solutions for safety-critical tasks. In this project we focus on detecting drowsy drivers which is a safety-critical task. We then perform experiments and use the insights from these experiments to improve model and training methods like preprocessing the dataset. We also use uncertainty measures to reduce the false-negative rate.

Acknowledgements

I would like to sincerely say thanks to Dr Dewei Yi for his support and guidance. It is challenging to remotely work on the project and maintain regularity and concentration during the pandemic. Regular weekly meeting with him helped me to maintain regularity in the project.

I would also like to thank various online platforms for freely releasing their content. Lectures from Deep Bayes Summer School, Moscow and Machine Learning Summer School, Tübingen helped me to get into Bayesian Modelling, a relatively new topic for me.

Contents

1 Background and Related Work **8**

1.1 Deep Learning 8

1.1.1 Convolutional Neural Networks 8

1.1.2 Long Short-Term Memory 10

1.1.3 Dropout 11

1.2 Bayesian Neural Networks 12

1.2.1 Bayesian Modelling 12

1.2.2 Variational Inference 13

1.2.3 Monte Carlo Dropout 13

1.2.4 Other Approximate methods 14

1.3 YawDD 15

1.3.1 Related Work 15

2 Experimental Design and Models **16**

2.1 Experimental Design 16

2.1.1 Dataset Preparation 16

2.1.2 Hardware and Software Configuration 17

2.1.3 Evaluation Methodology 18

2.2 Models 18

List of Tables

2.1 Hardware configuration 17
2.2 Software configuration 17

List of Figures

1.1 Sample architecture of CNN, image adapted from Stanford CS231n course slides. 8
1.2 9
1.3 Schematic diagram of LSTM unit, figure adpated from Greff et al. (2017). 11
1.4 Example of Dropout Layer, figure adapted from Srivastava et al. (2014). 12

2.1 Sample input from multi-frame data. 17
2.2 LeNet-5 Architecture. Image adapted from LeCun et al. (1998). 18

Chapter 1

Background and Related Work

1.1 Deep Learning

Neural Network, a computational model based on neurons has become immensely popular and needs no introduction. One of the most important aspects of Neural Network was the ability to stack multiple layers, which was referred as Deep Learning, This allowed the network to learn complex relations from data, which in earlier Machine Learning models were needed to be feature engineered from an expert.

1.1.1 Convolutional Neural Networks

The mechanism of Convolutional Neural Networks(CNN) is inspired from the early work done by Hubel and Wiesel (1962), they discovered two different types of cells: Simple and Complex cells and their responses to different stimuli. The current version of CNN is based on LeCun et al. (1989) and LeCun et al. (1998), in which CNN named LeNet-5 became popular due to it's success over handwritten digit recognition task. The architecture is represented in Figure 2.2.

The principle of CNN model can be understood in two parts: the first part consists of sets of convolution, activation(non-linearity) and pooling layers and it performs feature learning/extraction, while second part flattens the obtained features and passes them through multiple fully connected neural network layers to give the final prediction results. An example of a general CNN architecture is shown in Figure 1.1.

Convolution Layer It extracts features from the input. The layer consists of multiple kernels/filters, which are moved like a sliding window to perform convolution operation over the underlying input. Convolution is performed by taking dot product over the underlying inputs at each position. The filter is a set of weights and they regulate the type of feature to be extracted.

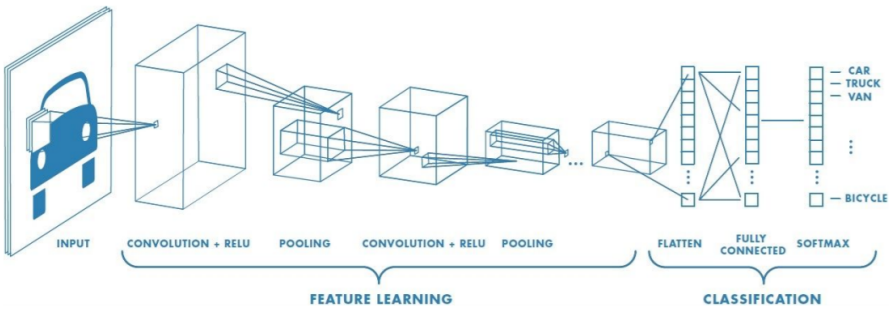
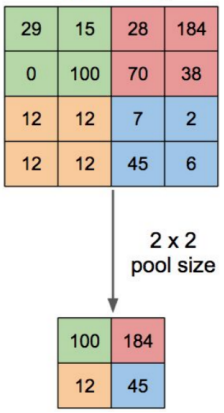
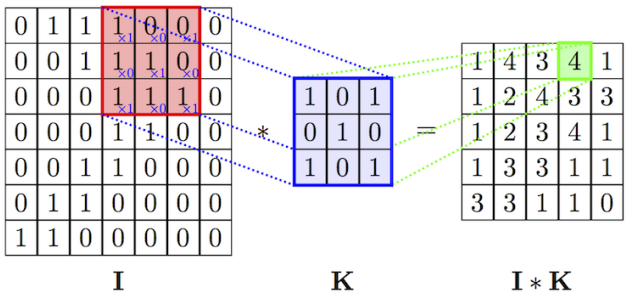


Figure 1.1: Sample architecture of CNN, image adapted from Stanford CS231n course slides.



(a) Example of Max Pooling operation.



(b) Demonstrating convolution operation of convolution layer, with image as I and kernel as K.

Figure 1.2

Using the same filter over the input reduces the number parameters and this is one of the reasons against directly flattening the image and using fully connected layers. This process is controlled by three hyperparameters: filter size, stride and number of filters. Filter size controls the area of convolution. Stride determines the pace of the sliding window. The number of filters determines the number of features extracted from the input. Another important hyperparameter is zero paddings which influence the initial position of kernel and output size. Example of convolution operation is shown in Figure 1.2b.

Activation layer Activation Layer is used to add non-linearity to the network. It has an activation function, which is an element-wise operation thus preserves the dimensions of the input. This operation outputs an activation map of the feature maps generated by convolution layer. There are multiple choices for the activation function. Each has its advantages and disadvantages, but Rectified Linear Units(ReLUs) have been observed to have an edge over others when used with CNN models. New activation functions like SWISH(Zoph and Le, 2018) have been introduced, yet ReLUs remain a popular choice for state-of-art models(Nwankpa et al., 2018). There are also methods to learn the most appropriate activation function(Manessi and Rozza, 2018).

Pooling Layer Pooling layer performs non-linear downsampling over the activation maps received as input. Rationale behind using this layer is to preserve the extracted features while discarding spatial information, thus making model translation invariant(Ranzato et al., 2007). It also reduces computational cost of the model. It has two hyperparameters filter size and stride, which are similar to those in convolution layers. Average Pooling and Max Pooling are the most commonly used methods. It has also been suggested to increase stride and reduce filters in convolution layer instead of using pooling layer. Figure 1.2a demonstrates example of Max Pooling Layer.

Fully Connected Layer This a regular neural network layer. It aggregates the features and maps them to the outputs. One or more layers are stacked to finally output a vector which can be then forwarded through Softmax function to get a set of probabilities for classification task.

Loss Layer The final layer of the model, it returns the loss for the training input. Loss is the distance(difference) between the output from the model and the desired output. There are many functions available to compute loss value. Cross-Entropy loss is most commonly used in these

models for the classification task.

1.1.2 Long Short-Term Memory

Sequential data contains temporal information, which regular Neural Networks(NN) fail to exploit. Since NN treat each instance of the sequence independently and do not have any memory, they fail to utilize the information from prior sequences. Recurrent Neural Network(RNN) introduce a new hidden vector called state vector which acts as a memory unit. At any timestep t, RNN unit will receive input x_t from input sequence and hidden state(memory) vector s_{t-1} from previous unit. Current hidden state vector depends upon both input vectors, and final output h_t depends upon final units hidden state vector. The equations 1.1 formalize these computations.

$$\begin{aligned} s_t &= \sigma(Wx_t + Vs_{t-1} + b_s) \\ h_t &= g(Us_t + b_h) \end{aligned} \tag{1.1}$$

In equations 1.1, U, V and W represent weight matrices and g represents activation function.

Historically, the Hopfield Network by Hopfield (1984) is considered the first network with recurrent connections. RNN are based on work done by Elman (1990). RNNs were found to perform poorly while modelling long term dependencies. While backpropagating error over a large sequence, error gradient was multiplied many times with same weight value which resulted in error gradient becoming too small or too large. This was popularly referred to as Vanishing and Exploding gradient problem. Many solutions were proposed to solve this problem and finally LSTM networks was introduced by Hochreiter and Schmidhuber (1997).

LSTM had a much more complex structure of LSTM unit compared to simple RNN. LSTM used the idea of gating to control the flow of information. Earlier version of LSTM only had *input and output gates*. It would sometimes become unstable when used on long input streams and to solve this *forget gates* were introduced by Gers et al. (1999). Greff et al. (2017) documents and compares many variations of LSTM. Main components of LSTM unit with forget gates is listed below:

1. Input : LSTM unit takes two inputs including input sequence for current time step and output of LSTM unit from previous time step.

2. Input Gate : Applies Sigmoid activation to weighted sum of the inputs.

3. Forget Gate: Forget gate is designed to discard irrelevant information. Sigmoid activation is applied to weighted sum of inputs and the result is multiplied by previous cell state.

4. Memory Cell : It consists of Constant Error Carousel unit, which has a unit weighted recurrent connection of one time step. The Cell State is the record of relevant past information. Current Cell State is calculated by combining past Cell State to forget gate output and then adding it to outputs from input gate.

5. Output Gate : Controls information outflow. Applies Sigmoid activation to weighted sum of inputs.

6. Output : The output of LSTM unit is calculated by multiplying output gate's value to result of *tanh* activation of cell state.

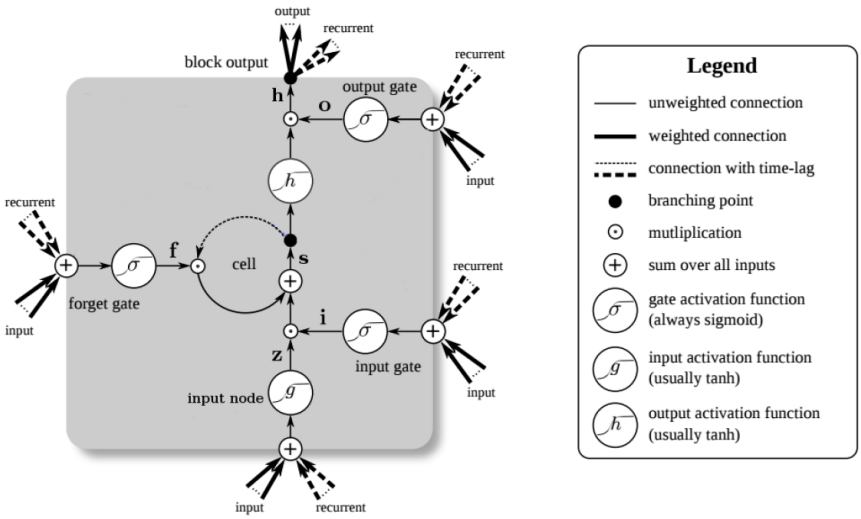


Figure 1.3: Schematic diagram of LSTM unit, figure adpated from Greff et al. (2017).

$$
\begin{aligned}
z_t &= \tanh(W^z x_t + R^z h_{t-1} + b^z) && \text{(input)} \\
i_t &= \sigma(W^i x_t + R^i h_{t-1} + b^i) && \text{(input gate)} \\
f_t &= \sigma(W^f x_t + R^f h_{t-1} + b^f) && \text{(forget gate)} \\
o_t &= \sigma(W^o x_t + R^o h_{t-1} + b^o) && \text{(output gate)} \\
s_t &= z_t \odot i_t + s_{t-1} \odot f_t && \text{(cell state)} \\
h_t &= \tanh s_t \odot o_t && \text{(output)}
\end{aligned}
\tag{1.2}
$$

The Figure 1.3 and Equations 1.2 convey the computation in a LSTM unit. The W^*s in equation are weights, R^*s are recurrent weights and b^*s are biases.

1.1.3 Dropout

Hinton et al. (2012) first introduced the idea of reducing overfitting by dropping a unit in neural network. This idea was inspired from the theory of evolution based on sexual reproduction(Livnat et al., 2010). Srivastava et al. (2014) applied above idea to feed forward Neural Networks. Co-adaptation is one of the major problems in large neural networks[will write more on this]. Dropout probability p is the only hyperparameter required to control this layer. Depending upon the value of the p, it's decided that whether a node will be involved in computation or will it be dropped out. There are other methods similar to dropout, like dropconnect ..... and these are generally referred to as stochastic regularization techniques. Labach et al. (2019) have documented different variations of dropout. So far, none have been conclusively proven to better than other methods. Figure 1.4 shows dropout layer in working.

To formalize the dropout, consider a neural network with L hidden layers. Let input and output vectors indexed by l be denoted by z^l and y^l respectively. W^l and b^l are biases at layer l.

Then for any hidden unit i, of standard neural network, feed forward operation can be defined as:

$$\begin{aligned} z_i^{l+1} &= w_i^{l+1} y^l + b^{l+1} \\ y_i^{l+1} &= g(z_i^{l+1}) \end{aligned}$$

After applying dropout, feed forward operation changes to:

$$\begin{aligned} r_j^l &\sim Bernoulli(p) \\ \widetilde{y^l} &= r^l * y^l \\ z_i^{l+1} &= w_i^{l+1} \widetilde{y^l} + b^{l+1} \\ y_i^{l+1} &= g(z_i^{l+1}) \end{aligned} \tag{1.3}$$

here g is any activation function and $*$ denotes element wise multiplication.

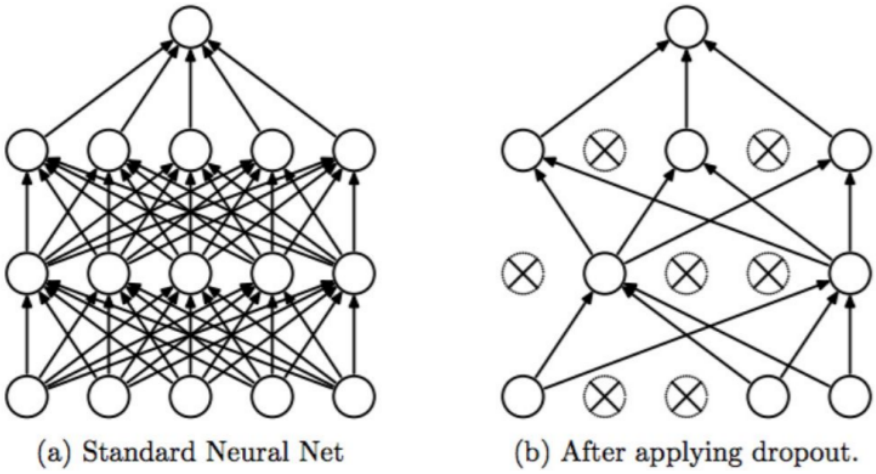


Figure 1.4: Example of Dropout Layer, figure adapted from Srivastava et al. (2014).

1.2 Bayesian Neural Networks

Probabilistic models offer statistical bounds over model's results, i.e. the mean and the variance of the distribution. We can use these bounds as an estimate of the confidence we should have over the results. In contrast Deep Learning models only provide us with the point estimates, thus we cannot gauge models' confidence in these results.

1.2.1 Bayesian Modelling

Let $\mathcal{D} = \{(X,Y)\}$ denotes a dataset with X,Y denoting input-output pairs. Let θ be the parameters parameterising function $y = f^\theta(X)$. Bayesian approach would be to place a distribution $p(\theta)$ called prior over the parameters, it represents our prior knowledge or belief. We would want update our prior beliefs as the data is observed, for this we would require likelihood distribution given by $p(y|x,\theta)$. Likelihood can be understood as chances of observing the output from given inputs and parameters. Updated distribution called posterior distribution using Bayes theorem is :

$$p(\theta|Y,X) = \frac{p(Y|X,\theta)p(\theta)}{p(Y|X)}. \tag{1.4}$$

We use the posterior distribution to perform inference, i.e. to get a predictive distribution for new inputs. The normaliser for posterior distribution, also called model evidence is calculated from:

$$p(Y|X) = \int p(Y|X,\theta)p(\theta)d\theta. \tag{1.5}$$

This computation is also called Marginalising likelihood over θ. Marginalisation is the key area for Bayesian analysis. Apart from few cases it's not possible to calculate evidence and posterior. And thus the need for approximation arises.

1.2.2 Variational Inference

Variational inference is one of the ways to approximate posterior distribution. We assume a easy to evaluate family of distribution called Variational Distribution $q_\omega(\theta)$ parameterised by ω. We will use KL Divergence to get the closest approximation to true posterior. KL Divergence is the measure of similarity(or Dissimilarity), so minimising this can be understood as reducing dissimilarity.

$$KL(q_\omega(\theta)||p(\theta|X,Y)) = \int q_\omega(\theta)\log\frac{q_\omega(\theta)}{p(\theta|X,Y)}d\theta \tag{1.6}$$

KL Divergence minimization equation can be rearranged to maximise the evidence lower bound(ELBO).

$$ELBO(q) := \int q_\omega(\theta)\log p(Y|X,\theta)d\theta - KL(q_\omega(\theta)||p(\theta)) \tag{1.7}$$

Maximising first term of ELBO can be understood as closely explaining the observed data and minimising the second term can intuitively be explained as making Variational distribution to be closer to prior. This process is called Variational Inference(Jordan et al., 1999). Using this approach we can apply Bayesian Modelling to wider selection of problems. However, problems with larger data and complex models are still hard to solve using this technique.

1.2.3 Monte Carlo Dropout

Deep Learning models with stochastic regularization techniques(SRT) like dropout can be interpreted as approximate Bayesian Neural Networks. Gal and Ghahramani (2016) have shown that NN architectures with dropout between each layer, can be sampled to get Monte Carlo estimate of models predictive variance. They also show that this interpretation can be extended to other SRT. For practical purposes, the difference between standard dropout and Monte Carlo dropout is that for Monte Carlo dropout feed-forward behaviour described in Equation 1.3 remains active in testing phase too. In standard dropout, dropout functionality only remains active in the training phase, during testing output is scaled by $\frac{1}{1-p}$, where p is dropout probability. While in MC Dropout feed-forward behaviour (1.3) is active while making predictions, so every forward pass would be stochastic. This stochastic behaviour is proved to be a Bernoulli approximation of variational inference. This approximation is further extended to CNN(Gal and Ghahramani, 2015) and RNN(Gal, 2016). Concrete Dropout (Gal et al., 2017) provides an alternative method to calibrate uncertainty estimates. Concrete dropout can be used instead of using a grid search over different model precision values, as grid search operation can be computationally exhaustive for bigger models.

Uncertainty measures for classification task variation ratios, predictive entropy(Shannon, 1948) and mutual information(Shannon, 1948) are three measures which can be employed to estimate uncertainties in classification tasks(Gal, 2016). Variation ratio can be understood as fraction of predicted classes different from mode predicted class. To calculate variation ratio, we first select the class with maximum probability(highest softmax output) as predicted class, for every stochastic pass of total T passes. We then find frequency f_n of mode class. The equations 1.8 and 1.9 summarise the calculation.

$$\begin{aligned} c^* &= \underset{c=1,..,C}{\operatorname{argmax}} \sum_t \mathbb{1}[y^t = c] \\ f_n &= \sum_t \mathbb{1}[y^t = c^*] \end{aligned} \tag{1.8}$$

$$\textit{variation-ratio} \coloneqq 1 - \frac{f_n}{T} \tag{1.9}$$

Predictive Entropy has its roots in information theory. It measures average amount of information in predictive distribution, $\mathbb{H}[y|x,\mathcal{D}_{train}]$. We approximate predictive entropy by averaging Softmax probabilities for T passes and multiplying it by log of that average, for each class and then summing over all classes. These calculations are summarised in Equations 1.10 and 1.11, where f^θ is network with model parameter θ.

$$\textit{softmax}(f^\theta(x)) \coloneqq [p(y=1|x,\hat{\theta}_t),...,p(y=C|x,\hat{\theta}_t)] \tag{1.10}$$

$$\widetilde{\mathbb{H}}[y|x,\mathcal{D}_{train}] \coloneqq -\sum_c (\frac{1}{T}\sum_t p(y=c|x,\hat{\theta}_t)).\log(\frac{1}{T}\sum_t p(y=c|x,\hat{\theta}_t)) \tag{1.11}$$

Mutual information is the last measure in the list. From Equation 1.12, we can see that calculation of mutual information, $\mathbb{I}[y,\theta|x,\mathcal{D}_{train}]$, is similar to predictive entropy with an extra term. In Section **??** we take a deeper look into these measures from point of view of our experimental settings.

$$\begin{aligned} \widetilde{\mathbb{I}}[y,\theta|x,\mathcal{D}_{train}] \coloneqq &-\sum_c (\frac{1}{T}\sum_t p(y=c|x,\hat{\theta}_t)).\log(\frac{1}{T}\sum_t p(y=c|x,\hat{\theta}_t)) \\ &+\frac{1}{T}\sum_{t,c} p(y=c|x,\hat{\theta}_t)\log p(y=c|x,\hat{\theta}_t) \end{aligned} \tag{1.12}$$

1.2.4 Other Approximate methods

There are other methods to perform approximate inference in BNNs: Hamiltonian Monte Carlo(HMC), Mean Field Variational Inference(MFVI). In HMC(Neal, 2012), a Markov Chain with invariant distribution $p(w|\mathcal{D})$, is defined and Hamiltonian dynamics is used to improve the state space exploration speed. In MFVI(Graves (2011), Blundell et al. (2015)), Guassian approximating distribution $q(w) \approx p(w|\mathcal{D})$ is used, where $q(w)$ is assumed to be dependent on some hyperparameters, which can be optimised by minimising divergence between $q(w)$ and $p(w|\mathcal{D})$. Jospin et al. (2020) summarises the differences between these methods.

1.3 YawDD

YawDD: A Yawning detection dataset (Abtahi et al., 2014), has two video datasets. Each dataset has drivers with various facial characteristics. This video dataset was designed to detect drowsiness and fatigue. Videos have drivers in three different mouth positions: normal, talking/singing, and yawning. First video dataset has a camera mounted at rear-view mirror(front mirror) and each driver has three-four videos and in each video has only one of the mouth position. The second video has a camera mounted at the dash and each driver has one video with all mouth positions. We can use the dataset to train models to detect dangerous driving behaviours by detecting talking and yawning.

1.3.1 Related Work

Previously, models have been used to specifically extract facial features like blinking eyes, mouth opening, nose orientation etc. These works have not been compared as the aim of the paper was to compare NNs with BNNs. Zhang and Su (2018) have used pre-trained GoogleNet CNN model and fine-tuned it to yawning dataset and then it is used with three-layered LSTM network. Xie et al. (2019) have used pre-trained Inception-V3 CNN model with one layer LSTM.

Chapter 2

Experimental Design and Models

2.1 Experimental Design

2.1.1 Dataset Preparation

As mentioned in Section 1.3, YawDD contains two video datasets. The video dataset involved in the project for training models is the one with the camera mounted on the front mirror. In this dataset, each driver has videos for all three mouth position and in a single video, there is only one mouth position. But, there is a problem in this assumption, videos labelled as yawning and talking will also have frames with the mouth in normal position. So, there will be frames with normal mouth position but labelled as talking or yawning and thus it may impact the model's training. We will talk more about it in Section **??**. We will prepare datasets for different models and also try to offset the above problems.

Dataset with single frames We use this dataset to train CNN models. We sample frames at a rate of one fps from videos and label each frame with the label of the sampled video. This dataset can also be used to train LSTMs with outputs at each time-step. We resize the frames to 128*128 pixels for our experiment. Figure **??** shows samples from dataset.

Sequential Dataset We use this dataset to train the LSTM model. We sample frames at two fps rate and maintain the sequence. We use a single training output for the whole video sequence. Although it doesn't make sense to use a model trained on this data on real-time prediction task, we use this experiment as an attempt to offset the effect of wrong labelling. Also, LSTM network may become too large to be able to train.

Multi Frame dataset We use this dataset with both CNN and LSTM network. We try to avoid the mislabelling problem, as well as train real-time usable model. The idea behind this dataset is to sample and group frames in such a way that it includes at least one frame that has labelled mouth position and also sequence length is small in size. We group frames sampled from sequential data while preserving temporal information of data. We set the number of frames per sequence for the dataset, in our experiment we used six frames. We divided each video sequence into six equal segments and each new sequence had one frame from each segment. This will ensure that when the length of the video varies, there will always be a segment which will provide the correct frame to the sequence. Algorithm 1 formalises this sampling method. We can use this dataset with CNN by joining all the frames in a sequence into a single image. Figure 2.1 shows a sample image from the dataset.

Algorithm 1: Sampling Algorithm for multi-frame dataset

```
input : Video dataset
output: Multi-frame dataset
new_dataset ← [];
for each row in video dataset do
    sequence_length ← Length(row);
    num_frames ← 6 ;  // number of frames per sequence of new dataset
    num_seq ← sequence_length ÷ number_frames;
    for i ← 0 to num_seq do
        temp ← [];
        foreach j ← 0 to num_frames do temp.Append(row[j*num_seq+i]);
        new_dataset.Append(temp);
    end
end
```

Figure 2.1: Sample input from multi-frame data.

Dataset for real-time analysis We use videos recorded from dash mounted camera to simulate real-time analysis. This also gives an opportunity to analyze models performance on out-of-distribution data.

2.1.2 Hardware and Software Configuration

Hardware All the experiments were done in Google Colaboratory environment. It's a virtual machine environment and configurations may vary over the time. General hardware configuration is mentioned Table 2.1.

Software All of the code except concrete dropout models is compatible with both Tensorflow 1 and 2. Since Tensorflow 2 has brought in significant changes to defining custom layer, concrete dropout code is not compatible with Tensorflow 2. We can choose Tensorflow version in Colab environment. Summary of software configuration is listed in Table 2.2.

Acceleration	GPU
Runtime	Python 3.6
GPU	Tesla K80
CPU	Intel Xenon CPU @ 2.20GHz
RAM	13GB
Disk	64GB

Table 2.1: Hardware configuration

Software	Version
Python	3.6
Tensorflow	1.15.2
Keras	2.2.4

Table 2.2: Software configuration

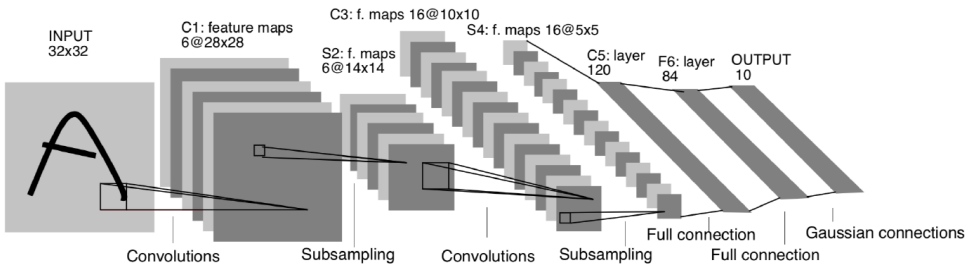


Figure 2.2: LeNet-5 Architecture. Image adapted from LeCun et al. (1998).

2.1.3 Evaluation Methodology

Classifying dangerous driving behaviour is a safety-critical task, so our final motive is to choose the model with the least false-negative scores. To simulate real-life scenario, we only train model at 20 per cent of data and test on the remaining 80 per cent. This approach will increase the chances of having out of distribution data in the testing set, which will create a similar situation to that in real-life scenarios where we often encounter new data points far from training data. This will also help in establishing the importance of having uncertainty measures. We will compare accuracy scores and confusion matrix for each model and dataset preparation methods. Most importantly we will analyse models' performance using uncertainty measures. We will also compare the uncertainty quantification of Bayesian Neural Networks.

2.2 Models

LeNet LeNet-5 architecture (LeCun et al., 1998) is used for classification. Feature detection part consists of two pairs of Convolution Layer and then Pooling layer and then two fully connected hidden layers. Since there are only three driving behaviours, so the output of the softmax layer is changed to 3. This model will be used as the baseline model for experiments. Figure 2.2 shows LeNet-5 model architecture.

LeNet with LSTM A combination of CNN and LSTM is used for the classification task. A single layer LSTM is inserted after feature extraction part of LeNet. An extra fully connected layer is used between the flatten layer and LSTM layer to reduce the number of input parameters for the LSTM layer. The remaining structure is the same as the baseline model. We will refer to this model as lenet-lstm. Since this model is utilizing temporal data and these models have previously performed better than CNN models, we will consider this model as an upper limit for other models. Although in past work, pre-trained CNN model was used to extract features from images. But, we are training the model from scratch to maintain similarity for model comparison.

LeNet with Dropout LeNet model with standard dropout (Srivastava et al., 2014) is used. We have used dropout layer in two ways: dropout only before the softmax and dropout layer before pooling layers in addition to the softmax layer. We will refer to this model as lenet-drp.

LeNet with Monte Carlo Dropout The model follows the probabilistic interpretation of the CNN model (Gal, 2016). In practice, to get Bernoulli approximate variational inference of convolution operation we should just use dropout layer before pooling layer. This can be understood as setting kernels to zero for random patches. We have used LeNet model with dropout before pooling layers

and before fully connected layers. We will refer to this model as lenet-mc-drp.

LeNet with Concrete Dropout Concrete dropout (Gal et al., 2017) is used to calibrate uncertainty through continuous relaxation of dropout probability. Architecture is the same as lenet-mc-drp, except concrete dropout layer is used instead of a dropout layer. We will refer to this model as lenet-cnc-drp.

Bibliography

Abtahi, S., Omidyeganeh, M., Shirmohammadi, S., and Hariri, B. (2014). YawDD: A yawning detection dataset. *Proceedings of the 5th ACM Multimedia Systems Conference, MMSys 2014*, pages 24–28.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *32nd International Conference on Machine Learning, ICML 2015*, 2:1613–1622.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis.

Gal, Y. and Ghahramani, Z. (2015). Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. pages 1–12.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *33rd International Conference on Machine Learning, ICML 2016*, 3:1651–1660.

Gal, Y., Hron, J., and Kendall, A. (2017). Concrete dropout. *Advances in Neural Information Processing Systems*, 2017-Decem:3582–3591.

Gers, F. A., Urgen Schmidhuber, J. J., and Cummins, F. (1999). Learning to Forget: Continual Prediction with LSTM. Technical report.

Graves, A. (2011). Practical Variational Inference for Neural Networks. Technical report.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. pages 1–18.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 81(10 I):3088–3092.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). Introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., and Bennamoun, M. (2020). Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users. 1(1):1–35.

Labach, A., Salehinejad, H., and Valaee, S. (2019). Survey of Dropout Methods for Deep Neural Networks.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to digit recognition.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323.

Livnat, A., Papadimitriou, C., Pippenger, N., and Feldman, M. W. (2010). Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences of the United States of America*, 107(4):1452–1457.

Manessi, F. and Rozza, A. (2018). Learning Combinations of Activation Functions. *Proceedings - International Conference on Pattern Recognition*, 2018-Augus:61–66.

Neal, R. M. (2012). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. pages 1–20.

Ranzato, M., Huang, F. J., Boureau, Y. L., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(4):623–656.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Xie, Y., Chen, K., and Murphey, Y. L. (2019). Real-time and Robust Driver Yawning Detection with Deep Neural Networks. *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*, pages 532–538.

Zhang, W. and Su, J. (2018). Driver yawning detection based on long short term memory networks. *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings*, 2018-Janua:1–5.

Zoph, B. and Le, Q. V. (2018). Searching for activation functions. *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, (1):1–12.