Deep Learning Assignment 2

Group Members:

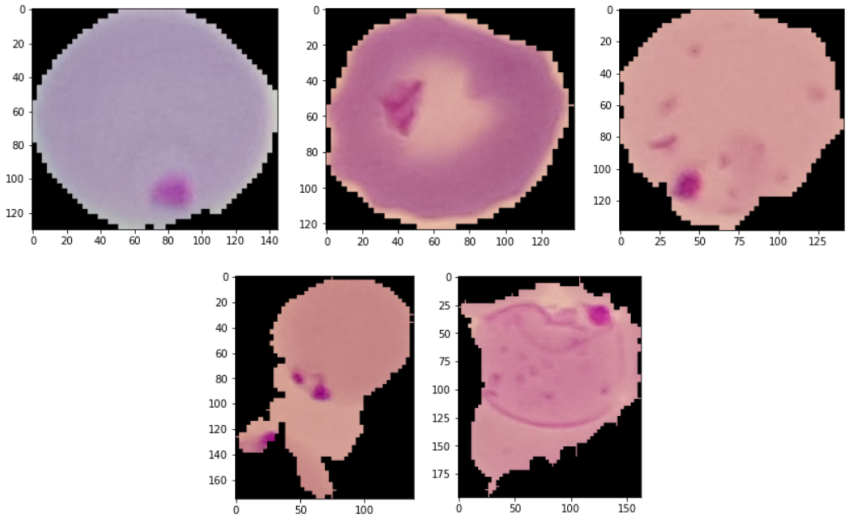
1. Devansh Shukla (MT21023)
2. Siruvuri Karan Raju (MT21094)

Part 1:

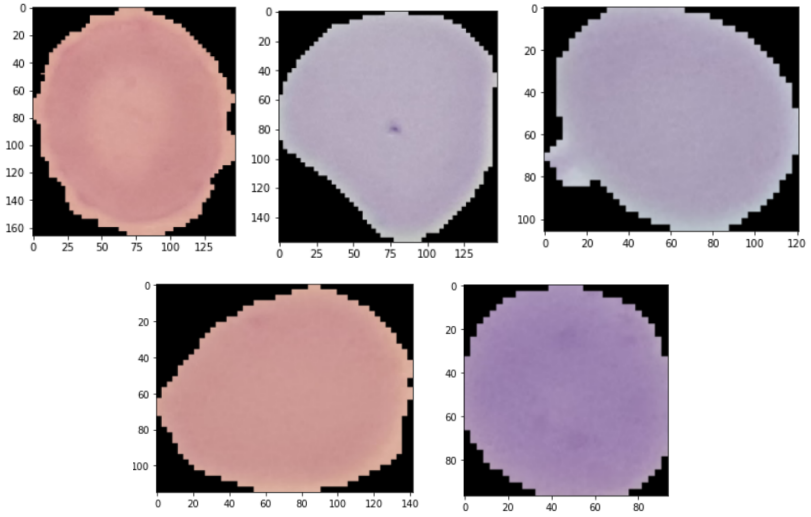
Dataset: The dataset consists of 27,558 cell images with equal instances of parasitised and uninfected cells.

Q1. Visualisation of 5 random images from both classes can be seen below:

Parasitised:



Uninfected:



It can be seen that all images are of varying sizes. Therefore, the images need to be resized to a standard size.

Q2. Pre-processing:

The steps that we have performed on our dataset are:

Resize (size = (128, 128), interpolation=nearest),
ToTensor( ),
Normalize (mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

After the above steps, the dataset is split into Train, Val, and Test in an 80:10:10 ratio.

Assumptions:

A size of (128,128) will be sufficient to capture enough information from our images. The number of epochs and learning rate has been kept the same across all models for consistency and better training times, with n_epochs being 30 and learning_rate being 0.0001. The optimizer function has been kept as Adam.

Helper Functions:

- **DataLoader( ):** This PyTorch function lets us load the data in batches.
- **Evaluate( ):** This is used to evaluate our model over the validation or test set.
- **Fit( ):** This function is used to fit the training data on our model over multiple epochs. This is also where we perform regularisation by sending the value as a parameter and adding it to our loss before sending it backwards. It returns the training and validation losses and scores over each epoch.
- **To_device( ):** It is used to send the data to our device.
- **Accuracy( ):** Returns the accuracy for the actual and predicted labels passed to it.
- **Plot_losses( ):** Function for plotting the train and validation losses and saving their figures.
- **Plot_accuracies( ):** Function for plotting the train and validation accuracies and saving their figures.

Methodology:

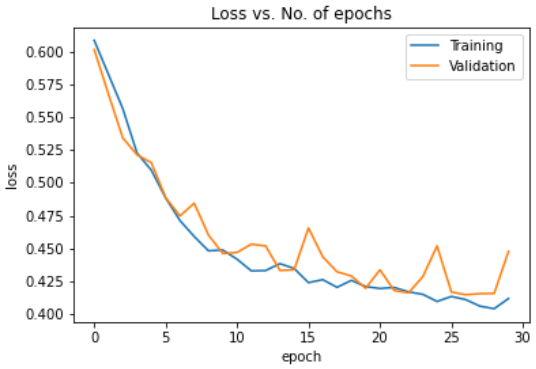
The architecture that we have used for our CNN model is as follows:

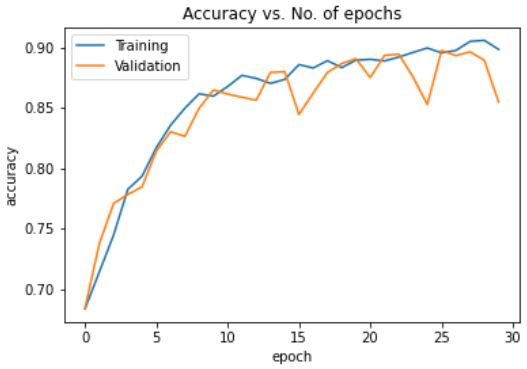
CNNModel(
(conv_a): Conv2d(3, 16, kernel_size=(9, 9), stride=(1, 1), padding=(2, 2))
(pool_a): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
(conv_b): Conv2d(16, 32, kernel_size=(6, 6), stride=(1, 1), padding=(2, 2))
(pool_b): MaxPool2d(kernel_size=2, stride=2, padding=1, dilation=1)
(conv_c): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(flatten): Flatten(start_dim=1, end_dim=-1)
(fcl): Linear(in_features=61504, out_features=2, bias=True)
~~(drop): Dropout(p=0.5, inplace=False)~~
(sgd): Sigmoid()
)

The in_features value for the linear layer is calculated using the formula:

((N + 2P – K)/S) + 1

Result:






```
Train Accuracy: 89.8392
Validation Accuracy: 85.4866
Test Accuracy: 87.2822
```

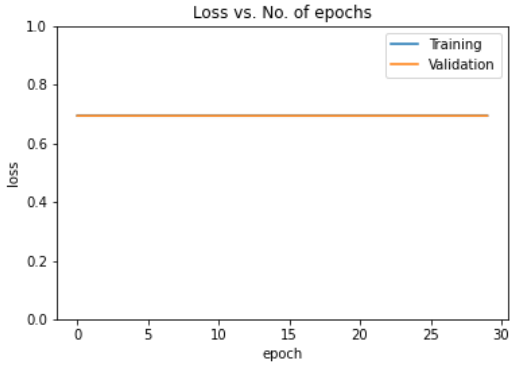
Q3.

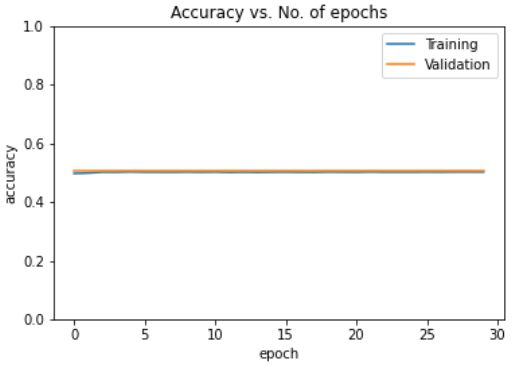
a) Zero Initialization

Methodology:

Weights have been initialised to zero for the convolution layers and the fully connected layer using torch.nn.init.zeros_.

Result:






```
Train Accuracy: 50.1761
Validation Accuracy: 50.4828
Test Accuracy: 48.2676
```

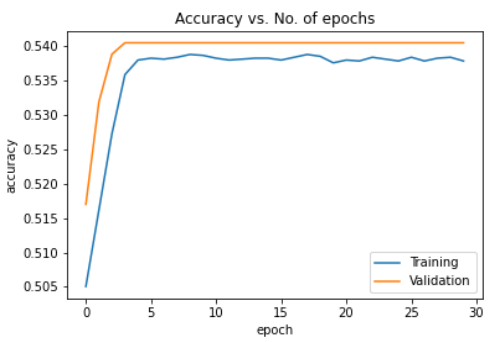
Setting the weights to zero deactivated the layers, and our model could not learn anything, which resulted in constant loss over all the epochs.

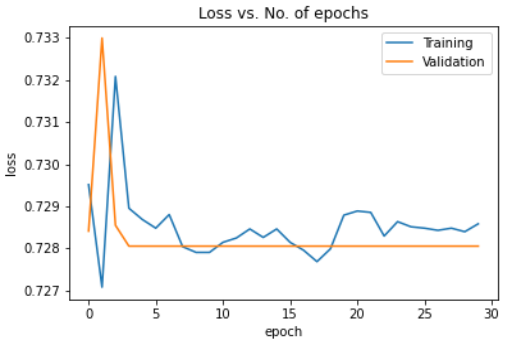
b) Random Initialization

Methodology:

Weights have been initialised to random for the convolution layers and the fully connected layer using torch.nn.init.normal_.

Result:






```
Train Accuracy: 53.7888
Validation Accuracy: 54.0539
Test Accuracy: 55.8541
```

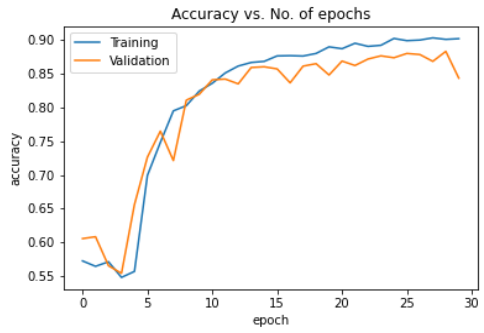
Setting the weights to random resulted in vanishing/exploding gradients causing uneven loss values.

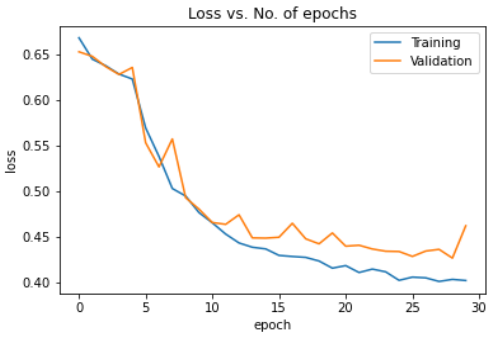
c) He Initialization

Methodology:

Weights have been initialised to He for the convolution layers and the fully connected layer using torch.nn.init.kaiming_normal_.

Result:






```
Train Accuracy: 90.1734
Validation Accuracy: 84.3326
Test Accuracy: 85.3112
```

Using He initialisation results in scaling of weights at every timestep, thus avoiding vanishing or exploding gradients by setting the variance as ‘1’ and making it the most suitable initialisation method for our further experiments.

Q4.

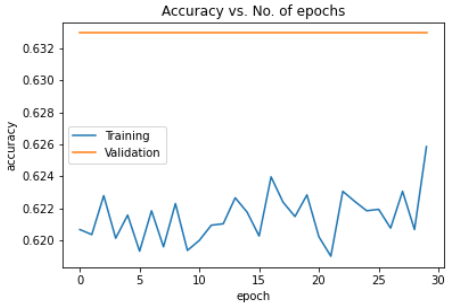
a) After convolutional layers

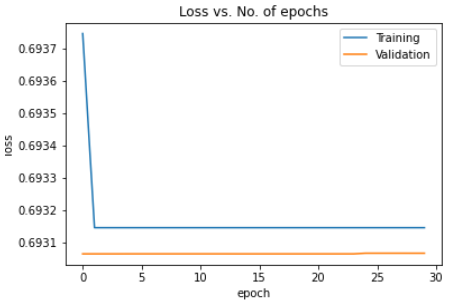
Methodology:

The dropout layers have been added after each convolutional layer, with the probability being passed as a parameter to the init function. The different values for keep probability that we have tested are 0.25, 0.50, and 0.75.

Result:

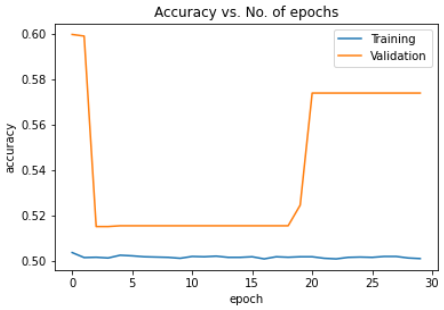
`p = 0.25`

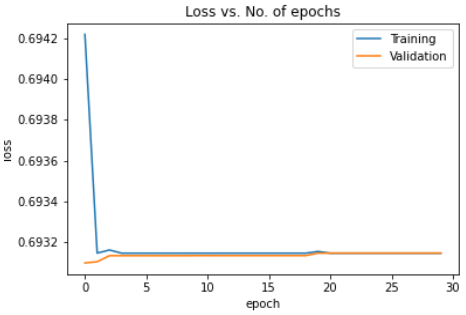





```
Train Accuracy: 62.5858
Validation Accuracy: 63.2960
Test Accuracy: 63.6711
```

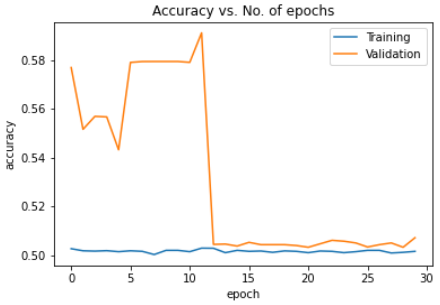
`p = 0.50`

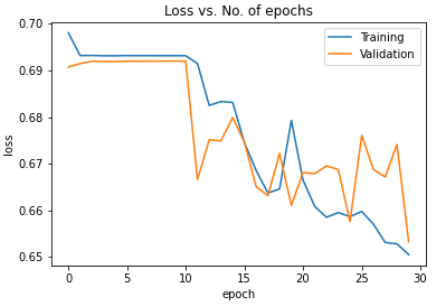





```
Train Accuracy: 50.0948
Validation Accuracy: 57.4120
Test Accuracy: 57.5807
```

`p = 0.75`






```
Train Accuracy: 50.1626
Validation Accuracy: 50.7202
Test Accuracy: 48.4096
```

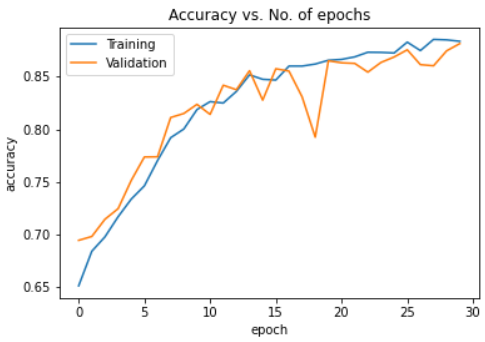
b) Between fully connected layers

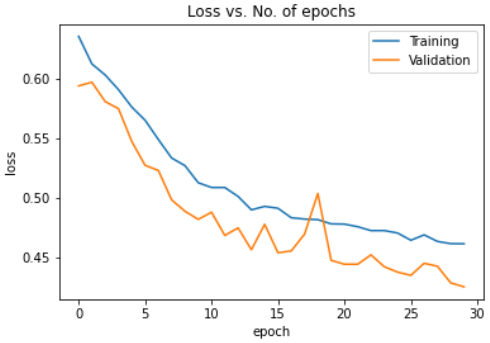
Methodology:

The dropout layers have been added after the linear layer, with the probability being passed as a parameter to the init function. The different values for keep probability that we have tested are 0.25, 0.50, and 0.75.

Result:

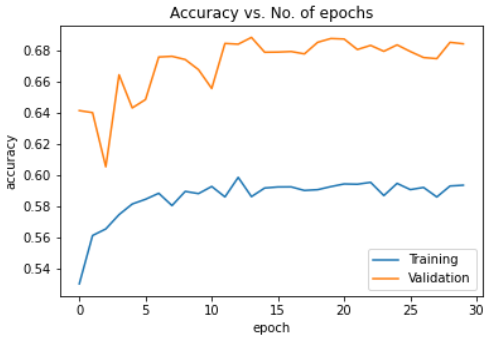
`p = 0.25`

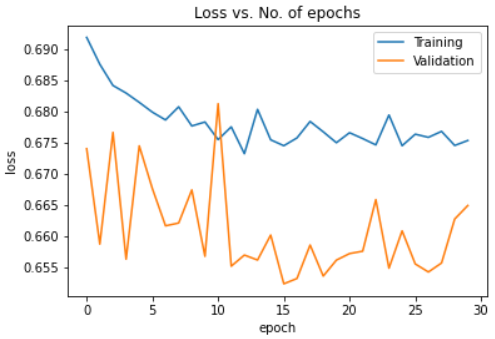





```
Train Accuracy: 88.3851
Validation Accuracy: 88.1678
Test Accuracy: 88.8336
```

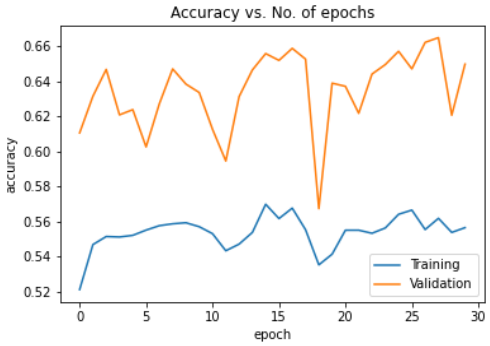
`p = 0.50`

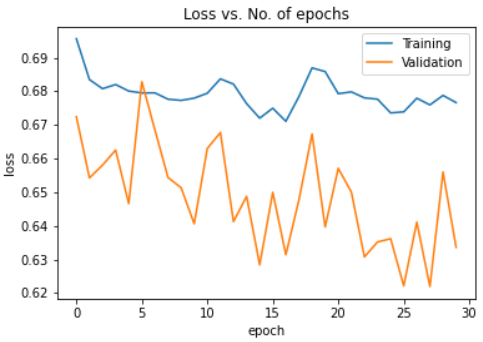





```
Train Accuracy: 59.3615
Validation Accuracy: 68.4741
Test Accuracy: 69.4595
```

`p = 0.75`






```
Train Accuracy: 55.6629
Validation Accuracy: 64.9962
Test Accuracy: 64.3036
```

Analysis:
The above dropout experiments show that applying the dropout layer after the linear layer with p = 0.25 gives us the best results.

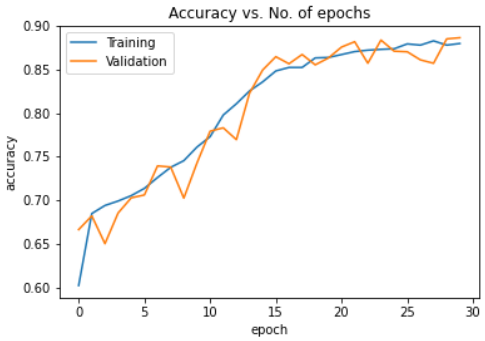
Q5.

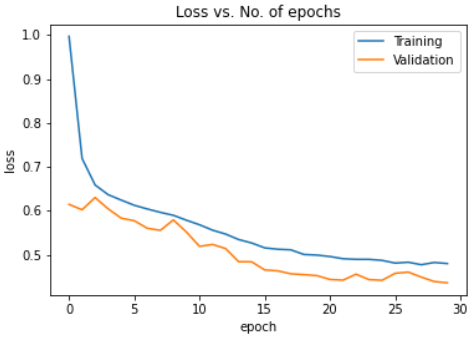
a) L1 Regularization

Methodology:
L1 regularisation has been implemented by adding the L1 norm of weights and bias for the fully connected layer to the loss value. We have implemented L1 regularisation for the following lambda values: 0.001, 0.01, and 0.1.

Result:

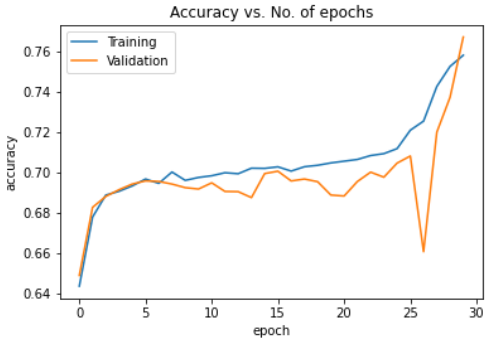
`l = 0.001`

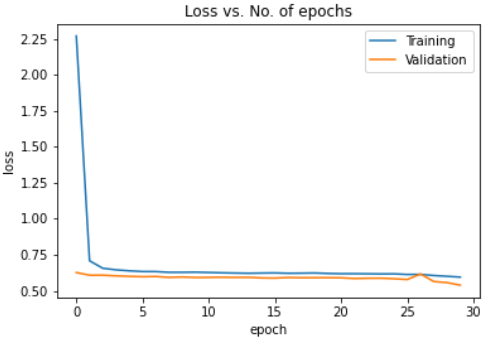





```
Train Accuracy: 87.9606
Validation Accuracy: 88.6251
Test Accuracy: 87.6994
```

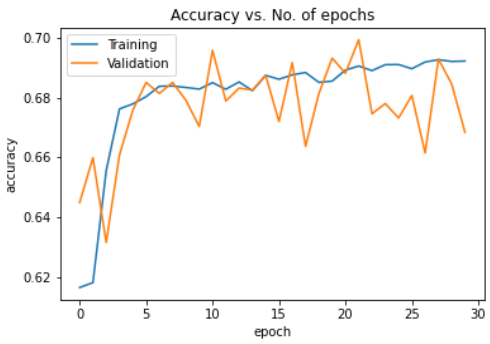
`l = 0.01`

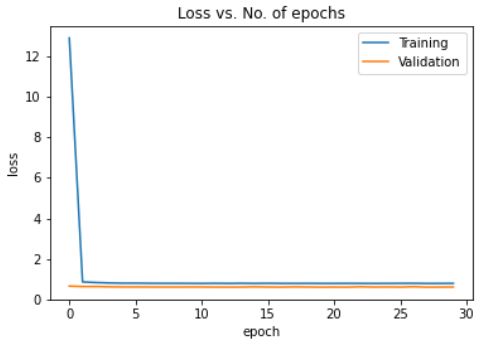





```
Train Accuracy: 75.8354
Validation Accuracy: 76.7395
Test Accuracy: 77.2545
```

`l = 0.1`






```
Train Accuracy: 69.2287
Validation Accuracy: 66.8404
Test Accuracy: 68.0035
```

Analysis:

The best performance for L1 regularisation is observed for lambda = 0.001. It can be seen from the graph that the train and validation loss are very close to each other, resulting in a good performing generalised model for our dataset.

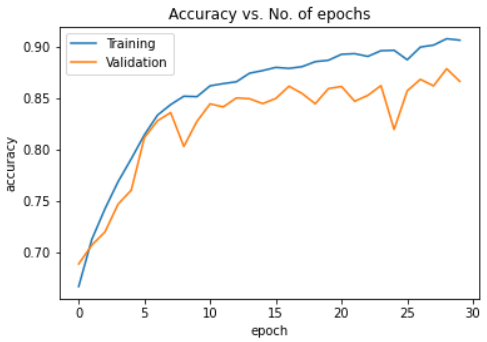
b) L2 Regularization

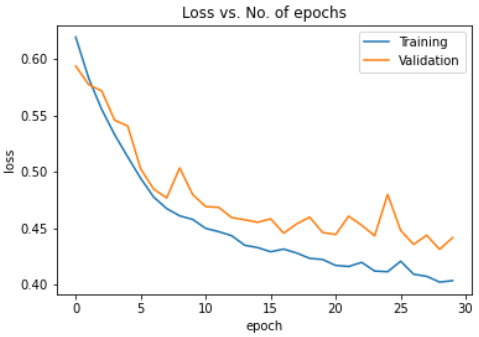
Methodology:

L2 regularisation has been implemented by adding the L2 norm of weights and bias for the fully connected layer to the loss value. We have implemented L2 regularisation for the following lambda values: 0.001, 0.01, and 0.1.

Result:

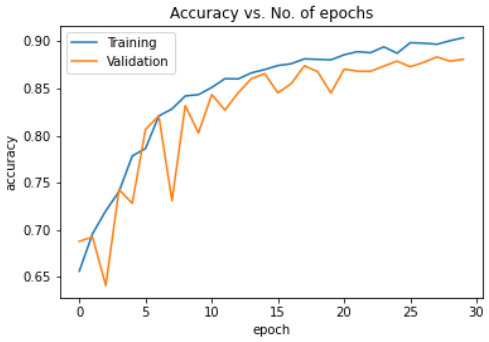
`l = 0.001`

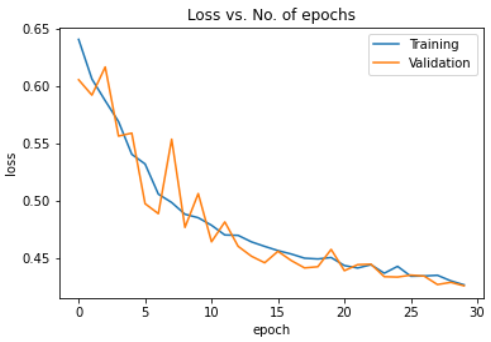





```
Train Accuracy: 90.6927
Validation Accuracy: 86.6652
Test Accuracy: 85.5154
```

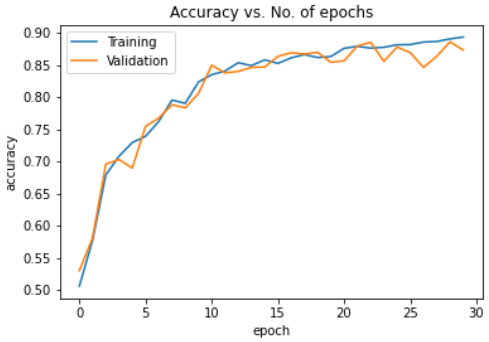
l = 0.01

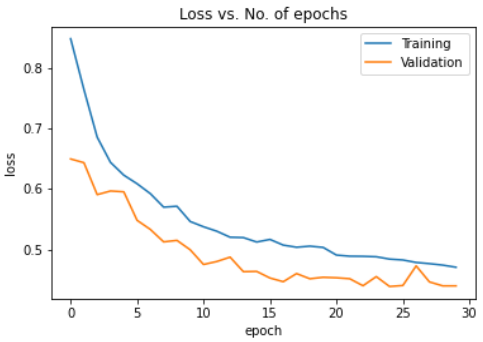





```
Train Accuracy: 90.3992
Validation Accuracy: 88.1190
Test Accuracy: 88.3275
```

l = 0.1



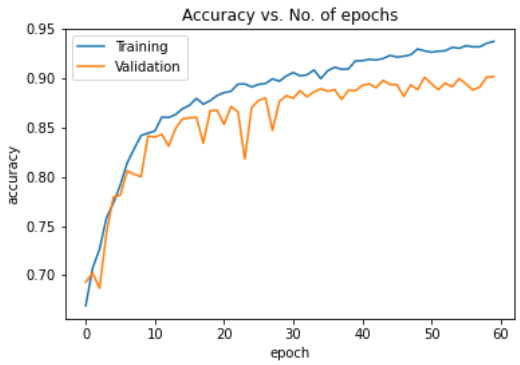


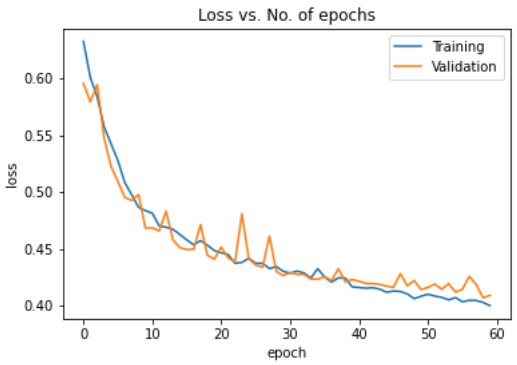

```
Train Accuracy: 89.3244
Validation Accuracy: 87.2911
Test Accuracy: 87.8837
```

Analysis:

We observe the best performance for L2 regularisation with lambda = 0.01. Out of L1 regularisation and L2 regularisation, it can be seen that L2 regularisation is better as we are getting better and more stable results over every value of lambda when compared with L1. The generalisation is as consistent as L1, but we get better performance for the same number of epochs.

Best performing model for higher epochs [60](L2 regularization, lambda = 0.01)






```
Train Accuracy: 93.7635
Validation Accuracy: 90.1743
Test Accuracy: 90.1231
```

Part 2:

Dataset: The Mini Daily Dialog Dataset is a smaller and more processed version of the Daily Dialog dataset defined for NLU tasks containing 700 dialogues in the train.csv file and 100 dialogues in the test.csv file along with the corresponding dialogue acts. There are four dialogue acts in the data and encoded as shown below: inform: 1, question: 2, directive: 3, commissive: 4.

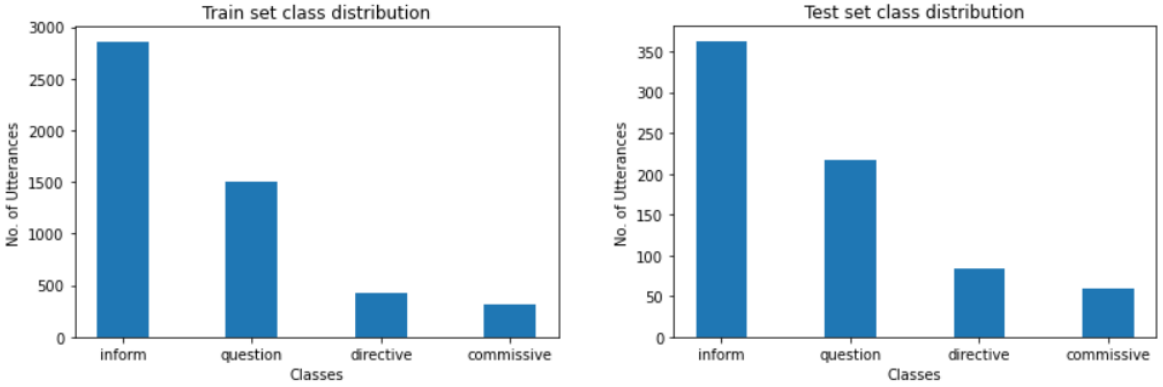
Q1.

Pre-processing:

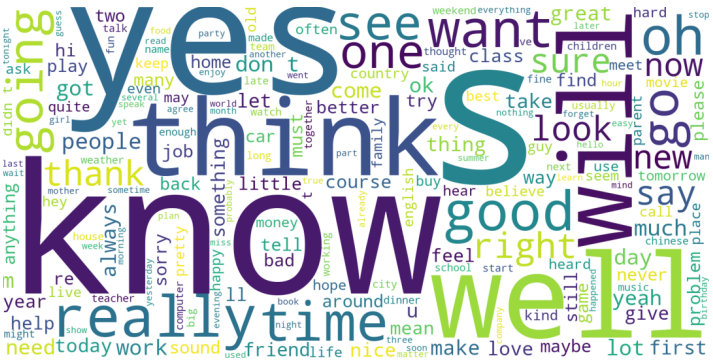
The train and test datasets have been lowered, and the 'act' labels have been adjusted for zero indexing.

Data Visualization:

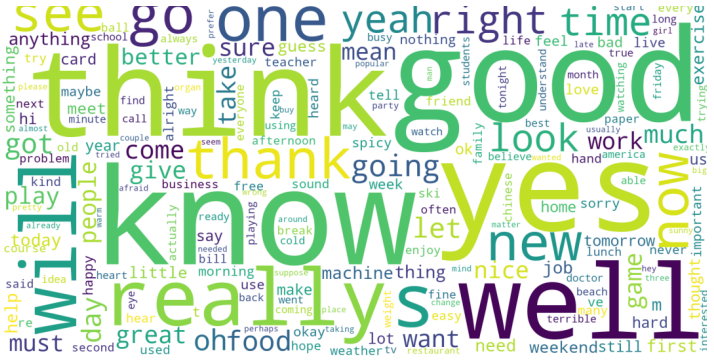
Class Distribution of the given dataset:



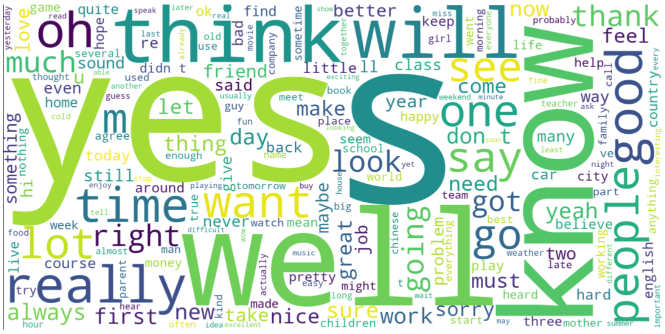
Wordcloud Representation of the training dataset:



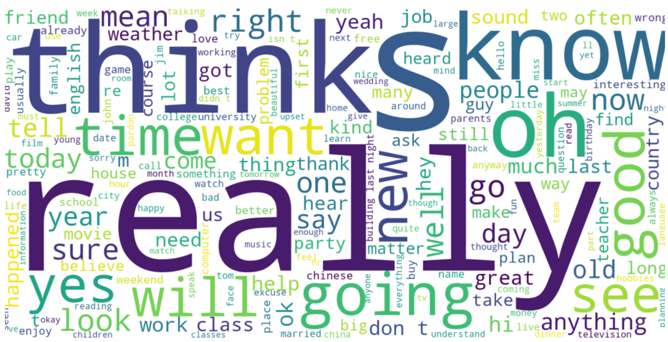
Wordcloud Representation of the test dataset:



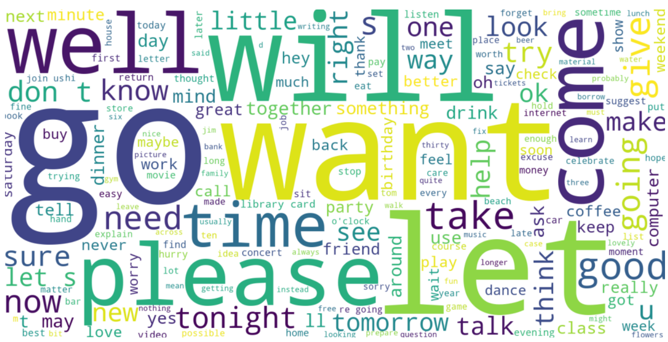
Classwise WordCloud Representation of the given dataset:



Class "inform" in the train set.



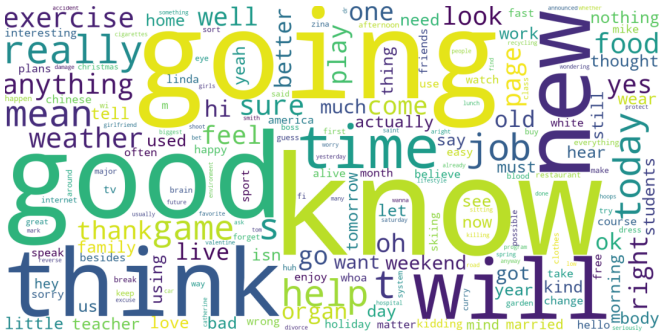
Class "Question" in the train set



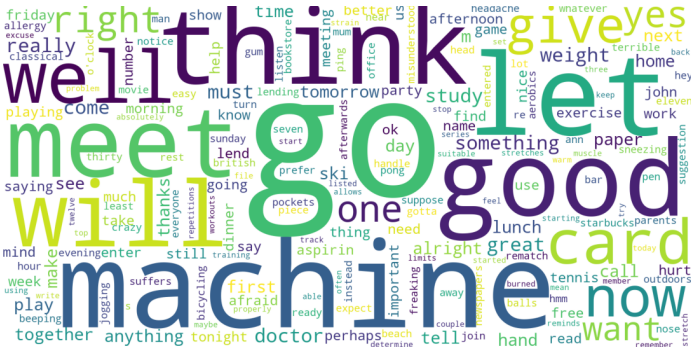
Class "Directive" in the train set

Class "Commissive" in the train set

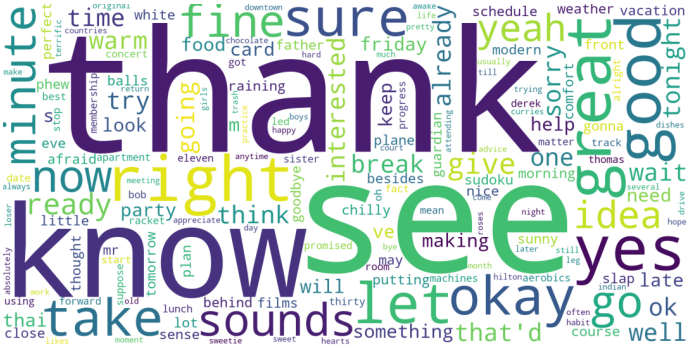
Class “inform” in the test set.



Class “Question” in the test set



Class “Directive” in the test set



Class “Commissive” in the test set

Average characters per utterance in Train set: 62.108840864440076
Average characters per utterance in Test set: 61.58310249307479

Q2. Pre-processing:

The given train dataset has been split into train and validation sets in a ratio of 85:15. The following preprocessing steps have been performed over the train, validation, and test sets:

Lower text, Remove numbers, Remove leading and trailing whitespaces, Fix labels, Remove Special symbols other than [!.,?], Tokenize words, Lemmatization, Make string again and Drop empty rows.

After the above steps, datasets are loaded using PyTorch and prepared for training by adding tokens, ids, and length. These are added so that the dataset could be passed to the DataLoader, making it available for our model to use.

Assumptions:

We tried removing the stopwords and punctuations but noticed that the model performed very poorly over the validation and test sets. Thus, we ended up not doing that. It can be seen that removing these would make our dataset lose valuable information.

The words have been embedded to a dimension of 300, and 128 features in the hidden state have been used. We assume that this will be enough to capture most of the information and not underfit our dataset.

Helper Functions:

- **DataLoader( ):** This PyTorch function lets us load the data in batches.
- **Preprocess_dataframe( ):** This function contains all the preprocessing steps applied to the dataset.
- **Tokenize( ):** this function will tokenise a row of a dataframe using a given tokeniser.
- **Numericalize_data( )**: This function will convert all the tokens into their respective index in the vocabulary.
-
- **Evaluate( ):** This is used to evaluate our model over the validation or test set.
- **Fit( ):** This function is used to fit the training data on our model over multiple epochs. This is also where we perform regularisation by sending the value as a parameter and adding it to our loss before sending it backwards. It returns the training and validation losses and scores over each epoch.
- **To_device( ):** It is used to send the data to our device.
- **Accuracy( ):** Returns the accuracy for the actual and predicted labels passed to it.
- **Plot_losses( ):** Function for plotting the train and validation losses and saving their figures.
- **Plot_accuracies( ):** Function for plotting the train and validation accuracies and saving their figures.
- **Plot_f1( )**: Function for plotting the weighted F1 score of training and the validation set.

Methodology:

```
LSTM_model(
  (embedding): Embedding(1018, 300)
  (lstm): LSTM(300, 128, batch_first=True, bidirectional=True)
  (fc): Linear(in_features=256, out_features=4, bias=True)
  (dropout): Dropout(p=0.25, inplace=False)
)
```

For the most part, the LSTM class utilizes functions similar to the CNN class we defined earlier for Part 1. The text data is first embedded and then passed forward through the above architecture. A probability distribution is obtained by applying sigmoid over the fully connected layer. The loss is propagated backwards through the model, and these operations are repeated for each epoch.

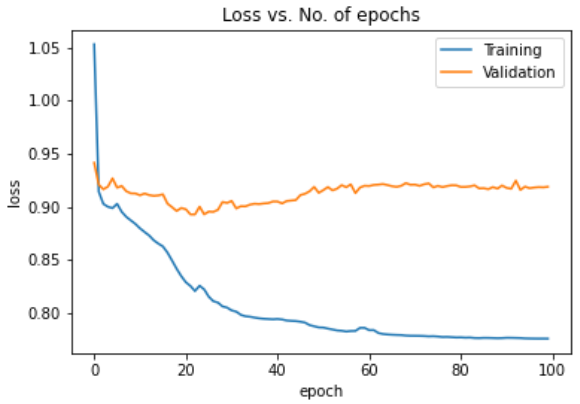
Previous X utterances' context has been considered by concatenating the previous X utterances at the start of the current utterance. This is then passed to our LSTM model as a sequence of tokens that considers the earlier utterances as context.

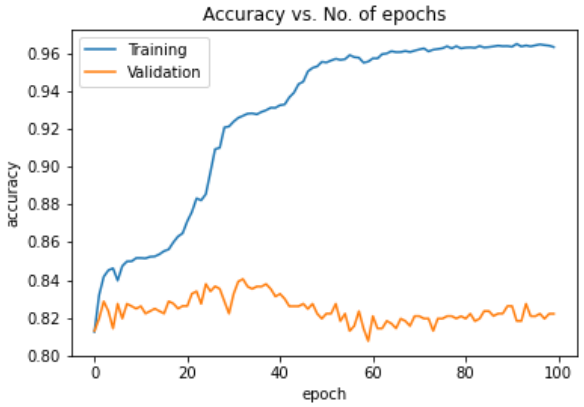
Q3.

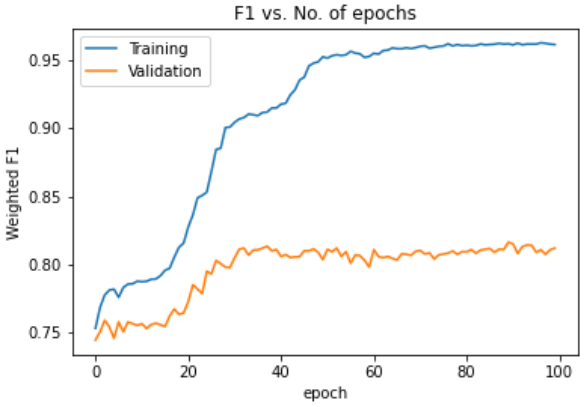
It can be observed from the below experiments for the dropout probability that 0.25 performs the best for our model.

a) X = 0

dropout probability = 0.25

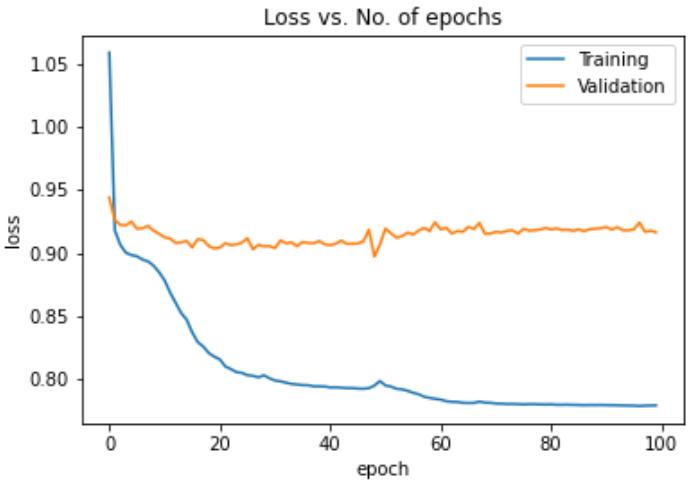


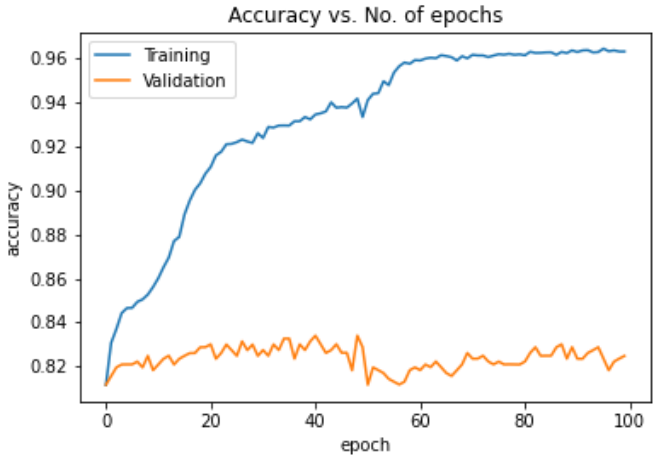


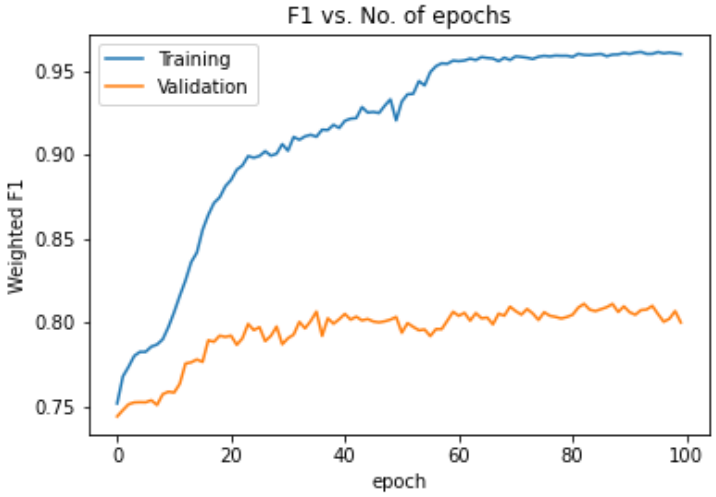



```
Train Accuracy: 96.3402
Validation Accuracy: 82.2161
Test Accuracy: 83.3841
```

dropout probability = 0.50

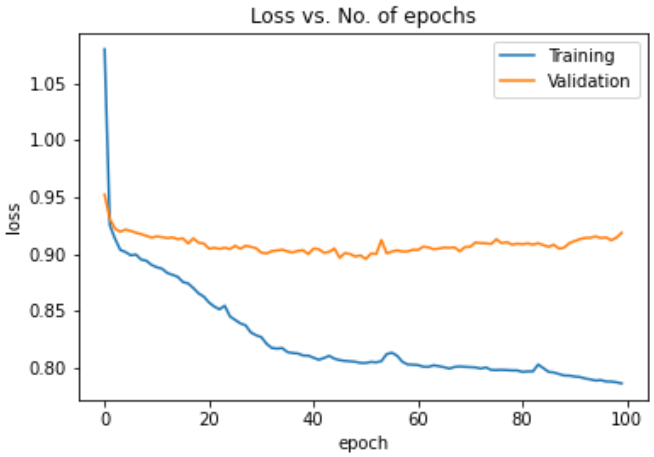


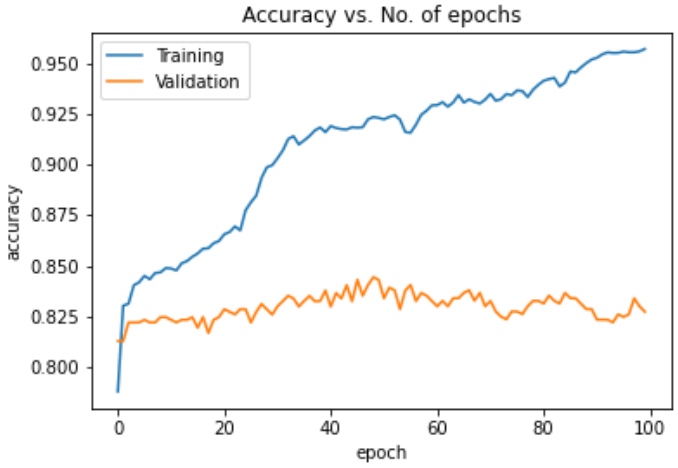


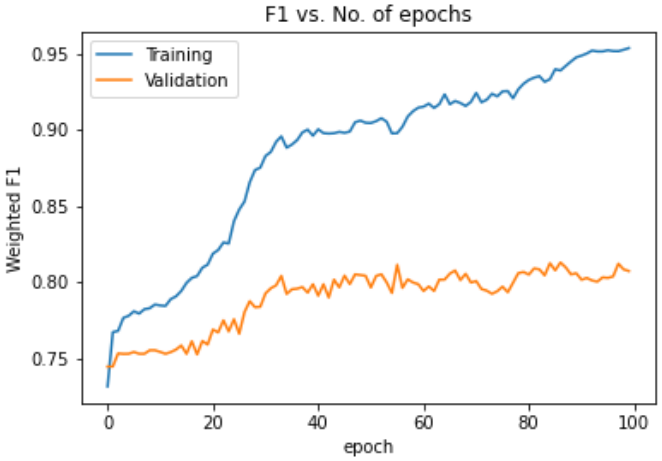



```
Train Accuracy: 96.3235
Validation Accuracy: 82.4807
Test Accuracy: 82.5870
```

dropout probability = 0.75

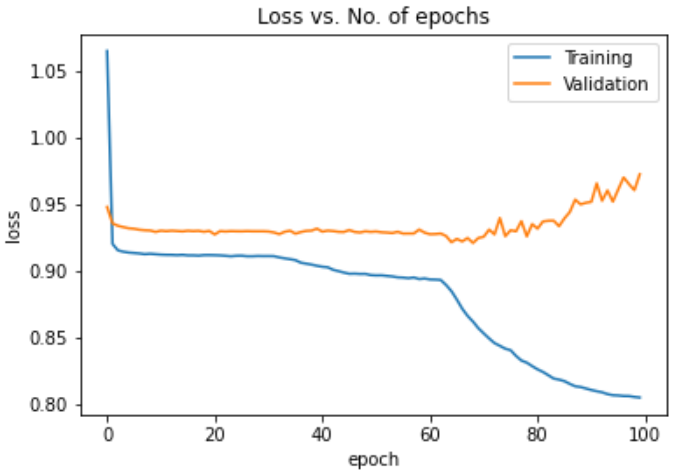


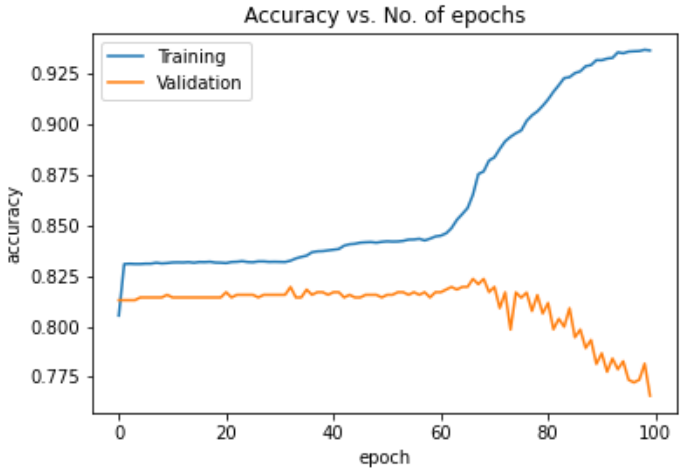


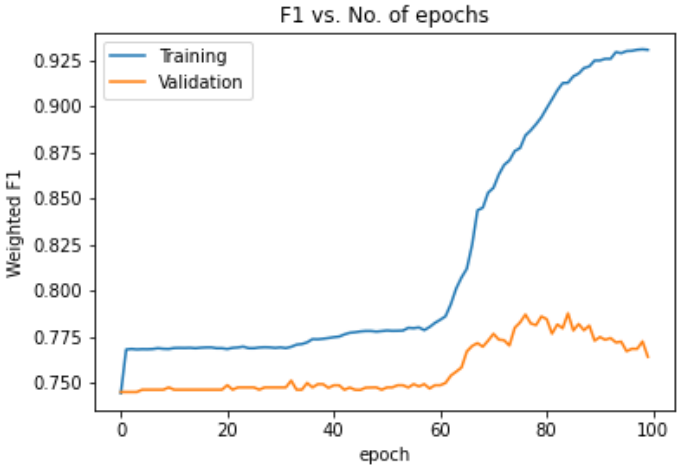



```
Train Accuracy: 95.6968
Validation Accuracy: 82.7495
Test Accuracy: 82.7903
```

b) X = 1

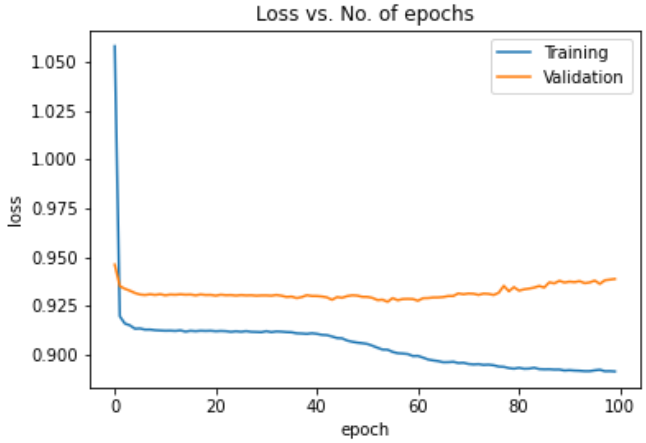


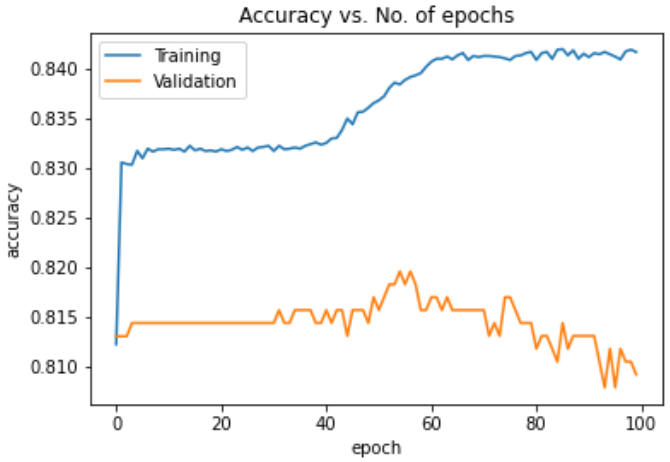


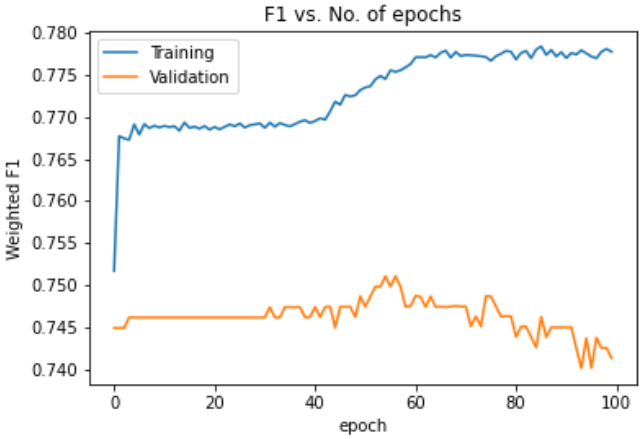



```
Train Accuracy: 93.6630
Validation Accuracy: 76.5709
Test Accuracy: 74.1902
```

c) X = 2

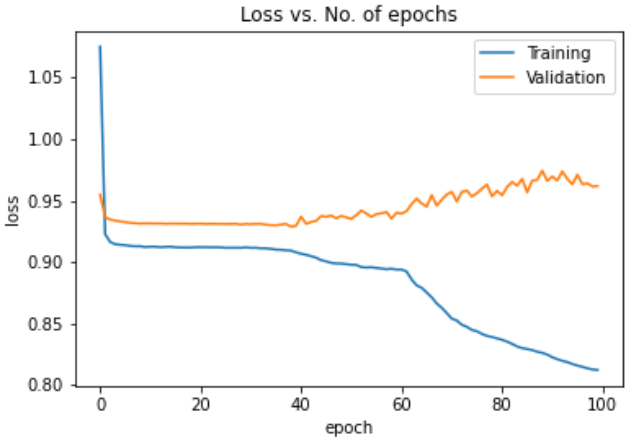


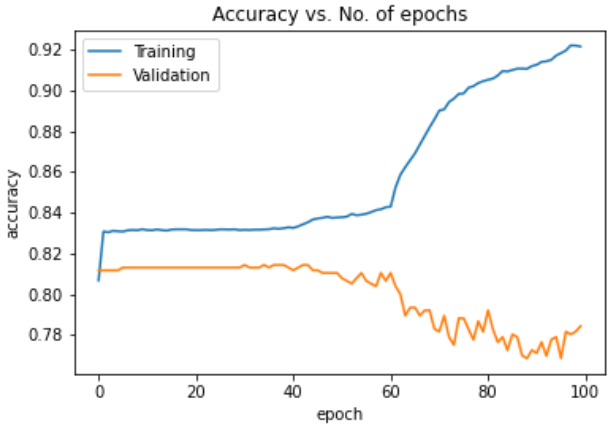


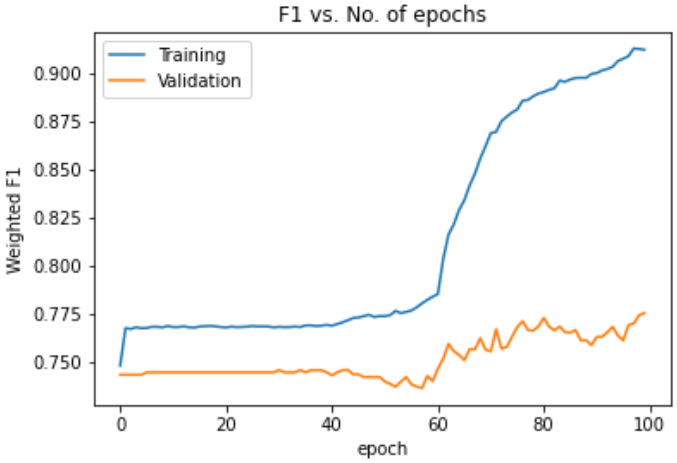



```
Train Accuracy: 84.1723
Validation Accuracy: 80.9140
Test Accuracy: 77.2167
```

d) X = 3

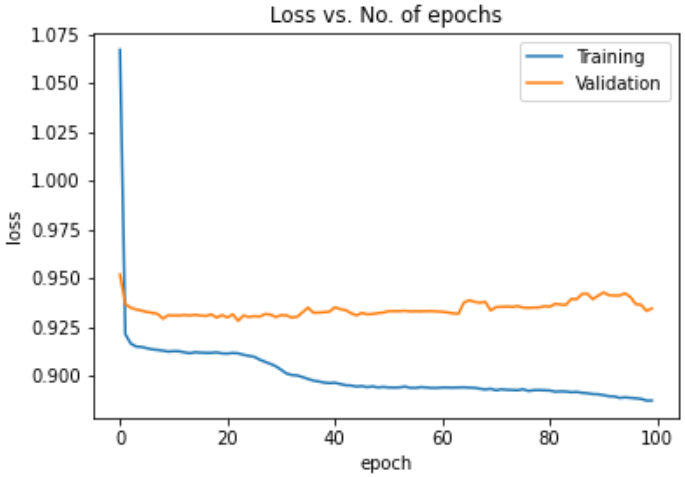


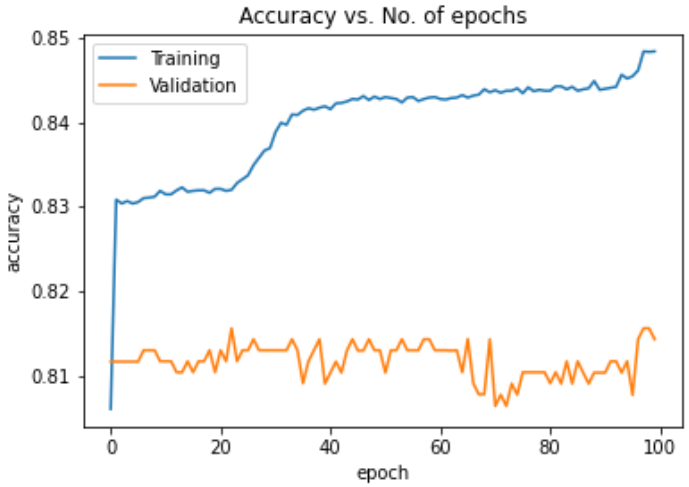


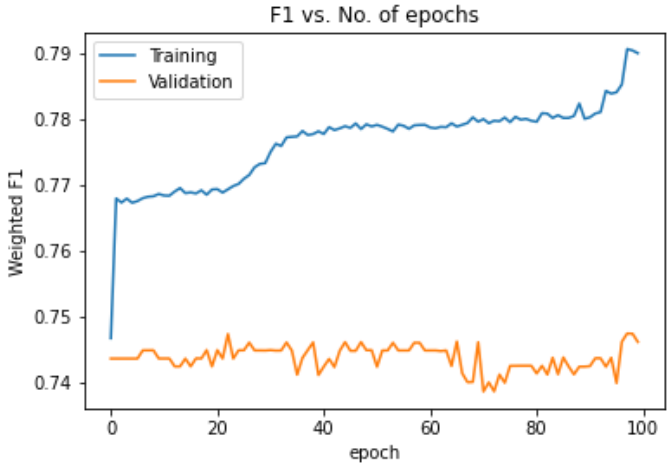



```
Train Accuracy: 92.1573
Validation Accuracy: 78.4274
Test Accuracy: 76.4768
```

e) X = 4








```
Train Accuracy: 84.8386
Validation Accuracy: 81.4348
Test Accuracy: 77.5502
```

Q4. Does the performance of the model increase with the increase in X? Justify.

	Train Score	Val Score	Test Score
X = 0	96.34	82.21	83.38
X = 1	93.66	76.57	74.19
X = 2	84.17	80.91	77.21
X = 3	92.15	78.42	76.47
X = 4	84.83	81.43	77.55

For the same number of epochs(100) and the same learning_rate of 0.001 and a dropout probability of 0.25, it can be seen that our model performs the best for X = 0. As we increase the value of X, a general trend can not be noticed as the performance fluctuates for each value of X and is neither increasing nor decreasing monotonically. It can be seen that X=4 is able to capture enough information to provide good results but is not good enough to beat the performance of X = 0. In the current dataset, utilising the previous utterances does not necessarily make sense as our model could capture enough information out of simple words like ‘what’ and punctuations like ‘?’. The dialogues are not coherent in the given dataset.

Another reason could be that the present architecture cannot take advantage of the past utterances as context as the size of the tokens in the sequence is increasing over each increment in X, which could be causing vanishing gradient problem or other performance issues.

As we saw from our experiments, the model's performance doesn’t improve when we use more and more of the past utterances as context. The present architecture cannot take advantage of the past utterances, but rather, its performance decreases as we give it more of the previous utterances.