Multi Class, Multi Label Image and Text Classification

using Convolutional Neural Networks and Bi-LSTM

Hari Nath Bingi

May, 2020

Contents

1 Introduction **6**

1.1 Aim of the Study 6

1.2 Importance of the Study 6

1.3 Methods and Motivation 6

2 Related Works **6**

3 Techniques **8**

3.1 Data Augmentation 9

3.1.1 Principle 9

3.1.2 Justification 9

3.1.3 Advantages and Novelties 9

3.2 Mini Batches 9

3.2.1 Principle 9

3.2.2 Justification 9

3.2.3 Advantages and Novelties 10

3.3 Batch Normalisation 10

3.3.1 Principle 10

3.3.2 Justification 10

3.3.3 Advantages and Novelties 10

3.4 Word2Vec 11

3.4.1 Principle 11

3.4.2 Justification 11

3.4.3 Advantages and Novelties 11

3.5 Bi-LSTM 11

3.5.1 Principle 11

3.5.2 Justification 12
3.5.3 Advantages and Novelties 12
3.6 ReLU Activation 12
3.6.1 Principle 12
3.6.2 Justification 12
3.6.3 Advantages and Novelties 13
3.7 Leaky ReLU 13
3.7.1 Principle 13
3.7.2 Justification 13
3.7.3 Advantages and Novelties 13
3.8 Sigmoid 14
3.8.1 Principle 14
3.8.2 Justification 14
3.8.3 Advantages and Novelties 14
3.9 Convolutional Layer 15
3.9.1 Principle 15
3.9.2 Justification 15
3.9.3 Advantages and Novelties 15
3.10 Pooling Layer 16
3.10.1 Principle 16
3.10.2 Justification 16
3.10.3 Advantages and Novelties 16
3.11 Fully Connected Layer 16
3.11.1 Principle 16
3.11.2 Justification 16
3.11.3 Advantages and Novelties 17
3.12 Early Stopping 17

3.12.1 Principle . . . 17
3.12.2 Justification . . . 17
3.12.3 Advantages and Novelties . . . 17
3.13 Adam Optimizer . . . 17
3.13.1 Principle . . . 17
3.13.2 Justification . . . 18
3.13.3 Advantages and Novelties . . . 18
3.14 Binary Cross Entropy with Weighted Loss . . . 18
3.14.1 Principle . . . 18
3.14.2 Justification . . . 18
3.14.3 Advantages and Novelties . . . 19

4 Experiments and results 19
4.1 Convolutional Neural Network . . . 19
4.1.1 Pre-trained models . . . 19
4.1.2 Data Augmentations . . . 20
4.1.3 Epochs . . . 22
4.1.4 Batch Size . . . 22
4.1.5 Learning Rate . . . 23
4.2 Bi-LSTM Network . . . 24
4.2.1 Embedding Size . . . 24
4.2.2 Gensim Data Set . . . 24
4.2.3 Sequence Length . . . 25
4.2.4 Epochs . . . 26
4.2.5 Batch Size . . . 27
4.2.6 Learning Rate . . . 28
4.3 Combined Neural Network . . . 29

4.3.1 Epochs 29
4.3.2 Batch Size 29
4.3.3 Learning Rate 30
4.3.4 Drop Out 30
4.3.5 Hidden Layer size 32

5 Discussion and Conclusion **32**

6 Appendix **39**
6.1 Final Architecture 39
6.1.1 First Network - CNN 39
6.1.2 Second Network - Bi-LSTM 40
6.1.3 Third Network - Combined 41
6.1.4 Final Results 42
6.2 Hardware and Software Specifications 45
6.3 Running the Code 45

1 Introduction

1.1 Aim of the Study

The aim of this study is to construct an optimal image classifier using Convolutional Neural Networks. The model should be designed with high efficiency and accuracy based on the given data set.

1.2 Importance of the Study

Convolutional Neural Networks (CNNs) are being widely used in various industries through their superior abilities in visual recognition tasks as in medical imaging, traffic surveillance and security[1]. It is clear that CNNs are becoming more attractive to use for many applications, but with more complex tasks, traditional models may not perform sufficiently. Therefore, it is important to analyse past models and deeply evaluate their features in order to guide the construction of an optimal and efficient convolutional neural network.

1.3 Methods and Motivation

As mentioned before, CNN architecture has been revised and tweaked over the years to increase accuracy. The classifier made is mainly based off ResNet because it is a fairly recent model which had a ground-breaking accuracy in a selection of computer vision tasks which won many competitions. The key feature of this network was that it still maintained very deep layers which other models had a decrease in accuracy. Having many layers aids in the analysis of more complex data sets.

2 Related Works

There has been a boom in popularity in computer vision since 2012, where Alex Krizhevesky and others [18] proposed a deep Convolutional neural network architecture model called AlexNet which won the most difficult Ima-

geNet challenge for visual object recognition. It was recognized as a breakthrough in the area of machine learning. Since then, the interest in computer vision and deep learning has been growing rapidly among researchers, especially with advances in technology to accommodate larger, more complex networks. In 2013, Matthew Zeiler and Rob Fergue [33] made improvement based on AlexNet, called ZFNet. They optimized the parameters of AlexNet by reducing the numbers required dramatically which improved the overall accuracy of classification. In 2014, VGG net[28] was proposed. The main contribution of VGGNet is that it showed the vital role depth had in the network to achieve even better accuracy in classification or recognition using CNN. GoogleNet[29] proposed in 2014, a reduction in complexity compared to other network while keeping satisfying performance. The number of parameters and computations are significantly lower than AlexNet or VGG net. In 2017, Gao Huang and others proposed DenseNet[17] which consists of densely connected layers and all the layers' output are connected in a dense block. In this assignment, we will use ResNet as our base model.
ResNet is a powerful model proposed by He et al.,[14] in 2015. The benefit of this model is that it creates very deep networks which allows for better feature extraction and it also reduces the complexity by removing the fully connected layer and the dropout property. It addressed the difficulty of training deep neural networks by using smaller filters but more of them. This model is an all rounder since it placed first in 5 main tracks in the fields of image classification, detection and localisation. The details of this model can be summarised in the 2016 paper, "Deep Residual Learning for Image Recognition" by He et al.

Another model used in the assignment is Bi-LSTM, which is an architecture of recurrent neural networks (RNNs). RNNs are artificial neural networks (ANN) with recurrent connections. These are used to model sequential data for sequence recognition and prediction [4]. Deep architectures of neural networks are exponentially more efficient than other shallow architectures[12]. In 2013, R Pascanu et al.[22] proposed a deep recurrent neural network with multi-layer perceptron that significantly increased the performance of a network. A Bi-directional RNN (BRNN) was proposed as a solution which considers the forward and reverse input sequences for predicting the past and future[27]. Furthermore, adopting reccurent connections into each convolutional layer results in a network called recurrent convolutional neural network[19]. It increases the depth of model, while the number of parameters

is constant by weight sharing between layers. Multi-dimensional recurrent neural networks [3] are another implementation of RNNs with high dimensional sequence learning. This network utilizes recurrent connections for each dimension to learn correlations in the data.
Although recurrent connections improve the performance of neural networks, the memory produced is limited by the algorithms. As a result, models may have exploding or vanishing gradients during training phase and the network cannot learn long term sequential dependencies in data. The most popular solution is to use long-short term memory (LSTM) RNNs [26]. It helps reduce the effects of vanishing and exploding gradients because of the forget gate feature[16]. LSTM has many different models, however, we used Bi-LSTM in this assignment.
It is possible to increase capacity of RNN by stacking hidden layers of LSTM cells in space, which makes a deep bi-directional LSTM[12]. Bi-LSTM networks are more powerful in comparison to other unidirectional LSTM networks[13]. It contains all information of the input sequences during training. The distributed representation feature of Bi-LSTM can be useful in different applications like natural language processing[30]. Similar to Bidirectional RNN, Bi-LSTM has the same advantages, whilst overcoming the vanishing or exploding gradient problem[26].

3 Techniques

The main techniques used to construct this image classifier is:

- **Batches**: Mini-batches, Batch Normalisation, Epochs
- **Activation functions**: ReLU, Leaky ReLU, Sigmoid
- **Natural Language Processing**: Word2Vec, Bi-LSTM
- **Convolutional Neural Network**: Data Augmentation, Convolutional layer, Pooling layer, Fully Connected layer, Transfer Learning
- **Optimizer/Regularisation**: Adam
- **Loss**: Binary Cross Entropy with Weighted Loss

3.1 Data Augmentation

3.1.1 Principle

Data Augmentation is a popular technique used in training CNNs to transform data in various ways to generate new inputs. Examples include skewing, resizing and translation.

3.1.2 Justification

It has extensive use in the CNN field and can cause a substantial increase in accuracy by helping the network adapt more easily to different inputs. Since we wish to optimise the classification ability of the neural network, it should be robust for many types of images. It is fairly straight-forward to implement with no specific fine-tuning of hyperparameters.

3.1.3 Advantages and Novelties

It allows the network to learn more complex features and avoid over-fitting. Not only does it expand the data set but also provides better generalisation to give a more robust model that is invariant to mutations[2].

3.2 Mini Batches

3.2.1 Principle

In the training phase, the data set is split into groups of a defined batch size. Weight updates are made according to the number of batches whereby smaller number of batches will be faster but less accurate and vice versa for larger number of batches.

3.2.2 Justification

Stochastic gradient descent is extremely time inefficient as it has to iterate the whole data set before making a single weight update. Batch gradient descent reduces the number of iterations, hence allowing faster learning[7].

3.2.3 Advantages and Novelties

A big factor is time efficiency as the network needs to update the weights less, once per batch. Additionally, it helps to remove noise in the data because of its summarising capabilities.

3.3 Batch Normalisation

3.3.1 Principle

In addition to batch training, batch normalisation is where the data is constrained to the same range. The purpose of this is to ensure that some features don't overshadow the effects of others which may lose important information. Additionally, it addresses the problem of covariate shift whereby the feeding of outputs of one layer into the inputs of the next propagates the data in a way such that may lead to exploding or vanishing gradients[9].

3.3.2 Justification

As we are using images as input which might have different distribution sources which might force each intermediate layer to continuously adapt to its changing inputs hence leading to covariate shift. Implementation of Batch normalization helps to overcome this issue. Not only this but normalisation also helps each feature have an equal impact on the model.

3.3.3 Advantages and Novelties

The network is able to converge faster because of the small values. It also allows for higher learning rates which is especially important for deep networks like our for faster execution. It limits the chances of exploding and vanishing gradients by regulating the value and also the need for other regularisation methods such as dropout.

Natural Language Processing

3.4 Word2Vec

3.4.1 Principle

Word2Vec is a group of models which are used to make word embeddings. They are 2-layer Neural Networks which are trained to obtain the linguistic context of words. It does so by taking a raw unlabelled corpus input and produces a vector space. Each word is then assigned a vector in the space and positiones appropriately. Words with similar context are located close to each other while dissimilar words in vice versa[32].

3.4.2 Justification

Since the data set includes captions for the images, it is wise to utilise these words to better understand the input and gain a more accurate representation using this additional information. To process these sentences, they must be transformed into vectors which is why this method is used.

3.4.3 Advantages and Novelties

It requires little memory and allows for increased learning of the model. Word2Vec also simple and intuitive to use.

3.5 Bi-LSTM

3.5.1 Principle

Bi-direction Long Short Term Memory (Bi-LSTM) is an architecture that runs input in two ways in a sequential manner. Bi-LSTM utilises a feedback loop with a memory cell whereby the past inputs affect the calculations of the next layer. This is helpful to remember only the relevant information. The bi-directional component reverses the input and feeds it in the opposite direction which allows for more complex and contextual information to be extracted[15].

3.5.2 Justification

This is useful for the captions of each image because the order is important for its semantic meaning. It is better than standard RNNS because it addresses the vanishing gradient problem where the accumulation of tanh activations is removed. This is done through the use of a forget gate to remove irrelevant information in the learning process

3.5.3 Advantages and Novelties

The LSTM method solves the vanishing gradient problem of standard RNNs. It is also nsensitive to gap length or long sequences thorugh the use of the forget gate which simplifies the learning process of the network by only remembering the important features.

Activation Functions

It is actually wiser to explore combinations of activation functions for better performance. This was depicted in Manessi and Rozza's paper in 2019, where the combined activation functions outperformed the corresponding base components[10]. Therefore various activation functions were utilised as listed below.

3.6 ReLU Activation

3.6.1 Principle

The Rectified Linear Unit, also known as ReLU, is a piece-wise linear function which outputs the value of the input if it is positive. All other values will be outputted as zero[6].

3.6.2 Justification

ReLU is a widely used activation function which has become the default for many CNNs. It avoids the likelihood of vanishing gradient which can be seen

in the tanh or sigmoid functions. Additionally, it is useful for deep networks like ours which is based off ResNet.

3.6.3 Advantages and Novelties

ReLU is computationally efficient and allows for better convergence than sigmoid. It still acts as a linear function which is easier to optimise and it produces good results in deep networks.

3.7 Leaky ReLU

3.7.1 Principle

Leaky ReLU is an alternative to regular ReLU where input values x lower than 0 will have a small negative slope instead of a gradient of 0. The only point at which the output of this activation function is 0 is when $x = 0$ which is different from ReLU where it outputs 0 for all inputs $x \leq 0$[20].

3.7.2 Justification

Unfortunately, ReLU units can be fragile during training and can "die". This is called "dying ReLU". Should a neuron ever becomes negative, it will always output 0 in standard ReLU and will never recover, so the neuron becomes useless. This hinders the learning process as it may affect a large part of the network. By changing the function to output 0.01 to the input should it become negative, allows for the neurons to work around this dying ReLU problem.

3.7.3 Advantages and Novelties

Leaky ReLu activation plays an important role by Being faster and more efficient than most of the activation functions (as it does not involve much computations). It avoids no vanishing gradients, just like ReLU but also avoids dying ReLU so the neurons remain functional and continue the learning process.

3.8 Sigmoid

3.8.1 Principle

Sigmoid activation is different from ReLU as it follows an S-shaped curve where the output values are between 0 and 1. This is utilised to predict the probability of an output[31].

3.8.2 Justification

Since ReLU and Leaky ReLU have already been implemented, piece-wise, these are very linear operations that grow in the output very quickly. Since sigmoid has values between 0 and 1, it allows for regulation. ReLU and Leaky ReLU already treat the vanishing gradient, so it is justifiable to use the sigmoid function. This is to be used after the Fully Connected Layer which is explained in section 3.11.

3.8.3 Advantages and Novelties

Sigmoid allows for decreased growth in the output values because of the 0-1 range and its combination with the two other activation functions adds more non-linearity to allow for more complex features to be extracted.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which takes in an input image and assigns importance via learnable weights and biases to aspects of the image which allows differentiation between them[25].

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. A CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters, which is not possible in a normal neural network. The pre-processing required in a ConvNet is also much lower as compared to other classification algorithms. The CNN is made up of 3 types of layers:

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

3.9 Convolutional Layer

3.9.1 Principle

The Convolutional layer is the core building block of a CNN which allows the network to learn filters corresponding to local regions of the input. It performs feature extraction of the images[23].

Every filter is small spatially, but extends through the full depth of the input volume. The convolutional layer produces feature maps that represent the low-level features of the input, for example edges and colour. A combination of these feature maps allow for higher level features to be extracted.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality in comparison to the input, one in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in case of the former, or Same Padding in the case of the latter. Padding is where the zero value pixels are added around the image to change its shape before CNN processing.

3.9.2 Justification

In order to extract the most vital features of the images, the network requires for these layers to be used, particularly smaller filters.

3.9.3 Advantages and Novelties

It requires less parameters due to weight sharing. It also learns complex features by combining the low level feature maps and performs particularly well with small features for even better feature extraction with a high receptive field if many are used.

3.10 Pooling Layer

3.10.1 Principle

The pooling layer is made to reduce the spatial size of the image representation. It also reduces the amount of parameters and computation in the network. Two types were analysed: Max pooling and Average pooling.

3.10.2 Justification

The network works with large images in high volumes, therefore, to decrease the number of calculations, pooling was a very suitable option. Since max pooling performs de-noising along with dimensionality reduction, it was more favorable than average pooling[24].

3.10.3 Advantages and Novelties

Pooling layer helps in decreasing the computational power required to process the data through dimensionality reduction. It also helps in extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model.

3.11 Fully Connected Layer

3.11.1 Principle

Fully connected layers are where all inputs from one layer are connected to every activation unit in the next layer. It essentially takes the outputs from the previous convolutional layers and aggregates them to represent high-level features.

3.11.2 Justification

In order to classify the input images, it is not enough to use only the local features. Therefore, it is required to implement this layer at the end to combine the results from the previous layers.

3.11.3 Advantages and Novelties

This layer allows for more accurate weight calculations and classifications and also lets the convolutional layers stay small in size.

Regularisation

3.12 Early Stopping

3.12.1 Principle

This regularisation technique involves training the data up to a certain point and then stopping training before it over-generalises the results.

3.12.2 Justification

Early stopping is a simple and easy to understand method that disallows redundant training and saves execution time by running less iterations. It is trivial to include in this model.

3.12.3 Advantages and Novelties

It disallows over-fitting, reduces execution time and ensures that the results are consistent.

3.13 Adam Optimizer

3.13.1 Principle

Adam is an adaptive learning rate optimization technique that attempts to find individual learning rates for each parameter[8].It utilises exponentially moving averages on the gradients to update the learning rate of each weight of the network.

3.13.2 Justification

This has a benefit over stochastic gradient descent which uses a single learning rate value, hence, Adam is more flexible and can handle more complex and larger inputs. The data set is full of thousands of images with many pixels with high variance therefore, it is more suitable to use Adam.

3.13.3 Advantages and Novelties

Since it is adaptive and the step size of the update is invariant to the magnitude of the gradient, it can navigate around saddle points and ravines to further improve gradient descent. It also converges very fast because of the implementation of momentum.

Loss functions

3.14 Binary Cross Entropy with Weighted Loss

3.14.1 Principle

Cross Entropy is a generally used loss function for multi-label classification task as it predicts the probability distribution amongst each class. Since there is a ground truth and the predicted output, this problem can be constructed into a collection of binary classifications where an element belonging to a certain class should not influence the decision for another class[11]. It utilises a principle called the Maximum Likelihood Estimation which addresses the imbalance of classes. Adding more weight adjusts the importance of the positive labels.

3.14.2 Justification

The model needs to be able to classify a collection of images into their appropriate target label. To do so, it is imperative to determine the highest probability to belong to a certain class and assign the label.

3.14.3 Advantages and Novelties

It maximises the likelihood estimation through the use of weights to adjust the probabilities and as a result, these probabilities will lead to a more accurate classification of the inputs.

4 Experiments and results

The following model is summarised in 3 main components: Bi-LSTM for the captions, CNN for the image processing and the combined network to join them together.

4.1 Convolutional Neural Network

4.1.1 Pre-trained models

The models utilised belong to the ResNet group. This includes ResNet50, ResNet152, Wide ResNet101 and ResNext101. The size and performance of these were measured. The below plot indicates the f1-score of these models on validation set and the size of the bubble corresponds to the model size.

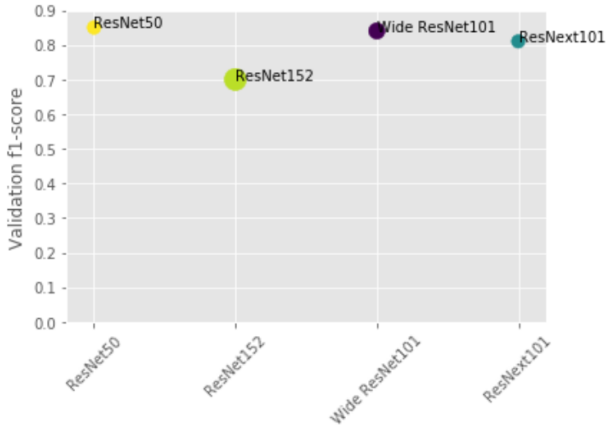

Figure 1: Size and f1-score comparison of ResNet variations

4.1.2 Data Augmentations

A number of data augmentations were used and can be summarised into tests. Their effects on the validation can be depicted in Table 1. It is also summarised graphically in Figure 2 to depict the accuracy differences as expressed in Validation f1-score.

Test No.	Data Augmentations	Validation-f1	Effect	Action Taken
1	Random Crop	0.64	Decrease in f1-score	Not considered for future data augmentations
2	Random Resize and Crop	0.8	Increase in f1-score	Considered for future data augmentations
2	Random Horizontal Flip	0.8	Increase in f1-score	Considered for future data augmentations
2	Random Vertical Flip	0.8	Increase in f1-score	Considered for future data augmentations
2	Normalize	0.8	Increase in f1-score	Considered for future data augmentations
3	Random Rotation	0.8	No change to f1-score	Not considered for future data augmentations
3	Random Erasing	0.8	No change to f1-score	Not considered for future data augmentations
4	Color Jitter	0.78	Decrease in f1-score	Not considered for future data augmentations
4	Random Affine	0.78	Decrease in f1-score	Not considered for future data augmentations

Table 1: Data Augmentation tests on f1-score

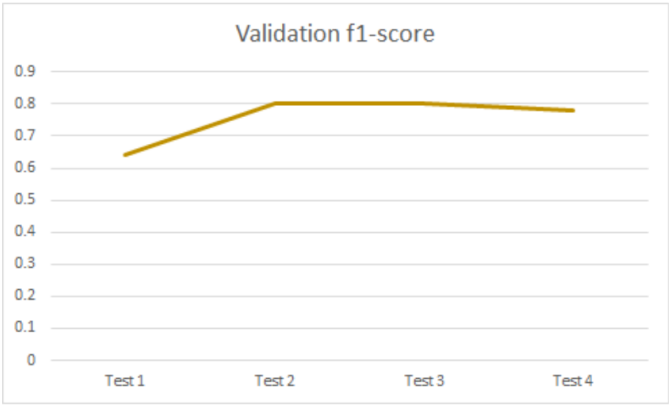

Figure 2: f1-score comparisons between the data augmentation tests

Random Resize and Crop, Random Horizontal Flip, Random Vertical Flip, Normalize really helped during training and hence they were considered. Random rotation and erasing did not have much effect on the f1-score but the training time was increased and hence were not considered. It was also clear that adding too many augmentations is affecting the performance negatively making it difficult for the model to identify important features.

4.1.3 Epochs

Epochs are tied together with the Early Stopping regularisation technique. During running of the model, it was realised that each epoch took approximately 400 seconds which accumulates drastically when the number of epochs increases. Therefore, around 15 epochs were tested.

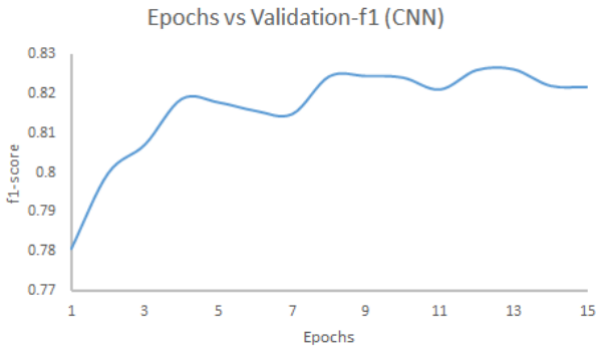

Figure 3: CNN f1-scores for Epochs

The epochs were a very limiting factor to this network because it negatively affected the execution time as it scaled up. The number of epochs for this component were constrained to 15, which still produced adequate results and is still relatively higher in comparison to earlier epoch accuracies.

4.1.4 Batch Size

A popular sequence of batch size values were used starting from 16 to 256. There is a trade-off between the number of iterations and the batch-size. Smaller batches call for less computations and faster execution.

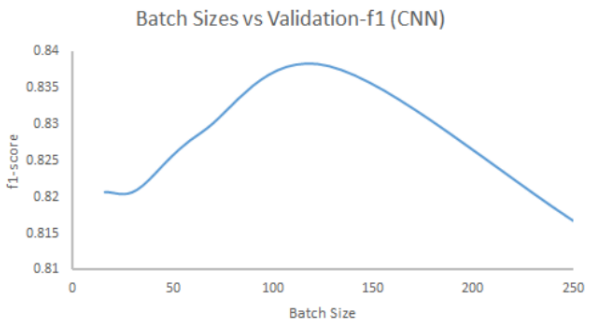

Figure 4: CNN f1-scores for different batch sizes

In Figure 4, batch size of 128 is the most suitable. It has a right amount of generalisation ability where less batches generalise more because of fewer computations whilst more batches over-fit to the data set.

4.1.5 Learning Rate

Generally, a learning rate of about 0.0001 to 0.01 for Adam optimiser is utilised, so the values tested were based around this [21].

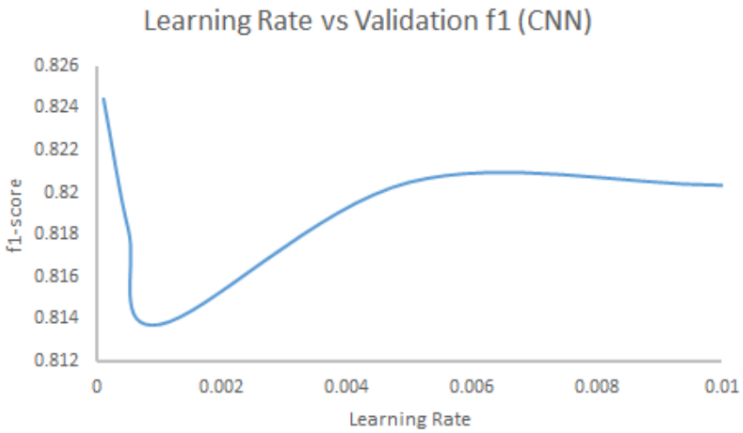

Figure 5: CNN f1-scores for different learning rates

In figure 5, the most optimal learning rate is 0.001 which agrees with literature. Not only does this give the best accuracy but also low training

time. It is the "sweet spot" in which it is not too low to converge but not too high to lead to overshooting the minima.

4.2 Bi-LSTM Network

4.2.1 Embedding Size

The following embedding sizes were used: 25, 50 and 100. These were examined against the validation f1 of the classifier.

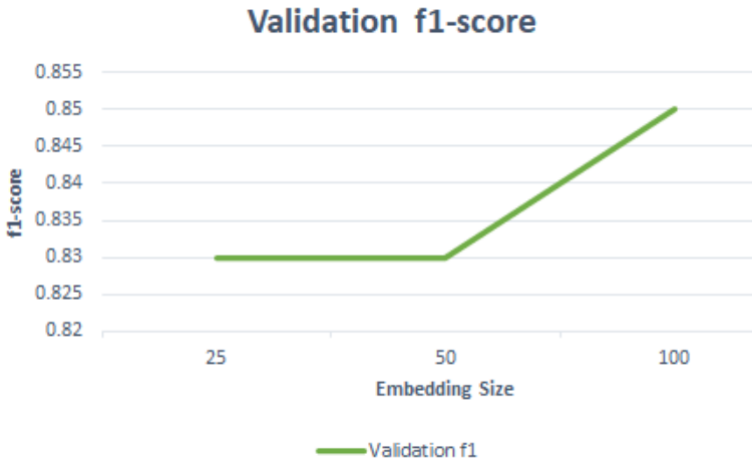

Figure 6: Bi-LSTM f1-scores for different embedding sizes

As depicted in plot 6, the best embedding size to choose was 100 as the validation f1-score was high and the validation loss was low. It can be extrapolated that high embedding size enabled better representation of the features and hence the better f1.

4.2.2 Gensim Data Set

The pre-trained Gensim Data set was leveraged for the use of Word2Vec embeddings. Gensim encompasses many data sets where; glove-twitter, glove-wiki and word2vec-google-news were tested to create word embeddings of the words. For our given dataset glove-twitter had the embedding for most of the

words where only 452 words did not have embeddings, hence glove-twitter was chosen. This is depicted in figure 7.

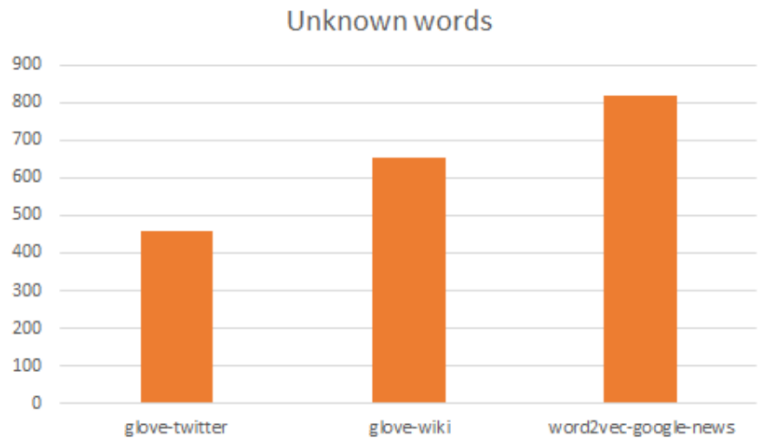

Figure 7: Words without embeddings for Gensim models

4.2.3 Sequence Length

Sequence length refers to the maximum length of words to be kept in a sentence before being passed to the Bi-LSTM model for classification. It is shown in the following histogram and box-plot in figure 8.

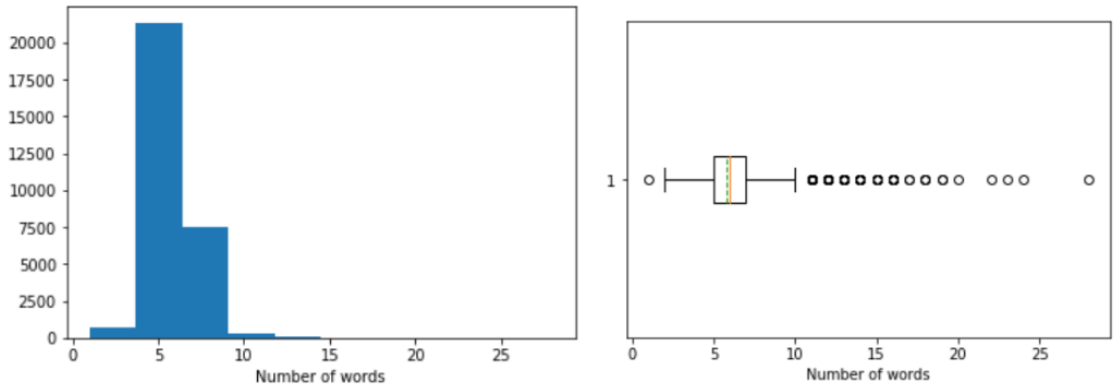

Figure 8: Frequency of words in a sequence

This histogram and box-plot shows the number of words in the input captions. It indicates that most of the captions have on average 7-9 words. The exact values are shown below.

- Number of captions less than 6 words : 13275
- Number of captions less than 7 words : 22015
- Number of captions less than 8 words : 26880
- Number of captions less than 9 words : 28853
- Number of captions less than 10 words : 29530

Extrapolating from this high frequency about these values, sequence lengths of 6,8 and 10 were used to test against validation loss and f1.

Figure 9: Validation loss and f1 for Bi-LSTM after sequence length limiting

It can be summarised from the results in figure 9 that sequence length of 8 had the best f1-score and least amount of loss. Fewer captions with empty padding resulted in better results as the features were not diluted in the process. Sequence length 8 was then chosen.

4.2.4 Epochs

To gain a better gist of how the model performs over time, a large epoch of 200 was chosen, shown in figure 10.

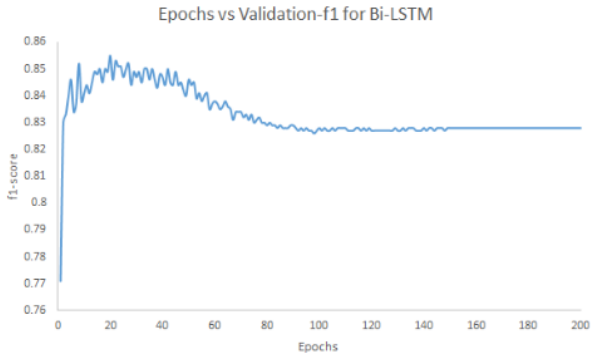

Figure 10: Bi-LSTM f1-scores for Epochs

There is an increase in f1-score at an epoch value of around 20, however, the curve is extremely noisy and not reliable. A plateau is observed to occur around 100 which completely stabilises at 150. Therefore, 150 epochs were used as going beyond this value made no improvement in accuracy. This was to ensure consistent results, even though there is a small sacrifice in accuracy.

4.2.5 Batch Size

It was assumed that batch sizes similar to section 4.1 would be beneficial, however, it was found that multiples of 50 worked better. This reflects that values in literature don't always need to be implemented in reality and there is no "one size fits all" for hyper-parameters in networks. The results for the effect of batch sizes on accuracy is shown in Figure 11.

The most optimal batch size is 150. 128 was tested, however, showed a decrease in results as mentioned before. Again, this batch size generalises the data well enough not to lose important information and avoids over-fitting.

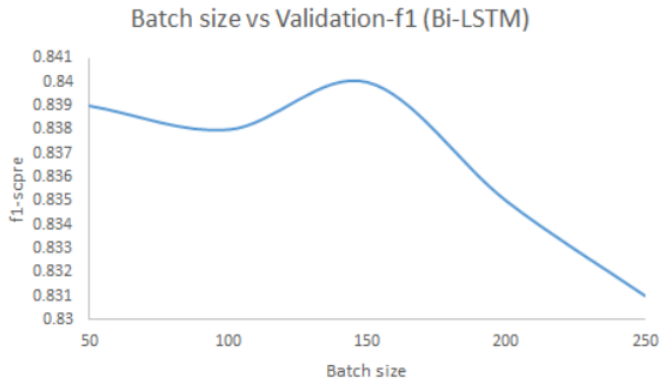

Figure 11: Bi-LSTM f1-scores for Batch sizes

4.2.6 Learning Rate

The learning rate hyper-parameter of the Bi-LSTM network was first tested with 0.01 for fast convergence. It was noticed that higher learning rates than the CNN component performed better and is depicted in the plot below in Figure 12.

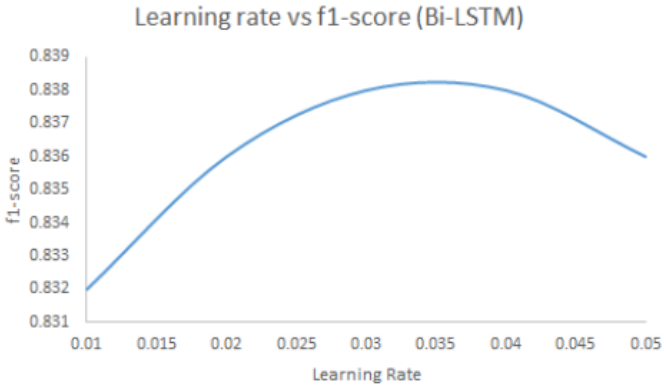

Figure 12: Bi-LSTM f1-scores for Batch sizes

The graph has a maximum accuracy about 0.03 which is chosen to be the most optimal learning rate for this part of the model. Below this, particularly values lower than 0.01 showed a noticeable decrease in accuracy because the network converges too slowly.

4.3 Combined Neural Network

4.3.1 Epochs

The epochs across this component could be large as it was fast in execution. Similar to the Bi-LSTM epochs, 200 was used to monitor the performance.

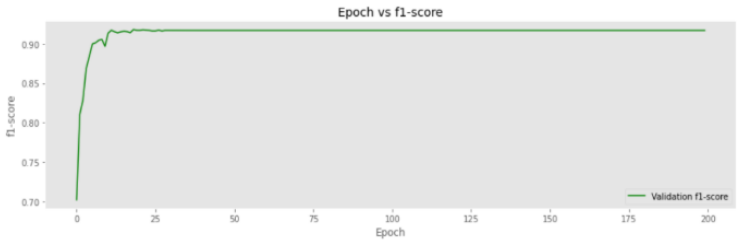

Figure 13: Combined NN f1-scores for Epochs

As shown in Figure 13, the accuracy stabilises around 20 epochs but to ensure full consistency, 80 epochs were used as execution was fast anyways.

4.3.2 Batch Size

Since the combined neural network utilises inputs from both the Bi-LSTM and the CNN components, a larger range of batch sizes were experimented.

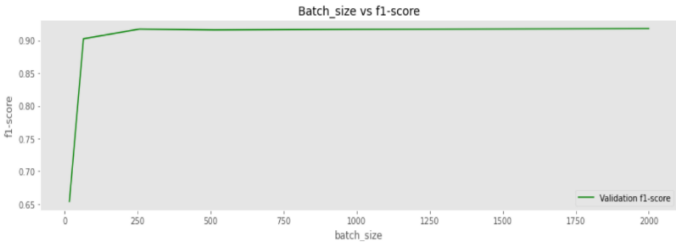

Figure 14: Combined NN f1-scores for different Batch Sizes

It can be concluded from Figure 14 that batch size of 250 was adequate where values beyond this could be used as it had no further effect on the accuracy.

4.3.3 Learning Rate

The final learning rate was again tested with values around 0.01.

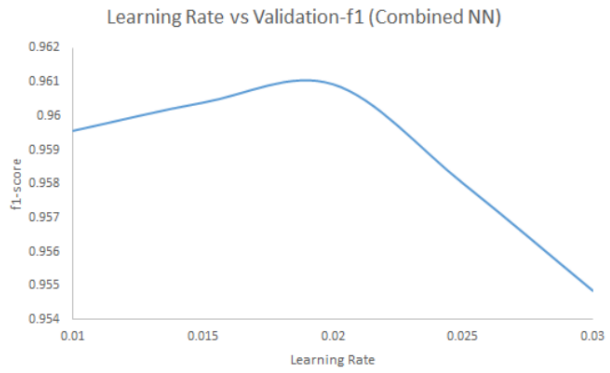

Figure 15: Combined NN f1-scores for different Learning Rates

It is shown above in Figure 15 that 0.02 was the most optimal with the same reasoning as the other sections. This hyper-parameter value was deemed suitable as it had the greatest f1-score.

4.3.4 Drop Out

To determine whether drop out has an effect or not, 3 tests were done where: drop out was not implemented, drop out was used once and drop out was used twice. The results are shown in Figure 16.

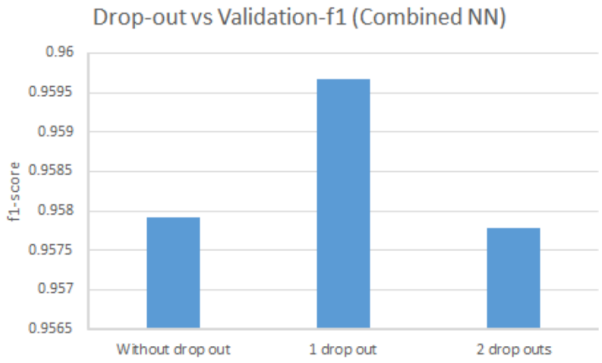

Figure 16: Combined NN f1-scores for dropout ablation study

Whilst ResNet removes the drop-out method, it can be shown that indeed the implementation has a noticeable difference when used once in the network. The reason for this is that it prevents over-fitting. However, when it is used twice, it has most likely dropped too many neurons and hence cannot learn the data set well.

The drop-out hyperparameter was also tested against the f1-score for its accuracy , depicted in the graph below.

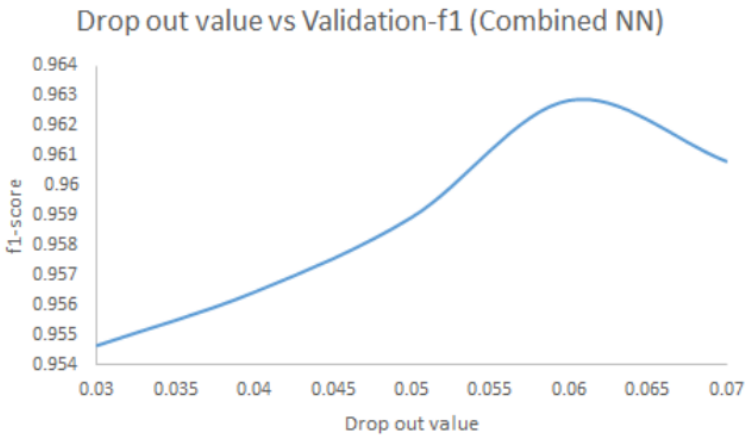

Figure 17: Combined NN f1-scores for different dropout values

Figure 17 above shows a peak in accuracy for $p = 0.06$. It is less than literature values of $p = 0.5$[5] because it is still based off of the ResNet model

which doesn't include drop-out at all. There are a large number of neurons used to process the image and caption features. With more layers and nodes, it is common for networks to over-fit the training data which is why drop out is a good regularisation technique.

4.3.5 Hidden Layer size

The hidden layers reveal how complex the features are and the corresponding f1-score depict how well it performs. A large range was again used to pinpoint the optimal layer size from 16 to 2000.

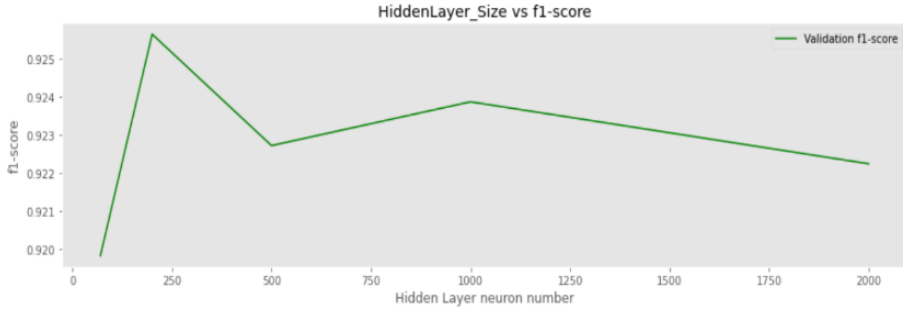

Figure 18: Combined NN f1-scores for different Hidden layer sizes

Whilst there is a noticeable peak in accuracy at a hidden layer value about 250 for Figure 18, the actual differences in f1-score is minimal, with the values between 0.001 to 0.002 of each other. Therefore, little impact was observed using more hidden layers.

5 Discussion and Conclusion

Upon analysis of the results collected in the experimentation section, it is clear that changing certain modules and considering various features from captions, does indeed have an influence on the accuracy. These all relate to their formulas and the theory behind the modules, whether it's for optimisation, augmentation or regularisation purposes. We agree on the below points based on our experiements:

1. It is very important to explore the data to understand how to extract maximum features for instance in this case, it was essential to extract features from image and caption for better results.

2. Normalizing R,G,B channels of the image data helped in attaining 0.8 f1 faster thereby reducing the training time.

3. It is very important to examine the data in this case class distribution to understand if the data is skewed and take necessary steps during training.

4. Using weighted loss in Binary Cross Entropy helped in penalizing the minor labels and thereby achieved better results

5. Early stopping is a powerful technique to prevent overfitting, we can always use modified versions of it like storing the best weights once the best f1 score of validation data set is attained.

6. Data augmentation helped in achieving slightly better results but it is important to test which augmentations help for our dataset.

7. Using too many augmentations is not a good idea as it is affecting the performance negatively making it difficult for the model to identify important features

8. Leveraging transfer learning saves a lot of time, so that we can focus on other techniques for optimization.

9. ResNet50 is a very stable architecture as it is faster than WideNet, ResNet 150 and is also compact in size. These good things make it an easy choice for beginners to improve their skills.

10. In most cases having more epochs does not increase the accuracy much and results in loss of computation and time hence it is vital to understand the training process to set the epochs.

11. We should try a couple of learning rates to understand how the loss and accuracy are affected and then choose one which leads to a reasonable increase of accuracy.

12. Pytorch provides some additional optimization techniques like gradient clipping, weight decay in optimization which helps the training in getting off the edge to perform slightly better.

13. Adam optimizer really helps in the optimal minima of the loss during training as it is a combination of all the good things i.e. SGD, Momentum and RMSProp.

14. Adding additional hidden layers and neurons might slightly add time for performing calculations without providing any better results. In most cases 1-2 layers are all that is required.

15. Adding too many hidden layers might over complicate the network.

16. ReLu causes dead neurons and hence your network might stop to learn after a while. Every activation function has its upsides and downsides and hence the right combination has to be chosen for the problem at hand.

17. Using mini batch saves a lot of time as the network does not perform back propagation for every data point, but this also means the network might not converge to local minima, hence our batch size should not be too large that it might miss local minima and not too small that it takes a lot of time performing back propagations.

18. Drop out introduces regularization into the neural network by forcing other neurons to be independent. High drop out value is not a good idea as all it might make most of the network inactive.

19. Introducing drop out in most or all hidden layers will also make most of the network inactive.

20. Batch Normalization helps in attaining better results mostly in CNN as there are a lot of parameters in a CNN.

21. Bi-LSTM allows the networks to have both backward and forward information about the sequence at every time step. This comes in handy when we want our features to have memory during the training process, like training on sentences. Secondly Bi-LSTM are faster than RNNs.

22. It is a mandatory step to clean unstructured data like sentences by using off the shelf packages like nltk prior to training.

23. Gensim offers pre-trained word2vec embedding by directly using them we can reduce all the training time required for creating word2vec embeddings.

24. The length of sequence (i.e. caption) to be used for training also plays an important role in training and impact the results and hence it is vital to examine the distribution of sequence length to identify the maximum occurring length as optimal length.

The final architecture can be summarised in the appendix 6.1.

References

[1] K. Smelyakov A. Chupryna A. Arsenov, I. Ruban. Evolution of convolutional neural network architecture in image classification problems. *Kharkiv National University of Radio Electronics*, 2019.

[2] P. Konig A.H. Garcia. Further advantages of data augmentation on convolutional neural networks. *Cornell University*, 2019.

[3] Pierre Baldi and Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *Journal of Machine Learning Research*, 4(Sep):575–602, 2003.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[5] J. Brownlee. A gentle introduction to dropout for regularizing deep neural networks. *Machine Learning Mastery*, 2018.

[6] J. Brownlee. A gentle introduction to the rectified linear unit (relu). *Machine Learning Mastery*, 2019.

[7] J. Brownlee. How to control the stability of training neural networks with the batch size. *Machine Learning Mastery*, 2019.

[8] V. Bushaev. Adam - latest trends in deep learning optimization. *Towards Data Science*, 2018.

[9] J. Collis. Glossary of deep learning: Batch normalisation. *Medium*, 2017.

[10] A. Rozza F. Manessi. Learning combinations of activation functions. *International Conference on Pattern Recognition*, 2018.

[11] R. Gomez. Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names. *Raul Gomez Blog*, 2018.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[13] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] G. Hever. Sentiment analysis with pytorch — part 4 — lstmmodel. *Medium*, 2020.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3367–3375, 2015.

[20] D. Liu. A practical guide to relu. *Medium*, 2017.

[21] D. Mack. How to pick the best learning rate for your machine learning project. *Medium*, 2018.

[22] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[23] Prabhu. Understanding of convolutional neural networks (cnn). *Medium*, 2018.

[24] N. Rahman. What is the benefit of using average pooling rather than max pooling? *Quora*, 2017.

[25] S. Saha. A comprehensive guide to convolutional neural networks. *Towards Data Science*, 2018.

[26] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[27] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[30] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[31] Wikipedia. Logistic function. https://en.wikipedia.org/wiki/Logistic_function#Neural_networks, 2020.

[32] Wikipedia. Word2vec. https://en.wikipedia.org/wiki/Word2vec, 2020.

[33] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

6 Appendix

6.1 Final Architecture

We find the following neural network architecture reasonable for the problem data set. We will need 3 different neural networks:

1. CNN model for extracting features from image
2. Bi-LSTM model for extracting features from captions
3. Combined (vanilla) neural network for using features extracted from the above networks and performing the final classification

6.1.1 First Network - CNN

- Data Augmentation - Resize and Crop, Random Horizontal Flip, Random Vertical Flip.
- Pretrained CNN Model - ResNet50
- CNN Activation Function - Change from Relu to LeakyRelu
- FC layer:
  - Activation Function - Sigmoid
  - Output Neurons - 18 (class labels)
- Mini batch size - 128
- Epochs - 15
- Learning Rate - 0.0001
- Optimizer - Adam
- Regularization - Early stopping
- Loss Function - Binary Cross Entropy

6.1.2 Second Network - Bi-LSTM

- Preprocessing - Case folding, remove punctuation marks, digits, stop words, tokenize, lemmatize
- Word2Vec
  - Dataset - glove-twitter-100
  - Embedding Size - 100
- Sequence Length - 8
- Layer 1 - Bi-LSTM
  - Dropout - 0.1
  - Input Neurons = 100
  - Output Neurons = 50
- Layer 2 - Linear
  - Input Neurons = 100
  - Output Neurons = 18 (class labels)
- Activation Function - Sigmoid
- Mini batch size - 250
- Epochs - 100
- Learning Rate - 0.02
- Optimizer - Adam with weight decay 1e-5
- Regularisation
  - Early Stopping
  - Learning Rate Scheduling - StepLR with gamma = 0.95
  - Gradient Clipping with max norm = 0.5
- Loss Function - Binary Cross Entropy with weighted loss

  - weights = [0.4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

Once both these models are trained we will extract features for our entire dataset in evaluation mode. We will then concatenate both these features to pass as inputs for the next network.

6.1.3 Third Network - Combined

- Layer 1 -
  - Input - 2148 neurons (i.e. length of each combined features - 2048 + 100)
  - Output - 2000 neurons
  - Activation - Relu
  - Dropout - 0.05
- Layer 2 -
  - Input - 2000 neurons
  - Output - 18 neurons (unique class labels)
  - Activation - Sigmoid
- Mini batch size - 1000
- Epochs - 80
- Learning Rate - 0.02
- Optimizer - Adam
-
  - Early stopping
  - Learning Rate Scheduling - Reduce when validation loss does not improve for 2 times (patience = 2)
  - Gradient Clipping with max-norm = 0.5
- Loss Function - Binary Cross Entropy with weighted loss
  - weights = [0.5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

6.1.4 Final Results

- Validation loss: 0.043
- Validation f1-score: 0.927
- Test f1-score (from Kaggle): 0.877

The total performance encapsulated as f1-scores and training/validation loss is summarised in the below plots.

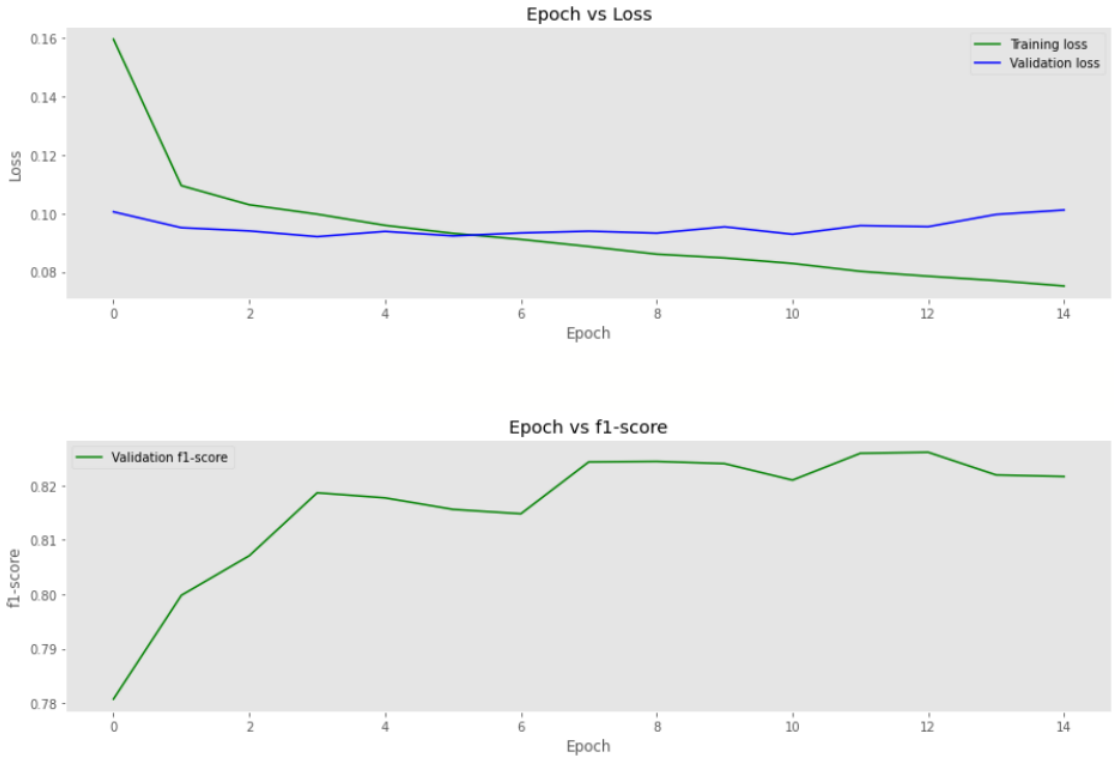

Figure 19: Optimal CNN performance over 14 epochs

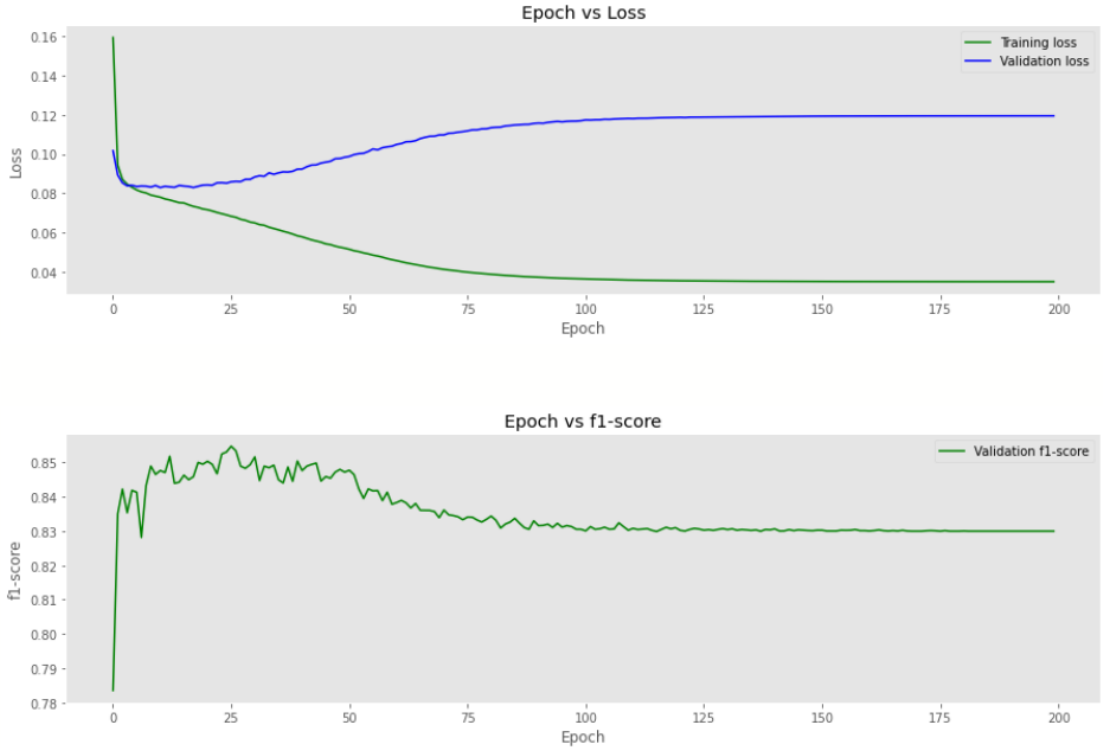

Figure 20: Optimal Bi-LSTM performance over 14 epochs

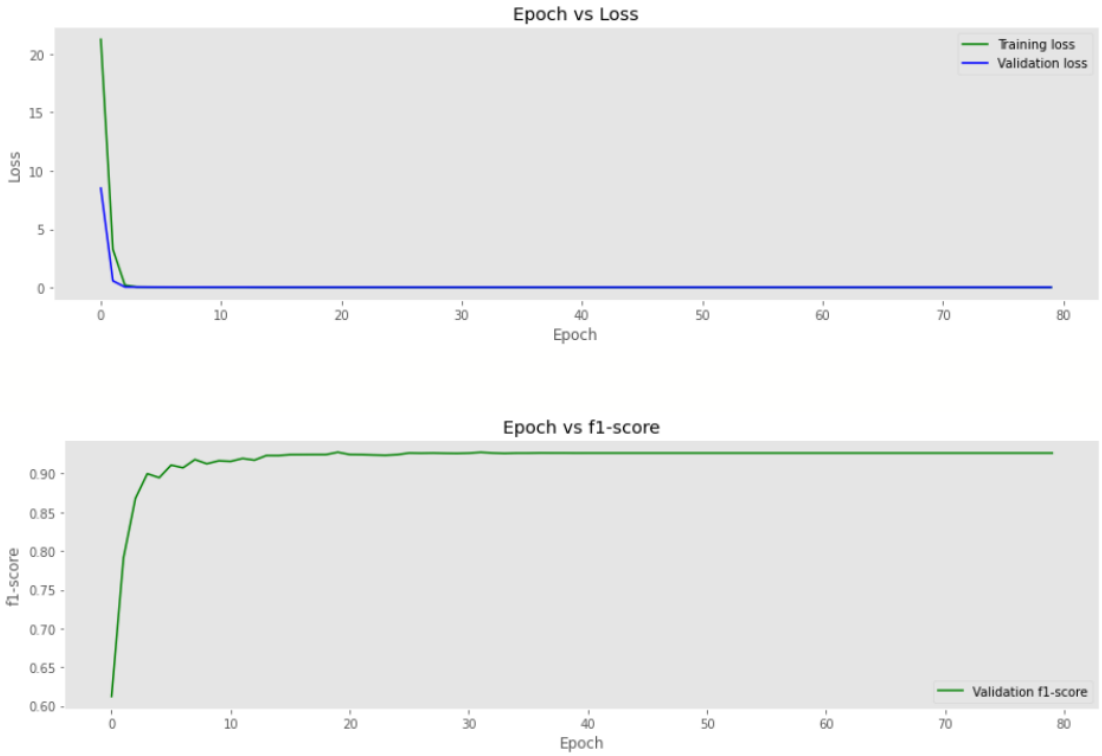

Figure 21: Optimal Combined NN performance over 14 epochs

6.2 Hardware and Software Specifications

The code was run on Google Colab which utilises:

- Processor: 2-core Intel(R) Xeon(R) CPU @ 2.20GHz
- CPU count: 2
- 32 RAM
- 72 HDD

6.3 Running the Code

1. The Google Colab version of the code can be found here - [Click for Google Colab Link]
2. Place the following files in **Code/input**:
   - **train.csv** - training data set
   - **test.csv** - testing data set
   - **2020s1comp5329assignment2.zip** - all images in a zip file

Figure 22: Input files

3. Update PATH_TO_CODE_FOLDER variable in "Inputs and hyper parameters" section.
4. Execute "Mount Google Drive" section, only if this code is executed in Colab.
5. Perform the below steps to train the model:

(a) Run all the cells, except "Load Pretrained Model",there will be 3 of these sections.

- When you reach the *drive.mount('/content/drive/'))*, you'll be redirected to another browser with the authorization code. Copy and paste this into the google colab input box.

Figure 23: Google Authorization

(b) This will train all 3 networks and the predictions will be saved in **Code/output** folder

6. Perform the below steps to test the model:

(a) Please download the models from below links and save to **Code/algorithm** folder.

- Image Classification Model - [Click here for link]
- Caption Classification Model - [Click here for link]
- Combined Classification Model - [Click here for link]

(b) Run all the cells except "Training Model", there will be 3 of these sections.

(c) This will load pre-trained models, generate predictions and save them in **Code/output** folder