Multilingual Classification using Neural Networks

Ashmita Mukherjee

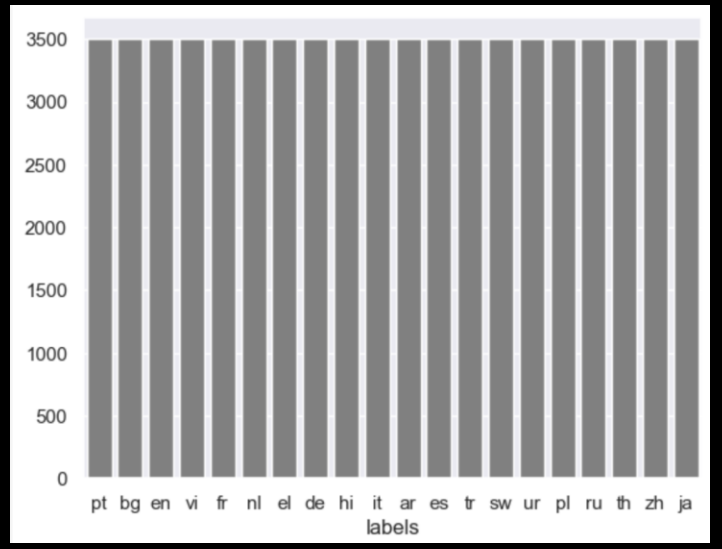
GitHub: *Multilingual Classification*

Introduction

- *Objective* is to develop neural network models capable of detecting the language of a given text input.
- Neural network is the right choice because the complexities and *autoregressive manner of texts* can be well captured by RNNs and 1D CNNS.
- The dataset is from Hugging Face and has a balanced number of data points (3500) for each of the 20 classes. *https://huggingface.co/datasets/papluca/language-identification*
- The train set contains 70k samples, while the validation and test sets 10k each.
- Given languages are: *arabic (ar), bulgarian (bg), german (de), modern greek (el), english (en), spanish (es), french (fr), hindi (hi), italian (it), japanese (ja), dutch (nl), polish (pl), portuguese (pt), russian (ru), swahili (sw), thai (th), turkish (tr), urdu (ur), vietnamese (vi), and chinese (zh)*

Data Analysis

- The labels have equal distributions with 3500 texts in each category.
- This is essential for a neural network to be able to learn each category equally.
- For Validation and Test sets each have 500 texts in each category.



	labels	text
0	pt	os chefes de defesa da estónia, letónia, lituâ...
1	bg	размерът на хоризонталната мрежа може да бъде ...
2	zh	很好，以前从不去评价，不知道浪费了多少积分，现在知道积分可以换钱，就要好好评价了，后来我就把...
3	th	สำหรับ ของเก่า ที่ จริงจัง ลอง honeychurch ...
4	ru	Он увеличил давление .

A look into the dataset with the original text prior to encoding.

After encoding and scaling the text data was transformed into numeric data (768 embeddings for each token)and ready for the neural network

0.213562	-0.282861	0.106654	-1.545983	-1.014773	1.050059	-0.392528	0.321693	-0.195429	-0.079574
-0.259685	-0.377271	1.157918	0.357854	0.661032	0.274081	-0.447011	-0.004175	1.368662	0.126061
0.184296	0.923643	-1.464936	-0.426851	0.292415	-0.497616	0.180975	0.727360	-0.547008	-0.856253
-1.489129	-0.159570	-0.992029	1.613393	-1.123803	-0.539435	-0.702530	1.038175	-0.141246	-0.008131

Preprocessing steps

- Text cannot be processed by neural networks by itself.
- It needs to be transformed into a suitable format for the model and text data is usually converted into embeddings for this purpose.
- I have used to create *xlm-roberta-base* embeddings for the text. This model was used since it can accommodate multilingual data embeddings.
- The resulting tensor is of shape *(length_of_text,768)*.
- For homogeneity, I have converted it to shape *(1,768)* by taking the mean of each along *len(length_of_text)*.
- Further scaled the data which is essential for NN.
- After running the models of this length, it was taking significant amount of time. Hence applying *PCA* would be beneficial.
- PCA from 768 to 350. Explained variance: *98.6. X_train is (70000, 350)*
- Labels are One-Hot encoded. *Y_train is (70000,20)*
- Same has been done with the test and validation set and they are of shape *X = (10000,350) and y = (10000,20)*

Models used

I have used the following models for the classification of multilingual texts. The models are ordered from worst performing to best performing. My model selection process was iterative and I considered regularizing models, performing early stopping according to their progression. My models started from below 10% accuracy and worked up to 80% and above as the epochs increased which showed that the model was indeed learning well. There is slight overfit and underfit which I have tried so minimize but also kept some as the val accuracy and test were similar and it depicted the importance of using cross validation in any ML technique.

GRU

LSTM

1D CNN

BiLSTM

LSTM & CNN

GRU

TECHNIQUE:-

- Increasing the number of units significantly increased the accuracy and reduced the loss. I started with two 16 units and received a 68% accuracy but on increasing it to 32& 64, it reached 80.38%.
- I took f1-score as a metric since it shows the individual performance of each class and we can see that some classes perform considerably well whereas class 13 (Russian) can just predict correctly 25% of the time suggesting that the model has difficulty predicting this class correctly.
- A solution to this is to increase the class weights. I have done so for other models but kept this intact to highlight the issue.

Layer (type)	Output Shape	Param #
gru_14 (GRU)	(None, 350, 32)	3,360
dropout_51 (Dropout)	(None, 350, 32)	0
batch_normalization_28 (BatchNormalization)	(None, 350, 32)	128
gru_15 (GRU)	(None, 64)	18,816
dropout_52 (Dropout)	(None, 64)	0
batch_normalization_29 (BatchNormalization)	(None, 64)	256
dense_87 (Dense)	(None, 20)	1,300

Total params: 23,860 (93.20 KB)

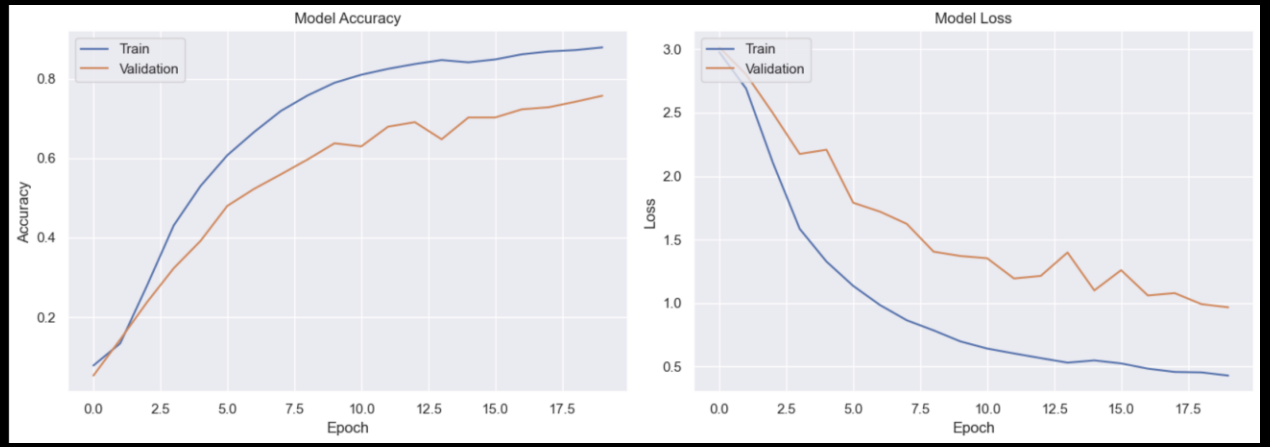
Trainable params: 23,668 (92.45 KB)

Non-trainable params: 192 (768.00 B)

```
[0.7674481868743896,
 <tf.Tensor: shape=(20,), dtype=float32, numpy=
 array([0.8949011 , 0.5687074 , 0.9294117 , 0.88583505, 0.79687494,
        0.9257142 , 0.9685039 , 0.5958041 , 0.9390243 , 0.89962816,
        0.67736995, 0.798623  , 0.64441216, 0.25034383, 0.94335735,
        0.8564971 , 0.934322  , 0.89084893, 0.8057553 , 0.8623853 ],
       dtype=float32)>,
 0.8113999962806702]
```

RESULTS:-

- The train accuracy > val accuracy and test accuracy indicating that the model is overfitting.
- Similarly, the loss is 0.4 for train set and 0.7 for test.
- To overcome this, we can put in more dropout layers or introduce the L2 regularization.



LSTM

TECHNIQUE:-

- Including 3 LSTM layers , the model has a 80.74% accuracy. Adding more LSTM layers increased the accuracy considerably.
- Introducing batch normalization helps the model converge faster.
- I was initially using many dense layers but simplifying them cut out the time and did not affect accuracy much.
- The model fails to identify 2 classes correctly indicating the need of class weights increase to about 2x for better performance.

Layer (type)	Output Shape	Param #
lstm_21 (LSTM)	(None, 350, 64)	16,896
dropout_42 (Dropout)	(None, 350, 64)	0
lstm_22 (LSTM)	(None, 350, 64)	33,024
dropout_43 (Dropout)	(None, 350, 64)	0
batch_normalization_20 (BatchNormalization)	(None, 350, 64)	256
lstm_23 (LSTM)	(None, 32)	12,416
dropout_44 (Dropout)	(None, 32)	0
batch_normalization_21 (BatchNormalization)	(None, 32)	128
dense_81 (Dense)	(None, 64)	2,112
dense_82 (Dense)	(None, 20)	1,300

Total params: 66,132 (258.33 KB)

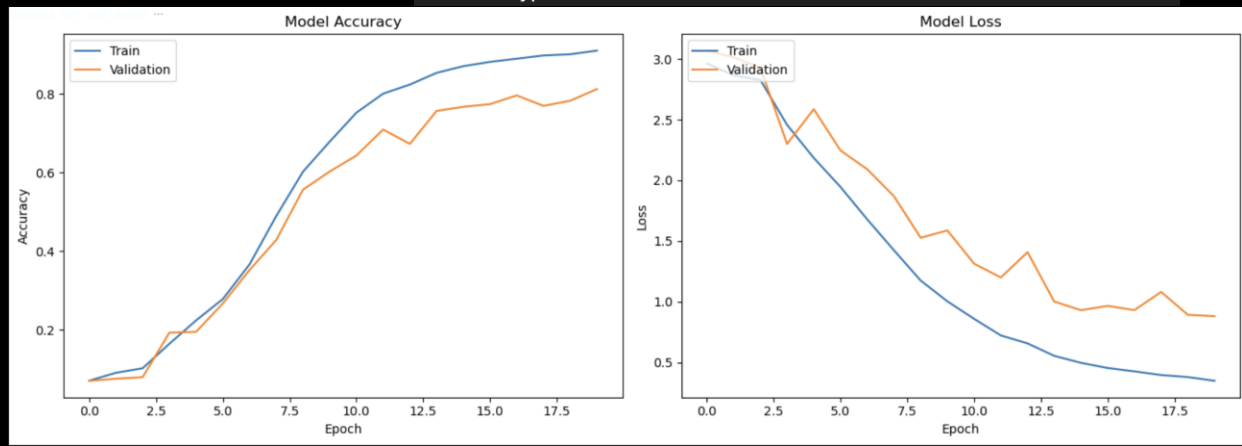
Trainable params: 65,940 (257.58 KB)

Non-trainable params: 192 (768.00 B)

```
[0.8346284031867981,
 0.8151999711990356,
 <tf.Tensor: shape=(20,), dtype=float32, numpy=
 array([0.78407955, 0.7112763 , 0.9549179 , 0.7644444 , 0.86039877,
        0.94399995, 0.96124023, 0.39228293, 0.93541664, 0.9285714 ,
        0.5756276 , 0.92914975, 0.7981042 , 0.38929433, 0.95208967,
        0.8199767 , 0.9078821 , 0.9142236 , 0.8061716 , 0.88337916],
       dtype=float32)>]
```

RESULTS:-

- Training accuracy (90.78%) > val (81.24) and test (80.74).
- The model seems to be overfitting on the data suggesting that the model is too complex for the data and cutting back 1 layer of LSTM might help stabilize the situation.



BiLSTM

TECHNIQUE:-

- Bidirectional model has more impact in case of contextual understanding. First, I used 1 BiLSTM in the output layer which gave 80% accuracy. To test it further, I decided to stack 2 BiLSTMs and there was a 5% jump in lesser epochs. Therefore, stacking BiLSTMs is powerful.
- Initially, the model could still not predict classes 7 and 11 correctly so I increased the class weights for the 2 and it created a significant impact on its performance with nearly a 2.5x better performance.
- This shows that increasing the class weights can improve individual f1-score.

RESULTS:-

- The model overfits the training data but the val accuracy and test accuracy are approximately equal.
- This emphasizes the need of a validation dataset and val should be treated as the actual eval metric instead of train which gives a false notion of the accuracy.
- The loss is also seems to be decreasing at the same pace initially but later train loss< val.

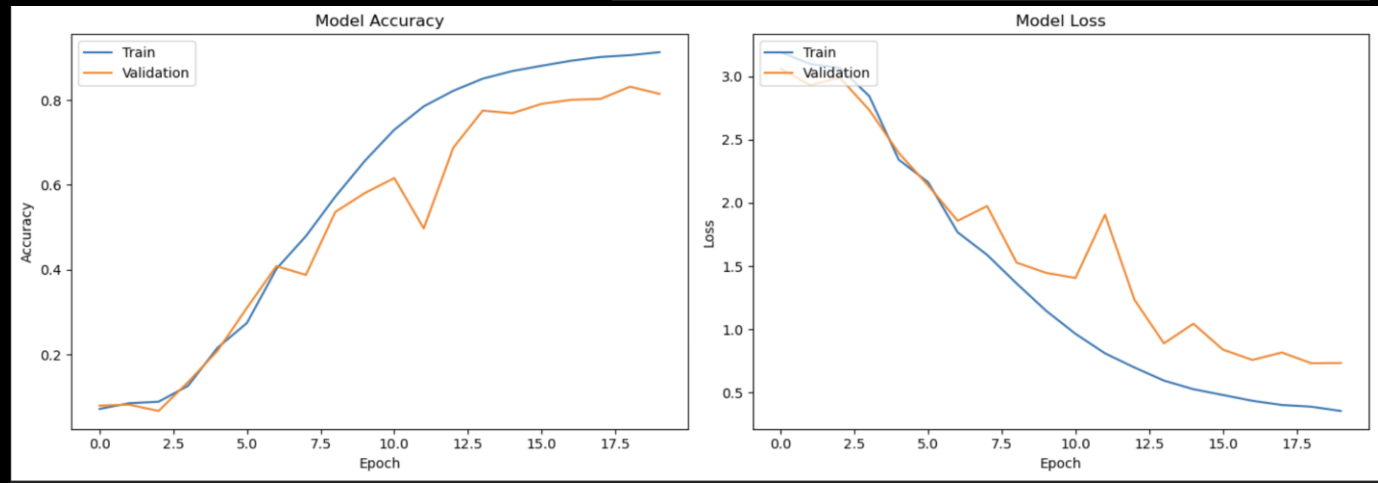
Layer (type)	Output Shape	Param #
bidirectional_8 (Bidirectional)	(None, 350, 96)	19,200
dropout_11 (Dropout)	(None, 350, 96)	0
batch_normalization_9 (BatchNormalization)	(None, 350, 96)	384
bidirectional_9 (Bidirectional)	(None, 64)	33,024
dropout_12 (Dropout)	(None, 64)	0
batch_normalization_10 (BatchNormalization)	(None, 64)	256
dense_8 (Dense)	(None, 64)	4,160
dense_9 (Dense)	(None, 20)	1,300

Total params: 58,324 (227.83 KB)

Trainable params: 58,004 (226.58 KB)

Non-trainable params: 320 (1.25 KB)

```
[0.532986044883728,
 0.8535000085830688,
 <tf.Tensor: shape=(20,), dtype=float32, numpy=
 array([0.73506194, 0.7868852 , 0.92725503, 0.8874296 , 0.8761514 ,
        0.9370764 , 0.9717172 , 0.7004131 , 0.93028086, 0.9475766 ,
        0.70077515, 0.8608776 , 0.7515151 , 0.7343412 , 0.94516444,
        0.68676275, 0.922619  , 0.91772145, 0.8997078 , 0.9291949 ],
       dtype=float32)>]
```



1D-CNN

TECHNIQUE:-

- 1D CNN is proficient at feature extraction from text. This is the second- best model.
- From the f1-score it can be seen that the model predicts the classes accurately without the need of any class weight initialization. It has a kernel size of 7.
- It is preferable and practice to use a larger kernel size in case of text classification.
- Deeper networks are better than wider networks for computational efficiency so I have used 4 layers of size 16.

Layer (type)	Output Shape	Param #
conv1d_14 (Conv1D)	(None, 344, 16)	128
dropout_27 (Dropout)	(None, 344, 16)	0
conv1d_15 (Conv1D)	(None, 338, 16)	1,808
dropout_28 (Dropout)	(None, 338, 16)	0
max_pooling1d_9 (MaxPooling1D)	(None, 169, 16)	0
conv1d_16 (Conv1D)	(None, 163, 16)	1,808
dropout_29 (Dropout)	(None, 163, 16)	0
conv1d_17 (Conv1D)	(None, 157, 16)	1,808
dropout_30 (Dropout)	(None, 157, 16)	0
max_pooling1d_10 (MaxPooling1D)	(None, 78, 16)	0
dropout_31 (Dropout)	(None, 78, 16)	0
flatten_1 (Flatten)	(None, 1248)	0
dense_57 (Dense)	(None, 20)	24,980

Total params: 30,532 (119.27 KB)

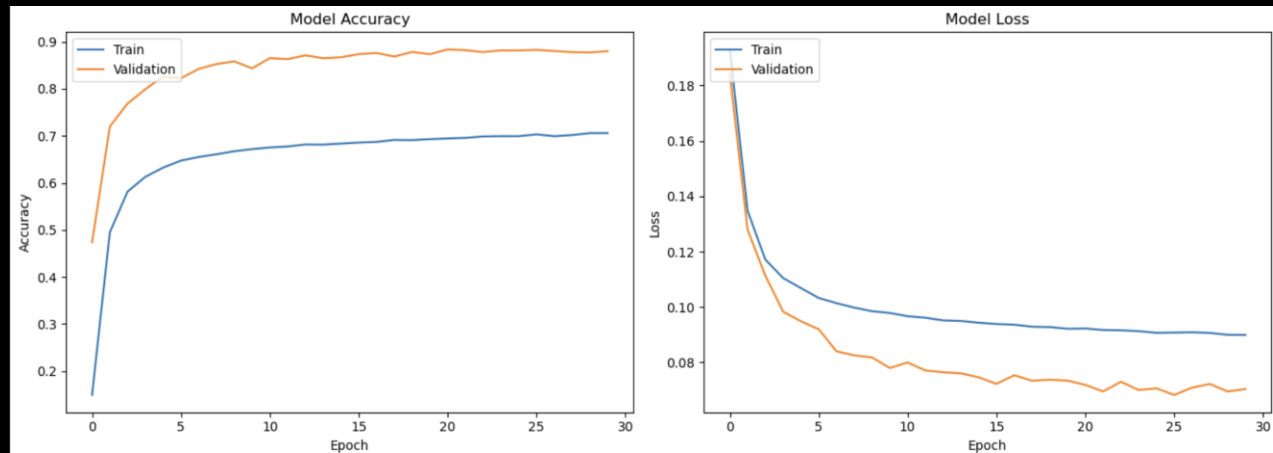
Trainable params: 30,532 (119.27 KB)

Non-trainable params: 0 (0.00 B)

```
[0.07035670429468155,
 0.8823000192642212,
 <tf.Tensor: shape=(20,), dtype=float32, numpy=
 array([0.83582085, 0.8287526 , 0.95427436, 0.80282927, 0.92450875,
        0.9564336 , 0.94187194, 0.70194   , 0.9404517 , 0.9838709 ,
        0.8049029 , 0.94884646, 0.8602383 , 0.8365878 , 0.9046692 ,
        0.8656387 , 0.91707313, 0.78805393, 0.9435084 , 0.93873304],
       dtype=float32)>]
```

RESULTS:-

- The accuracy and loss graphs here show that the model is underfitting.
- The model has 88% accuracy of the train and test data but only 70% on training.
- This suggests that there are outliers in the training dataset which the model predicts with low confidence and is mostly incorrect.
- Due to selection bias the val and test sets which are comparatively smaller and have a better distribution, the model performs confidently on those.
- Thus, the f1-score is also finally over 88%.



CNN + BiLSTM

TECHNIQUE:-

- This is the best performing model. It shows how comparing 2 methodologies can significantly increase model performance.
- The f1-score for each of the classes is high without the need of class weight initialization. Except Russian, all classes have 90%+ f1-score.
- I have kept the model relatively simple with 2 CNN layers and BiLSTM.
- As opposed to the 20-30 epochs that other models took to run, this model gave the best performance in only 10 epochs.
- Therefore, this model was the fastest to train and best performing making CNN+BiLSTM a suitable combination for text classification.

Layer (type)	Output Shape	Param #
conv1d_32 (Conv1D)	(None, 350, 32)	256
max_pooling1d_25 (MaxPooling1D)	(None, 175, 32)	0
conv1d_33 (Conv1D)	(None, 175, 64)	14,400
max_pooling1d_26 (MaxPooling1D)	(None, 87, 64)	0
bidirectional_7 (Bidirectional)	(None, 128)	66,048
dense_67 (Dense)	(None, 20)	2,580

Total params: 83,284 (325.33 KB)

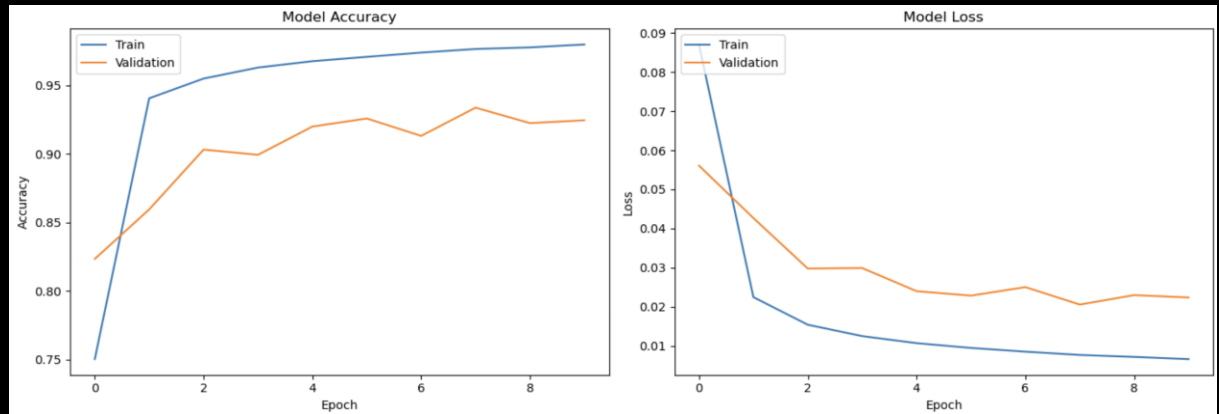
Trainable params: 83,284 (325.33 KB)

Non-trainable params: 0 (0.00 B)

```
[0.020196808502078056,
 0.9384999871253967,
 <tf.Tensor: shape=(20,), dtype=float32, numpy=
 array([0.9334698 , 0.9253731 , 0.974155  , 0.9293892 , 0.9355468 ,
        0.97061795, 0.9841269 , 0.9484536 , 0.96890664, 0.94375587,
        0.90073526, 0.92366403, 0.8976834 , 0.78422266, 0.96709865,
        0.93617016, 0.969031  , 0.9744637 , 0.9467336 , 0.94061303],
       dtype=float32)>]
```

RESULTS:-

- The train accuracy and loss is slightly more than the val and test accuracy and loss so it is slightly overfitting.
- This can be controlled with regularization (L1/L2) or Dropout layer introduction.



Result Comparison

Model	Accuracy	Loss	F1-Score
GRU	80.38	0.7955	0.7951
LSTM	80.74	0.8604	0.8032
BiLSTM	84.78	0.5398	0.8460
1D CNN	88.38	0.0707	0.8848
CNN+ BiLSTM	93.91	0.0198	0.9377

CNN+BiLSTM gives the best performance.

Further, I have also tested an LLM finetuned on multilingual data "xlm-roberta-base-language-detection" and performed zero shot learning on the model. The model has an f1-score of 99.5%.

Learnings

- My results before and after adding the dropout layer have proven that the dropout layer is essential for generalization and helps with overfitting. Early Stopping greatly helps with overfitting as well.
- BatchNormalization facilitates faster convergence by mitigating the covariate shift due to the layer transformations.
- BiLSTM helps in capturing autoregressive dependencies in both left to right and right to left directions, this is crucial for me because some languages are written right to left instead of left to right. The BiLSTM really shone for these languages.
- The LSTM did suffer in predicting a few classes for this dataset and needed the help of class weight even though this dataset is balanced. The languages LSTM had trouble predicting may have lesser autoregressive characteristics than other languages.
- CNN-LSTM turned out to be a great combination, extracting a blend of feature extraction in the text while also capturing it's autoregressive nature helping us get the highest accuracy within 10 epochs.
- CNN is a very powerful algorithm as the extraction and convolving feature helps a lot in retaining information. However, it took 30 epochs epochs for performing well so its learning is slow compared to RNNs.
- My next steps would be to fine-tune an LLM on this dataset and compare its computational efficiency with that of my customized models.