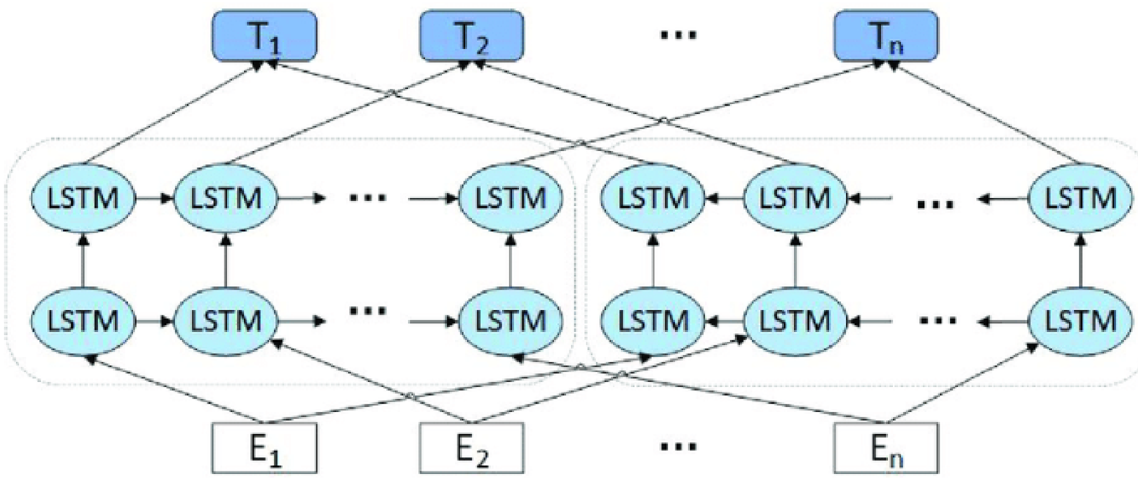
Report

1.Pretraining ELMO:

Architecture:



Hyperparameters used:

```
# hyperparameters
batch_size = 64
embedding_dim = 300
hidden_dim = 300
num_layers = 1
learning_rate = 0.001
num_epochs = 10
device = torch.device('cuda' if torch.cuda.is_available() else
```

Training ELMO:

```
Epoch: 1/10, Loss: 8.846520706431072
Epoch: 2/10, Loss: 7.307172158813477
Epoch: 3/10, Loss: 6.746782798512776
Epoch: 4/10, Loss: 6.397141324361165
Epoch: 5/10, Loss: 6.142149953460693
Epoch: 6/10, Loss: 5.938797519938151
Epoch: 7/10, Loss: 5.770049016571045
Epoch: 8/10, Loss: 5.624275687917073
Epoch: 9/10, Loss: 5.495608435058593
Epoch: 10/10, Loss: 5.38072425181071
```

2. Downstream Task:

The word embeddings are obtained from the pretrained ELMO as follows:

$$E = \lambda_1 e + \lambda_2 h_1 + \lambda_3 h_2$$

$$e = concat(ef, eb)$$

$$h_1 = concat(h_{f1}, h_{b1})$$

$$h_2 = concat(h_{f2}, h_{b2})$$

Here e_f and e_b refers to the embeddings from the Embedding layer for forward LSTM and backward LSTM respectively. In the same way, [h_{f1} and h_{b1}] and [h_{f2} and h_{b2}] refers to the hidden states of first and second stacks of forward LSTM and backward LSTM.

Hyperparameters used:

```
# hyperparameters
   embedding_dim = 600
   hidden_dim = 300
   num_layers = 2
   num_classes = len(set(labels))
   batch_size = 64
   activation_fn = nn.ReLU()
```

```
    bidirectional = True
    device = torch.device('cuda' if torch.cuda.is_available() el
```

1. Trainable λs

```
Training Model...
Epoch 1/5, Train Loss: 0.6077, Train Accuracy: 0.7601, Val Loss: 0.3773, Val Accuracy: 0.8638
Epoch 2/5, Train Loss: 0.3056, Train Accuracy: 0.8922, Val Loss: 0.3787, Val Accuracy: 0.8664
Epoch 3/5, Train Loss: 0.2138, Train Accuracy: 0.9245, Val Loss: 0.3931, Val Accuracy: 0.8695
Epoch 4/5, Train Loss: 0.1593, Train Accuracy: 0.9442, Val Loss: 0.4117, Val Accuracy: 0.8682
Epoch 5/5, Train Loss: 0.1268, Train Accuracy: 0.9563, Val Loss: 0.4364, Val Accuracy: 0.8701
Lambda Values: (1.1081287860870361, 0.3256469666957855, -0.018114658072590828)
```

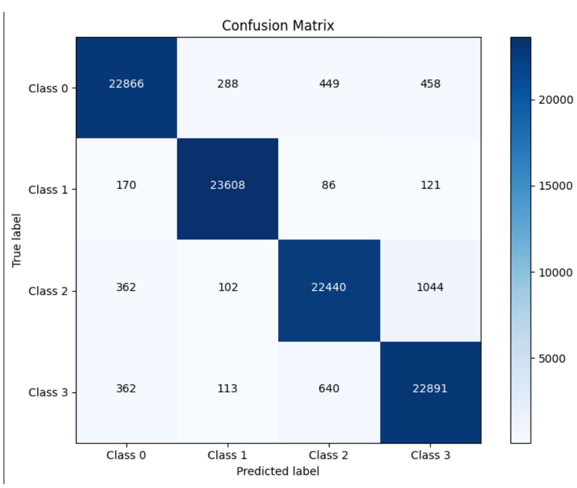
λs after training : (1.1081287860870361, 0.3256469666957855, -0.018114658072590828)

Metrics :

Training data:

```
---------------------------------------------
Final Metrics on Train Data:
---------------------------------------------
Accuracy: 0.9563020833333333
Precision: 0.9563892433213067
Recall: 0.9563020833333333
F1 Score: 0.9562970774367474
```

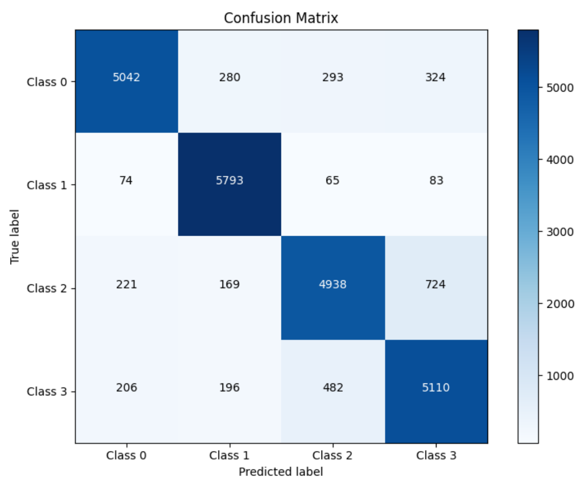
Confusion Matrix



Validation data:

```
--------------------------------------------
Final Metrics on Validation Data:
--------------------------------------------
Accuracy: 0.870125
Precision: 0.8706047665909579
Recall: 0.870125
F1 Score: 0.869638667248279
```

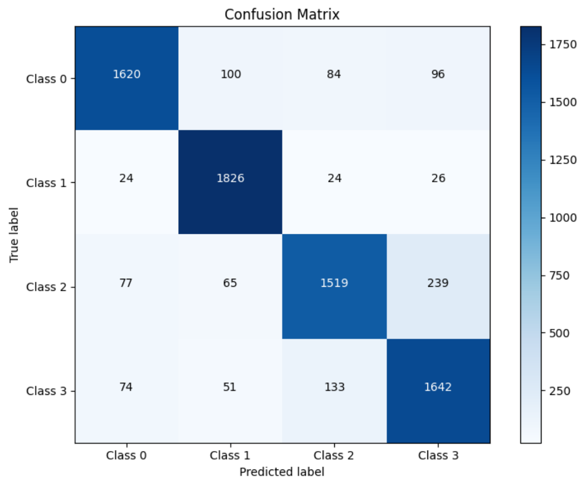
Confusion matrix



Test data:

```
----------------------------------------------
Final Metrics on Test Data:
----------------------------------------------
Accuracy: 0.8693421052631579
Precision: 0.8698917104264552
Recall: 0.8693421052631579
F1 Score: 0.868688149509318
```

Confusion Matrix



2. Frozen λs

```
Training Model...
Epoch 1/5, Train Loss: 0.8038, Train Accuracy: 0.6639, Val Loss: 0.5308, Val Accuracy: 0.7991
Epoch 2/5, Train Loss: 0.4296, Train Accuracy: 0.8417, Val Loss: 0.4536, Val Accuracy: 0.8303
Epoch 3/5, Train Loss: 0.3103, Train Accuracy: 0.8877, Val Loss: 0.4007, Val Accuracy: 0.8537
Epoch 4/5, Train Loss: 0.2263, Train Accuracy: 0.9194, Val Loss: 0.4444, Val Accuracy: 0.8491
Epoch 5/5, Train Loss: 0.1685, Train Accuracy: 0.9408, Val Loss: 0.4650, Val Accuracy: 0.8542
Lambda Values: (0.3507940769195566, 0.7755380868911743, 0.07888883352279663)
```

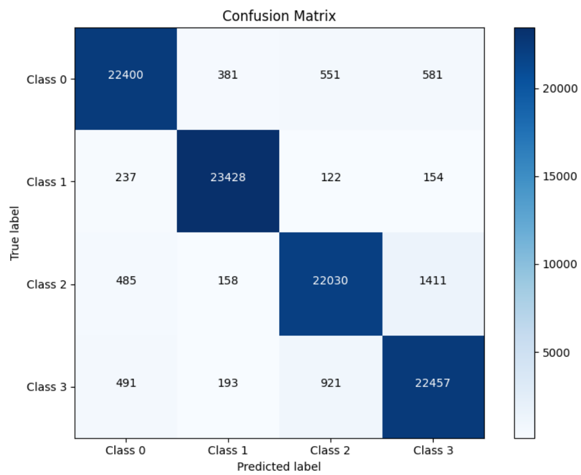
λs after training : (0.3507940769195566, 0.7755380868911743, 0.07888883352279663)

Metrics :

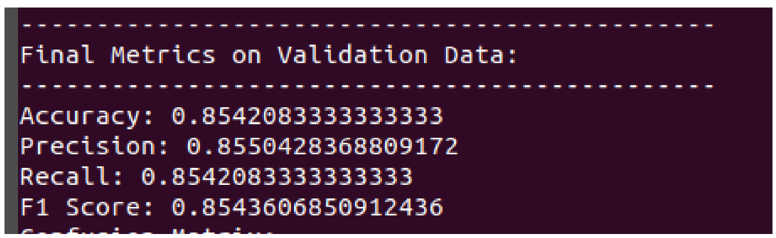
Training Data:

```
---------------------------------------------
Final Metrics on Train Data:
---------------------------------------------
Accuracy: 0.94078125
Precision: 0.9408580743028069
Recall: 0.94078125
F1 Score: 0.9407551363295897
```

Confusion matrix:

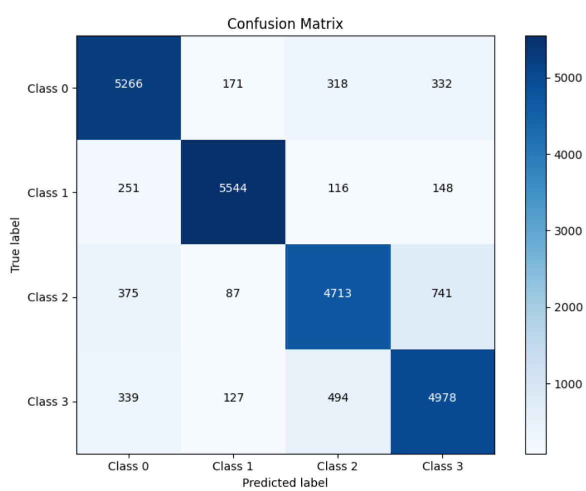


Validation Data

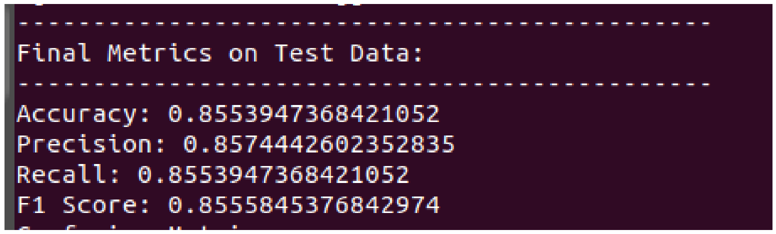


```
---------------------------------------------
Final Metrics on Validation Data:
---------------------------------------------
Accuracy: 0.8542083333333333
Precision: 0.8550428368809172
Recall: 0.8542083333333333
F1 Score: 0.8543606850912436
```

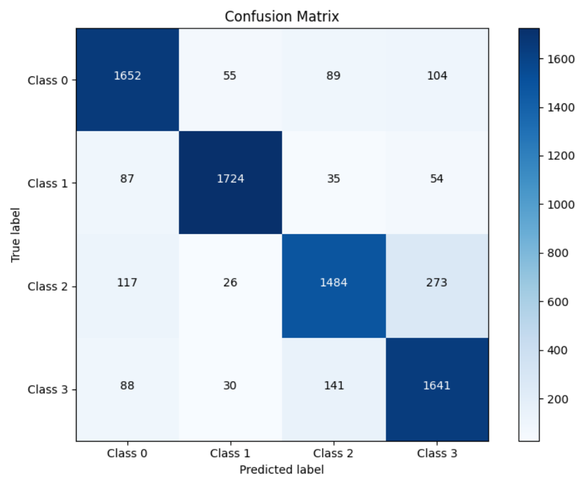
Confusion matrix



Test Data:



Confusion matrix



3. Learnable Function

As a neural network learns a function, I used a Feed Forward Neural Network to learn the function.

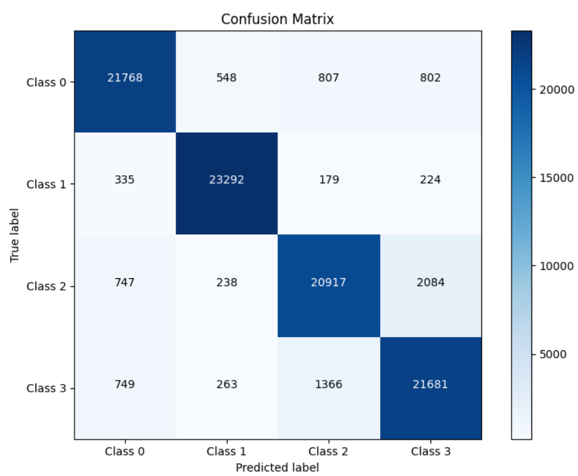
```
class FFNN(nn.Module):
    def __init__(self, input_dim, output_dim, activation_fn):
        super(FFNN, self).__init__()
        self.fc1 = nn.Linear(input_dim,output_dim)
        self.activation_fn = activation_fn

    def forward(self, X):
        out = self.fc1(X)
        out = self.activation_fn(out)
        return out
```

```
Training Model...
Epoch 1/5, Train Loss: 0.8574, Train Accuracy: 0.6238, Val Loss: 0.5301, Val Accuracy: 0.7995
Epoch 2/5, Train Loss: 0.4631, Train Accuracy: 0.8279, Val Loss: 0.4261, Val Accuracy: 0.8438
Epoch 3/5, Train Loss: 0.3642, Train Accuracy: 0.8672, Val Loss: 0.4092, Val Accuracy: 0.8503
Epoch 4/5, Train Loss: 0.2949, Train Accuracy: 0.8943, Val Loss: 0.3723, Val Accuracy: 0.8658
Epoch 5/5, Train Loss: 0.2430, Train Accuracy: 0.9131, Val Loss: 0.3854, Val Accuracy: 0.8659
```

Training Data

```
---------------------------------------------
Final Metrics on Train Data:
---------------------------------------------
Accuracy: 0.9131041666666667
Precision: 0.9131819190318811
Recall: 0.9131041666666667
F1 Score: 0.9130214371750641
```

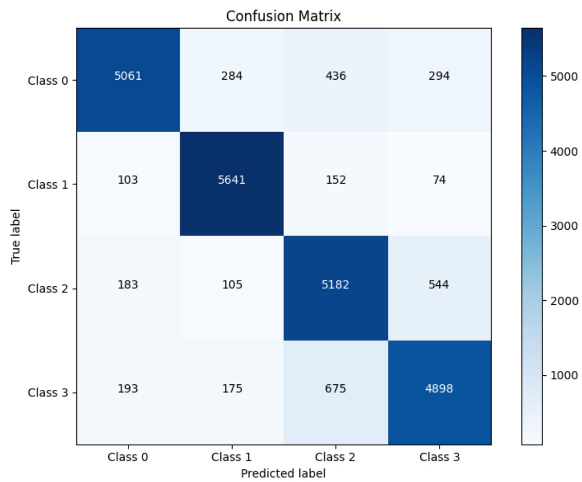
Confusion matrix



Validation Data

```
---------------------------------------------
Final Metrics on Validation Data:
---------------------------------------------
Accuracy: 0.8659166666666667
Precision: 0.8675417122517627
Recall: 0.8659166666666667
F1 Score: 0.8658988809065119
```

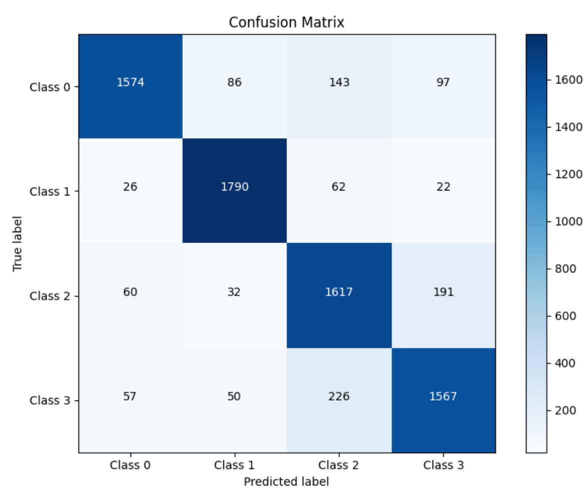
Confusion matrix



Test Data

```
-------------------------------------------
Final Metrics on Test Data:
-------------------------------------------
Accuracy: 0.861578947368421
Precision: 0.8638267444702749
Recall: 0.861578947368421
F1 Score: 0.8617961186054096
```

Confusion Matrix



Comparing the above 3 hyperparameter tuning the order of performance of the models on test data is observed as : trainable λs > frozen λs > learnable function.

Comparing SVD, Skipgram and ELMO embeddings

Test accuracies of best performing SVD, Skipgram and ELMO embeddings on downstream task is as follows :

- SVD (CW=3) : 84.6578
- Skipgram (CW=3) : 77.657
- ELMO(trainable λs) : 86.9342

We can observe that the ELMO embeddings perform better compared to that of the SVD and Skipgram. The reasons can be :

- Firstly, ELMo understands words based on how they're used in sentences, which helps with understanding words that have multiple meanings. SVD and skip-gram embeddings don't consider this context.
- Secondly, ELMo is pre-trained on a lot of text using advanced language models, making it good at understanding language nuances. This pre-training also helps when fine-tuning for specific tasks, making ELMo adaptable and effective.
- Thirdly, ELMo's neural network-based approach allows it to handle complex linguistic patterns better and learn faster compared to SVD and skip-gram.
- Lastly, ELMo needs less labeled data for training,as it is pre-trained already, which is an advantage when working with smaller datasets.