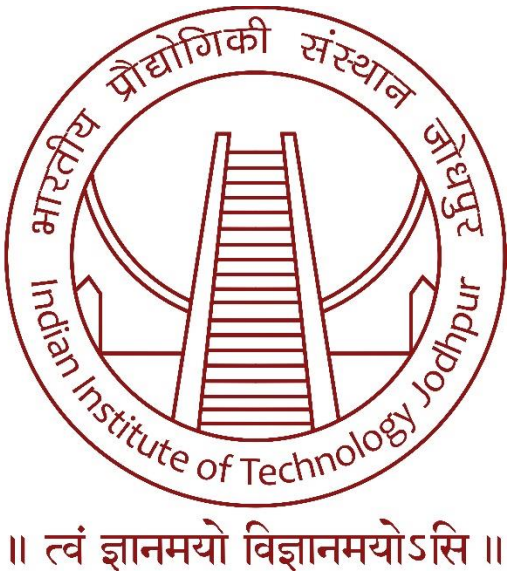
Assignment 2

Deep Learning



Submitted By:

Gyanendra Chaubey
M.Tech (AR-VR)
Roll No: M23AIR005

Submitted To:

Dr. Deepak Mishra
Assistant Professor
Dept. of CSE

Methodology:

This assignment is given to train two architectures (CNN and Transformer) for the audio classification. In the Transformer, it is required to train the model with 1, 2, and 4 number of attention heads. Also, each of the architecture need to be trained with k-Fold validation method (k=4) and train-test split method for 100 epochs.

Preprocessing to match the dimension of labels and signal: A preprocessing has been done to match the labels with input data. After applying the window and sampling the input signal has been converted to (9,1,16000) corresponding to which there was only a single label. It was causing dimensionality error while feeding to the model. So, for each of the window sampled, a label has been generated. So, for (9,1,16000) input now there is 9 labels. So that each of the signal has a size of (1,16000) with a unique same label.

Architecture1: Convolutional Neural Network Architecture:

Firstly, Draw a CNN architecture with 3 base convolutional layers and a fully connected layer. Since the input was of size (1,16000) that's why we have chosen 1D convolution for the extraction of the features.
The architecture of CNN is given in the Fig. 1.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1           [-1, 128, 8000]           1,024
       BatchNorm1d-2           [-1, 128, 8000]             256
           Dropout-3           [-1, 128, 8000]               0
         MaxPool1d-4           [-1, 128, 4000]               0
            Conv1d-5            [-1, 64, 2000]          41,024
       BatchNorm1d-6            [-1, 64, 2000]             128
           Dropout-7            [-1, 64, 2000]               0
         MaxPool1d-8            [-1, 64, 1000]               0
            Conv1d-9             [-1, 32, 500]           6,176
      BatchNorm1d-10             [-1, 32, 500]              64
          Dropout-11             [-1, 32, 500]               0
        AvgPool1d-12             [-1, 32, 250]               0
           Linear-13                  [-1, 10]          80,010
================================================================
Total params: 128,682
Trainable params: 128,682
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.06
Forward/backward pass size (MB): 31.19
Params size (MB): 0.49
Estimated Total Size (MB): 31.74
----------------------------------------------------------------
```

Fig. 1 Architecture of the CNN

The first Conv layer is containing the 128 filters with kernel_size=7, padding=3 and stride=2. The number of filters has been kept high in the first layer so that it can extract the different type of high-level features in the first layer because going forward in next layers signal will lose its dimensions. Then applied, batch normalization to normalize the outcome of the conv1, afterwards applied a dropout with p=0.5 followed by maxpooling with kernel_size=2 and stride=2.

In the second Conv layer, 64 filters have been used for the feature extraction with kernel_size = 5, padding=2, and stride=2. In this layer, reduced the number of filters and filter_size here to get the mid-level features. Then, applied batch normalization and maxpooling same as in first layer.

In the third Conv layer, 32 filters have been applied with kernel_size=3 and padding=1 and stride=2. In this layer, high level features have been extracted. Then applied batch normalization and averagepool layer.

In the output layer, I have flattened the output of third layer and given it to the output layer with 10 neurons as number of classes.

Calculation of Output shape and parameters of each layers:

Layer1:
Input size = (1,16000)
Filter= 7
Padding=3
Stride=2
No of filters=128

$$output_{conv1} = \frac{16000 - 7 + 2 * 3}{2} + 1 \approx 8000$$

Output size of conv1= (128,8000)
Parameters in conv1= 7*1*128+128=1024

Output size of BatchNorm and Droupout will be same.
Parameters in BatchNorm1=2 (mean + variance) *128 = 256

Output after Maxpool1:

$$output_{maxpool1} = \frac{8000 - 2}{2} + 1 = 4000$$

Output of maxpool1 = (128,4000)

Layer2:
Input size = (128,4000)
Filter= 5
Padding=3
Stride=2
No of filters=64

$$output_{conv2} = \frac{4000 - 5 + 2 * 3}{2} + 1 \approx 2000$$

Output size of conv2= (64,2000)
Parameters of conv2= 5*1*128*64+64=41024

Output size of BatchNorm and Droupout will be same.
Parameters in BatchNorm2=2*64 = 128

Output after Maxpool2:

$$output_{maxpool2} = \frac{2000 - 2}{2} + 1 = 1000$$

Output of maxpool2 = (64,1000)

Layer3:
Input size = (64,1000)
Filter= 3
Padding=1
Stride=2
No of filters=32

$$output_{conv3} = \frac{1000 - 3 + 2 * 1}{2} + 1 \approx 500$$

Output size of conv3= (32,500)
Parameters of conv3= 3*1*64*32+32=6176

Output size of BatchNorm and Droupout will be same.
Parameters in BatchNorm3=2*32 = 64

Output after Maxpool3:

$$output_{maxpool3} = \frac{500 - 2}{2} + 1 = 250$$

Output of maxpool3 = (32,250)

Layer4 (Output Layer):
Output of the third layer get flattened and passed to the 10 neurons.

Parameters of the Output Layer: 32*250*10+10=80010

Total parameters: 1024+256+41024+128+6176+64+80010=128682

Architecture2: Convolutional-Transformer

In this architecture, a transformer has been designed which takes the input from the base architecture of CNN model (Architecture 1) and give it to the Transformer as input. The model Architecture of the Convolutional-Transformer is given below in Table 1.

Table 1: Convolutional Transformer Architecture

```
TransformerClassifier(
  (cnn_model): Custom_One_Dimension_ConvolutionModel(
    (conv1): Conv1d(1, 128, kernel_size=(7,), stride=(2,), padding=(3,))
    (batch_norm1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (dropout1): Dropout(p=0.5, inplace=False)
    (pool1): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv1d(128, 64, kernel_size=(5,), stride=(2,), padding=(2,))
    (batch_norm2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (dropout2): Dropout(p=0.5, inplace=False)
    (pool2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
    (conv3): Conv1d(64, 32, kernel_size=(3,), stride=(2,), padding=(1,))
    (batch_norm3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (dropout3): Dropout(p=0.5, inplace=False)
    (pool3): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
  )
  (transformer_encoder): Sequential(
    (0): CustomTransformerEncoderLayerWithCLS(
      (self_attn): CustomMultiheadAttention(
        (out_proj): Linear(in_features=250, out_features=250, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (linear1): Linear(in_features=250, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=250, bias=True)
      (norm1): LayerNorm((250,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((250,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
    )
    (1): CustomTransformerEncoderLayerWithCLS(
      (self_attn): CustomMultiheadAttention(
        (out_proj): Linear(in_features=250, out_features=250, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (linear1): Linear(in_features=250, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=250, bias=True)
      (norm1): LayerNorm((250,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((250,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
    )
  )
  (mlp_head): Linear(in_features=250, out_features=10, bias=True)
)
```

In this architecture, first CNN architecture has been defined till the third layer to extract the features from the audio signal of input size (1,16000). The output of CNN Architecture for the features is (32,250). This serves as input for the Transformer Classifier Architecture.
The Transformer Architecture contains cnn layers, the Two blocks of the Encoder Layers and single MLP layer for the output as it can be seen in Table 1.

The architecture of CNN layers has been discussed in the Architecture 1.

Each of the encoder layer has been made custom as it is required to add the CLS token into the input then give it to the multihead attention layer for calculating attention weights and attention output. In the multihead attention layer, all query, key, value processing has been done and attention weights and attention output has been returned. Then, attention weights get added

with the input of transformer with dropout and normalisation, and feeded to the multilayer perceptron. Also, Multihead Attention has a parameter of num_heads to pass the number of heads for each of the architecture.

The output of the encoder is then passed to the MLP_head to do the classification.

The number of parameters in the Tranformer Architecture is 2,607,778. The number of parameters is same for each of the head of transformer because same number of MLP layers has been used in each of the model.

Results:

Convolutional Neural Network Architecture:

The model has been trained in both ways, using K-Fold cross validation and Train-Test split method.
For the K-fold cross validation, the model has been trained for 100 epochs for each fold.
The best Training accuracy, Validation Accuracy, and Testing Accuracy along with Training Loss, validation Loss and Testing Loss has been given for the model.

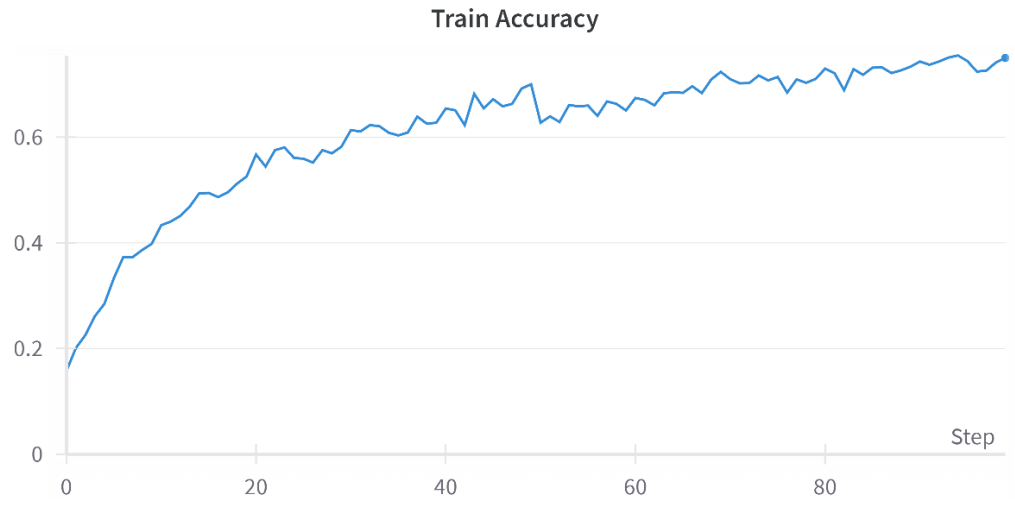


Fig. 1 Training Accuracy of CNN model

In the above Fig. 1, training accuracy curve we can observe that accuracy is getting increase by increasing the epochs and reached upto 75%.

Fig. 2 Training Loss of CNN model

In the above Fig. 2, from training loss curve it can be observed that loss is getting reduced by increasing the epochs and reached up to 0.75.

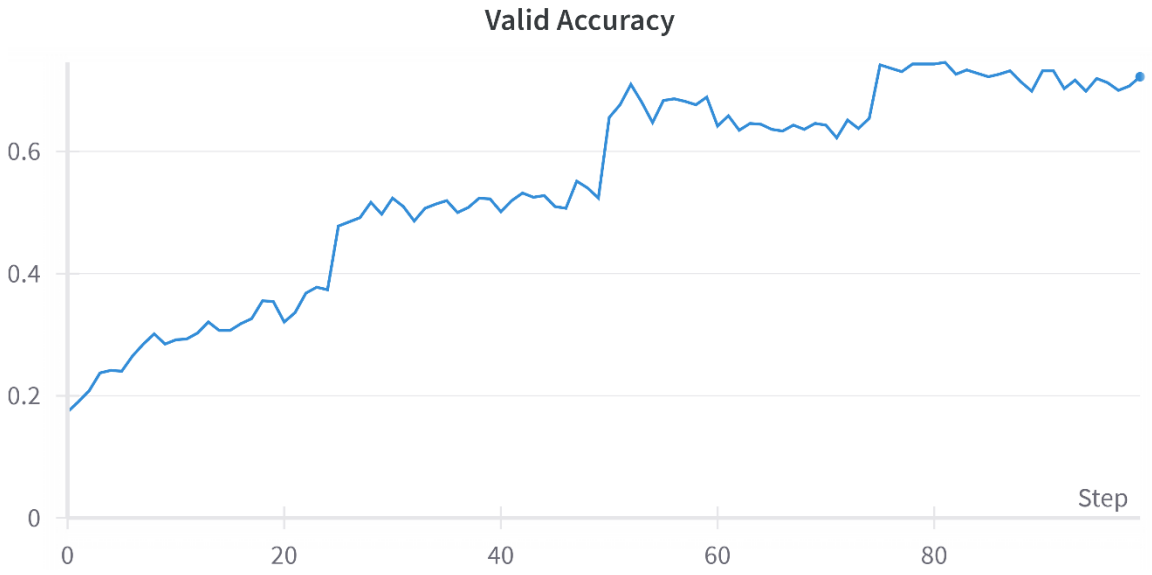


Fig. 3 Validation Accuracy of CNN model

In the Fig. 3, it can be observed that validation accuracy is increasing with increasing the number of epochs and reached up to 72%.

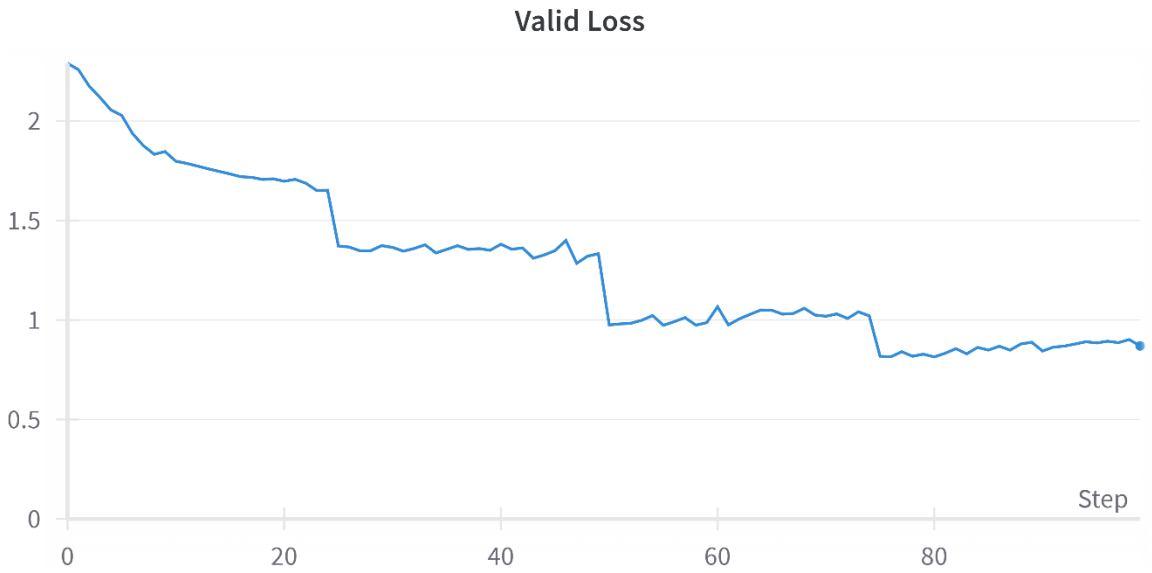


Fig. 4 Validation Loss of the CNN Model

The validation loss of the model is decreasing with increasing the number of epochs and reached upto 0.86.

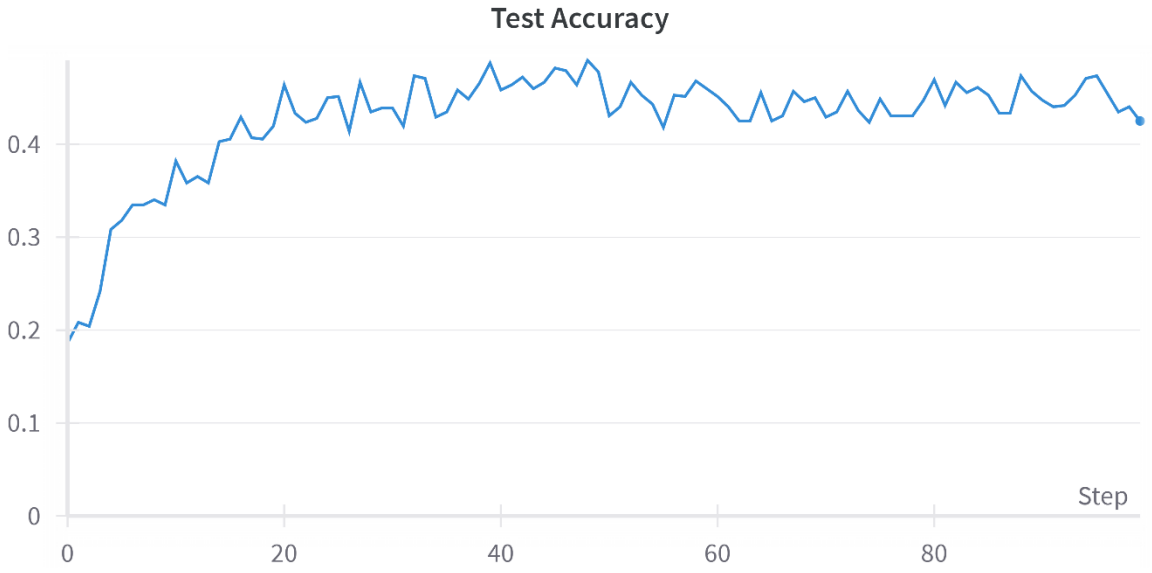


Fig. 5 Testing Accuracy of the CNN model

In Fig. 5, testing accuracy of the model is observed to be about 42%. On increasing the number of epochs, the testing accuracy is first increasing upto 40 epochs and then going to be constant with some variation.

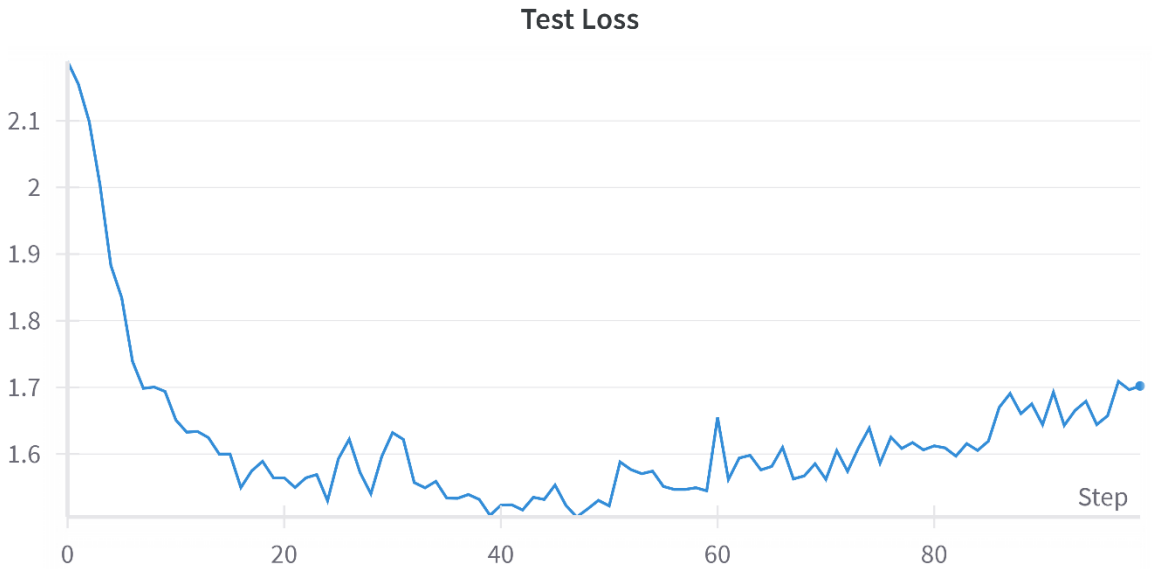


Fig. 6 Testing loss of the CNN model

In Fig. 6, test loss of CNN model has been mentioned, it is reducing upto 50 epochs and then increasing a slower rate and reached upto 1.7.

After performing the training and validation of the model. Predictions has been taken on the testing data. The confusion matrix regarding the test data for best accuracy the model is shown in Fig. 7.

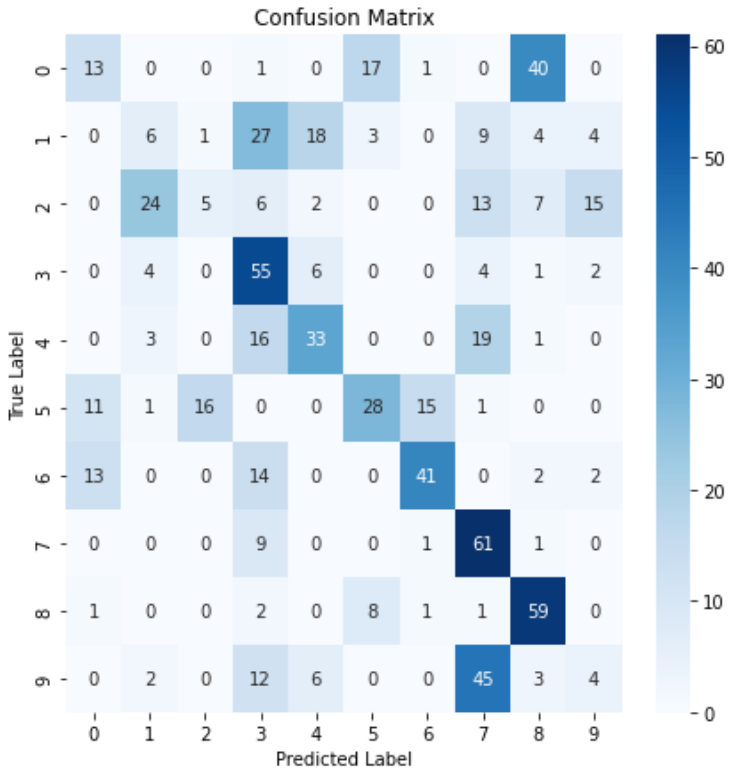


Fig. 7 Confusion Matrix of the CNN model

In the confusion matrix, for each of the class there are 720 values due to the testing data contains the 80 instances which has been later sampled into 9 channels and corresponding labels has been generated. So, total instances have become 80*9=720. It can be observed into

the matrix that maximum true positive predicted labels have been observed into class 3, 6, 7, and 8. Also, the falsest predicted levels can be observed into class 8, 7, and 3.
Accuracy of the model can be given as:

$$Accuracy = \frac{13+6+5+55+33+28+41+61+59+4}{\begin{array}{c}13+1+17+1+40+6+1+27+18+3+9+4+4+24+5+6+2\\+13+7+15+4+55+6+4+1+2+3+16+33+19+1+11+1\\+16+28+15+1+13+14+41+2+2+9+1+61+1+1+2+8+1+1\\+59+2+12+6+45+3+4\end{array}}$$

$$Accuracy = \frac{305}{720} = 0.42$$

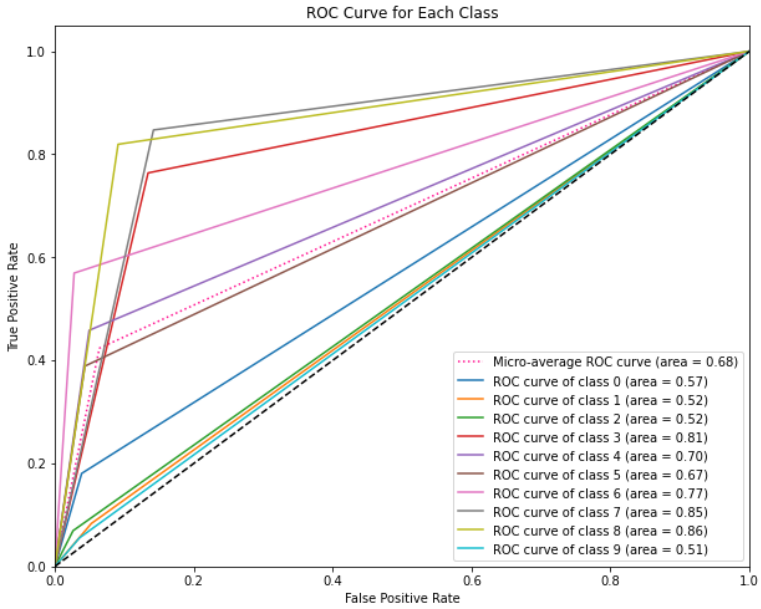


Fig. 8 ROC Curve for the CNN model

The AUC-ROC curve is shown in Fig. 8. The area under this curve (AUC) is approximately 0.68. From the graph, it can be observed that, class 3, 6, 7 and 8 has been maximum which can be also observed in the confusion matrix which verifies the accuracy for each of the class.

The F1 score corresponding to each of the class is given below:
[0.23636364 0.10714286 0.10638298 0.51401869 0.48175182 0.4375 0.6259542 0.54222222 0.62105263 0.08080808]

For the train test split method, data is fold=1 has been taken into testing and fold=3,4,5 has been taken into the training and run for 100 epochs. Due to some issue, the accuracy during the testing is not observed good earlier it was growing.

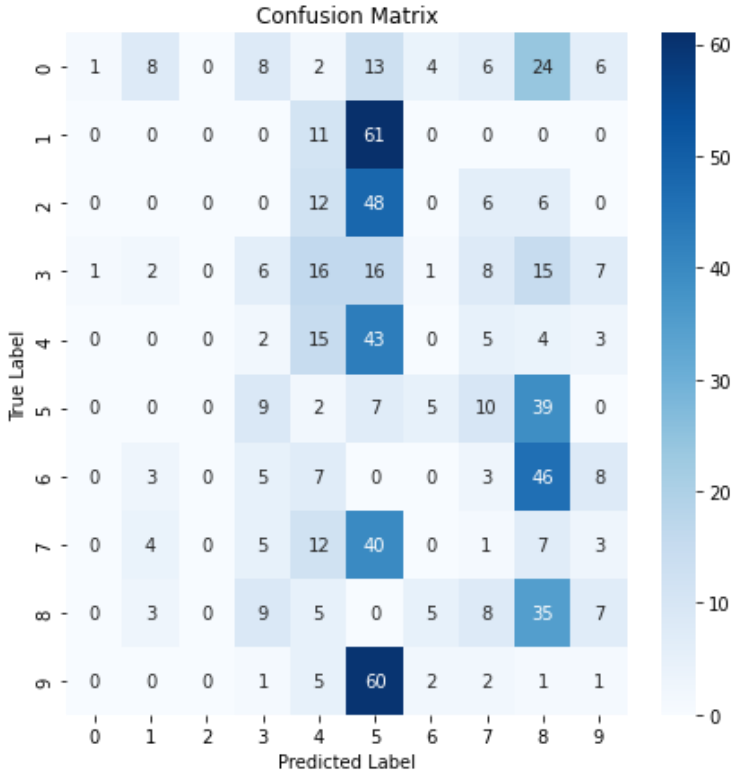


Fig. 9 Confusion Matrix for CNN with Test Split

The accuracy which is achieved during the Test Split configuration of CNN is 9.16% as can be seen in Fig. 9.

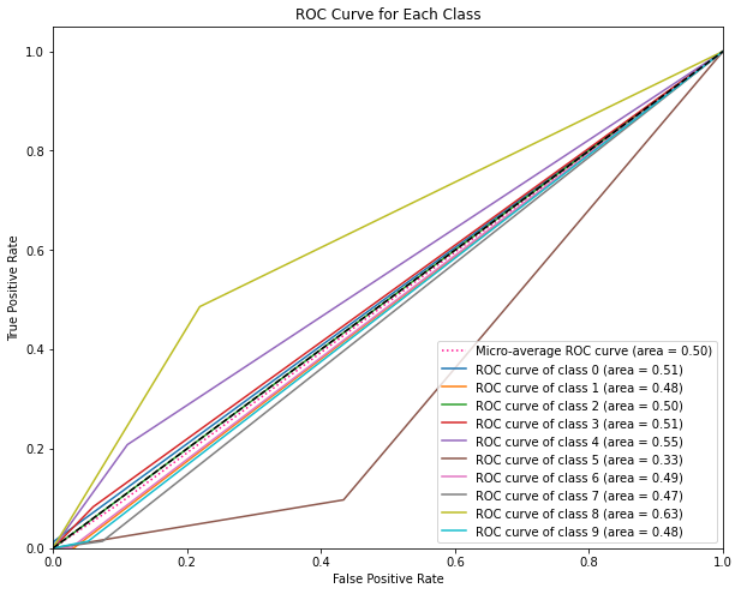


Fig. 10 ROC-AUC Curve for CNN with test Split

In Fig. 10, we can observe that highest accuracy class during Test split configuration is class 8 and lowest is observed for the class 5.

Convolutional Transformer with Single Head:

With single head, the model was not able to achieve a good accuracy. Only 10 percent accuracy has been achieved on both training, validation, and testing as it can be observed in the Fig. 12,

14 and 16 respectively. By increasing the epochs, the loss has be observed to fluctuating around zeros in each of the case as can be seen in Fig. 11, 13 and 15 for Training loss, Validation loss and Testing loss.

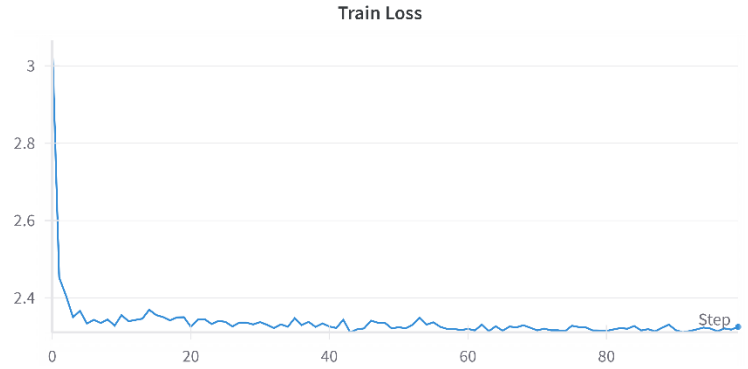


Fig. 11 Training Loss for 1Head Transformer with K-Fold

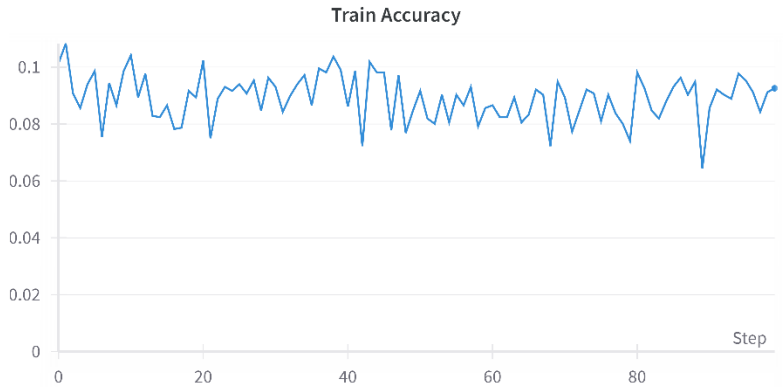


Fig. 12 Training Accuracy for 1Head Transformer with K-Fold

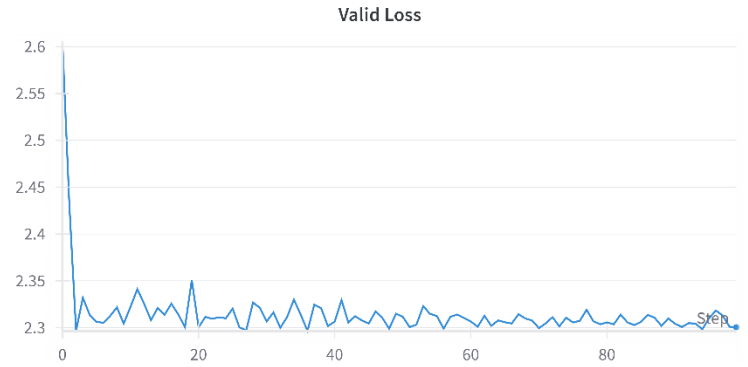


Fig. 13 Validation Loss for 1Head Transformer with K-Fold

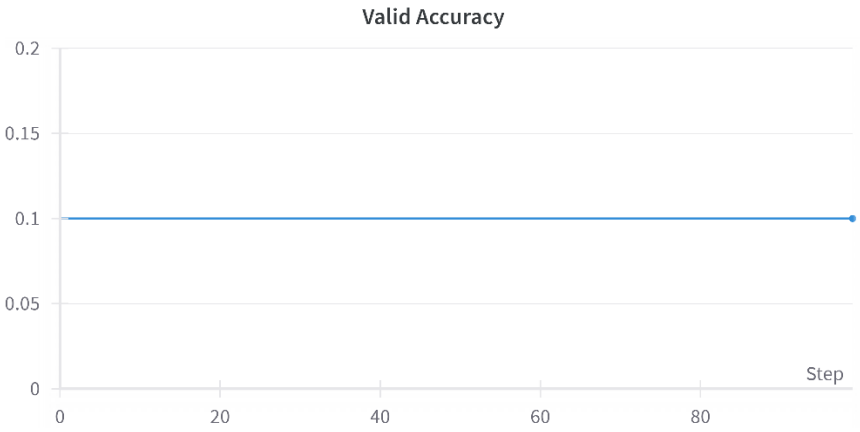


Fig. 14 Validation Accuracy for1Head Transformer with K-Fold

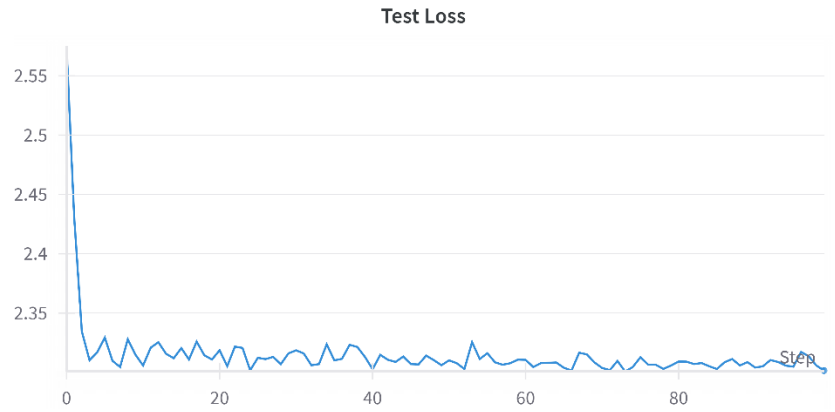


Fig. 15 Testing Loss for1Head Transformer with K-Fold

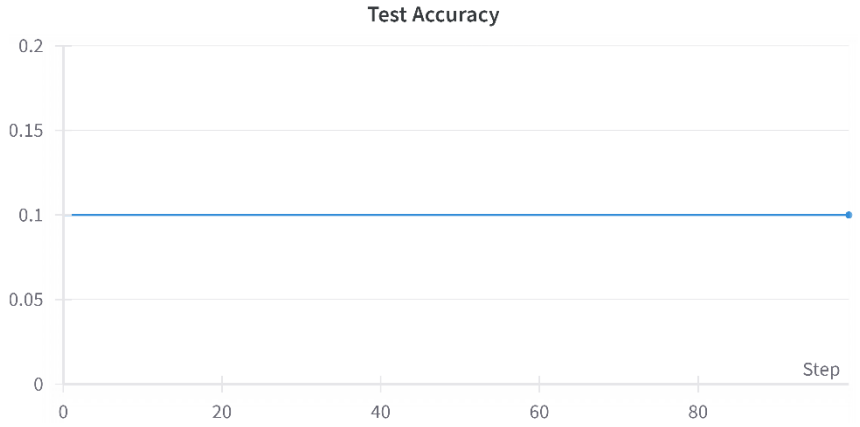


Fig. 15 Testing Accuracy for 1Head Transformer with K-Fold

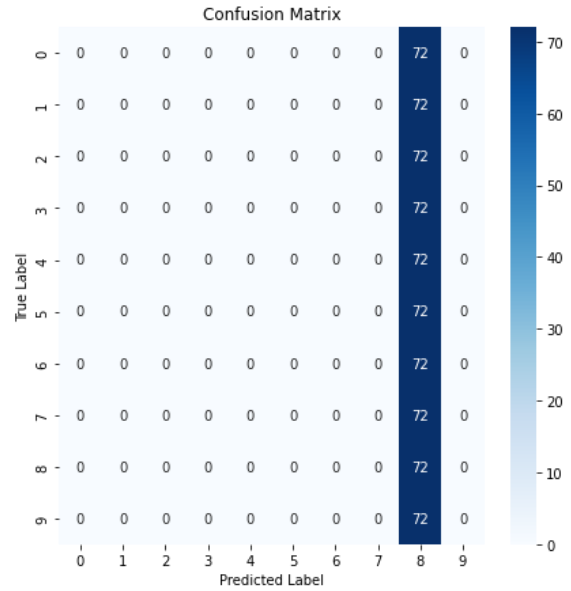


Fig. 16 Confusion matrix for 1Head Transformer with K-Fold

Fig. 16 shows the confusion matrix for the 1 head transformer with k-flod, it can be observed that the model is predicting all the values towards class 8 which shows that high biasness there of the towards class 8.

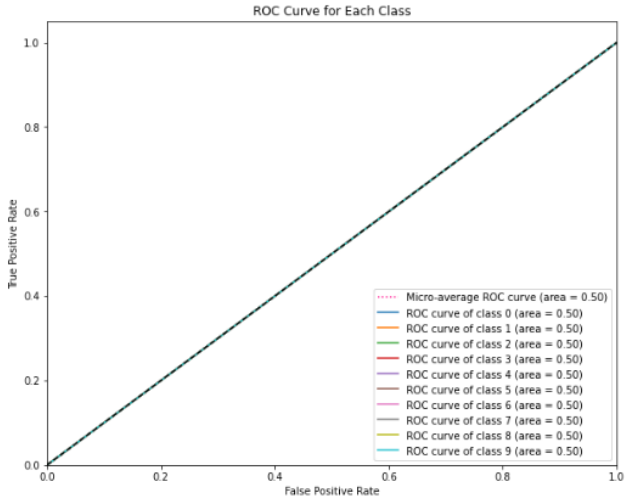


Fig. 17 ROC-AUC curve for 1Head Transformer with K-Fold

In the Fig. 17, ROC-AUC Curve shows that there 0.50 area for each of the class.

F1 score for 1head Transformer with k-fold is given as [0. 0. 0. 0. 0. 0. 0. 0. 0.18181818 0.]

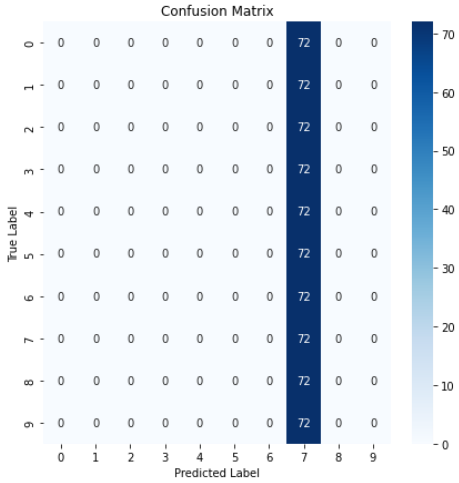


Fig. 18 Confusion matrix for 1Head Transformer with Test Split

In the Fig. 18, confusion matrix has been drawn for the model 1head transformer, it shows a prediction for class 7 only which shows a biasness towards class 7.

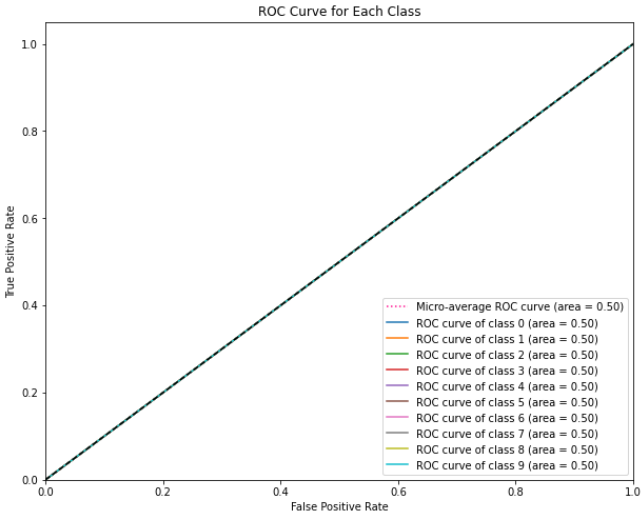


Fig. 19 ROC-AUC curve for 1Head Transformer with Test Split

In Fig. 19, ROC-AUC curve for 1Head Transformer with Test Split configuration has been drawn.

Convolutional Transformer with Two Heads:

With two head, the model was not able to achieve a good accuracy. Only 10 percent accuracy has been achieved on both training, validation, and testing as it can be observed in the Fig. 20, 25 and 23 respectively. By increasing the epochs, the loss has be observed to fluctuating around zeros in each of the case as can be seen in Fig. 21, 24 and 22 for Training loss, Validation loss and Testing loss.

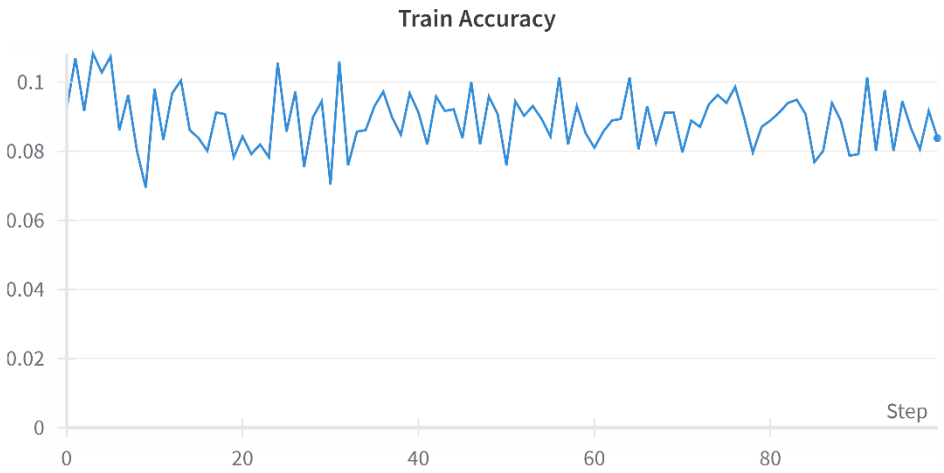


Fig. 20 Training Accuracy of the 2Head Transformer with k-fold

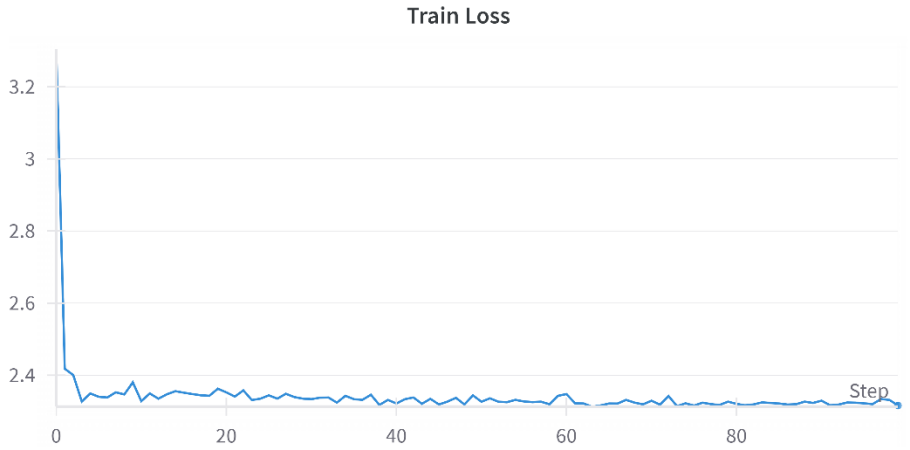


Fig. 21 Training loss of the 2Head Transformer with k-fold

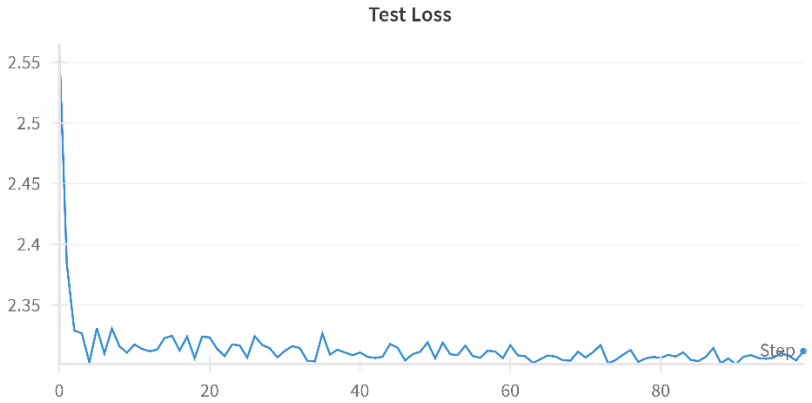


Fig. 22 Testing Loss of the 2Head Transformer with k-fold

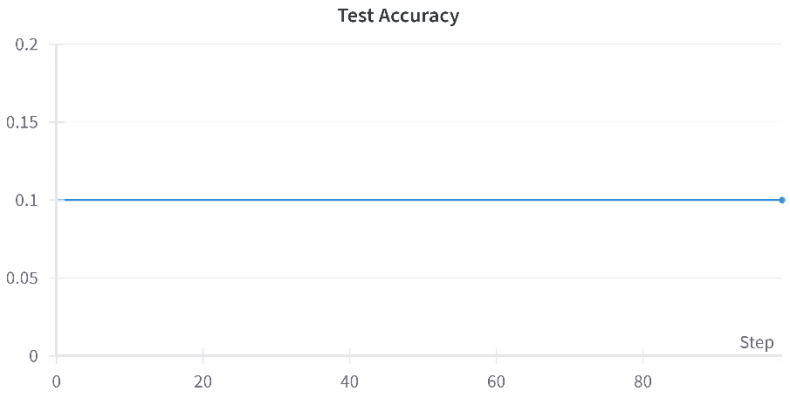


Fig. 23 Testing Accuracy of the 2Head Transformer with k-fold

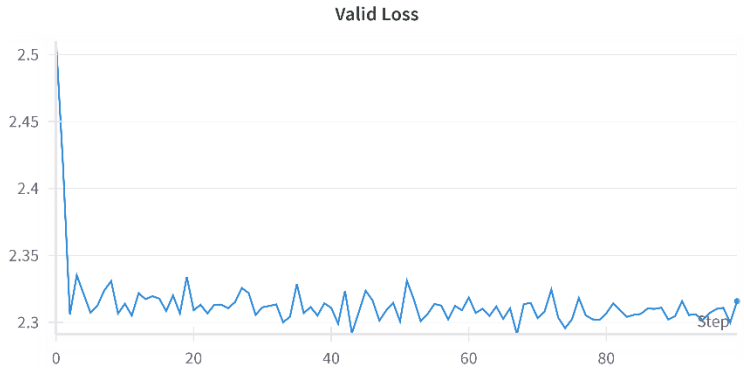


Fig. 24 Valid Loss of the 2Head Transformer with k-fold

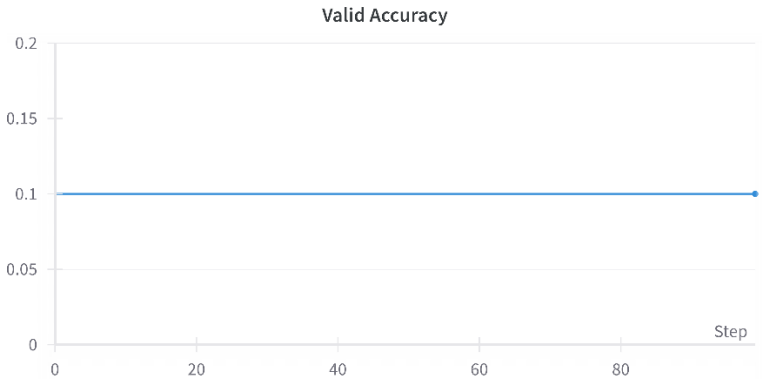


Fig. 25 Valid Accuracy of the 2Head Transformer with k-fold

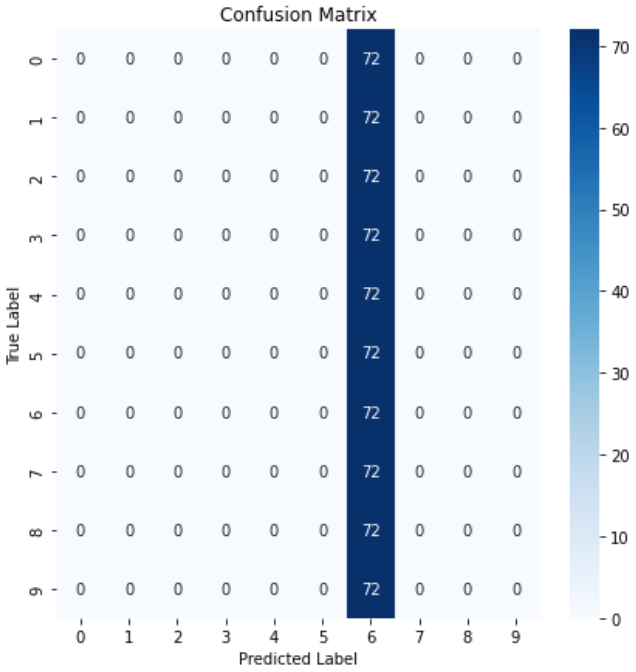


Fig. 26 Confusion matrix of the 2Head Transformer with k-fold

Fig. 26 shows the confusion matrix for the 2 head transformer with k-flod, it can be observed that the model is predicting all the values towards class 6 which shows that high biasness there of the towards class 6.

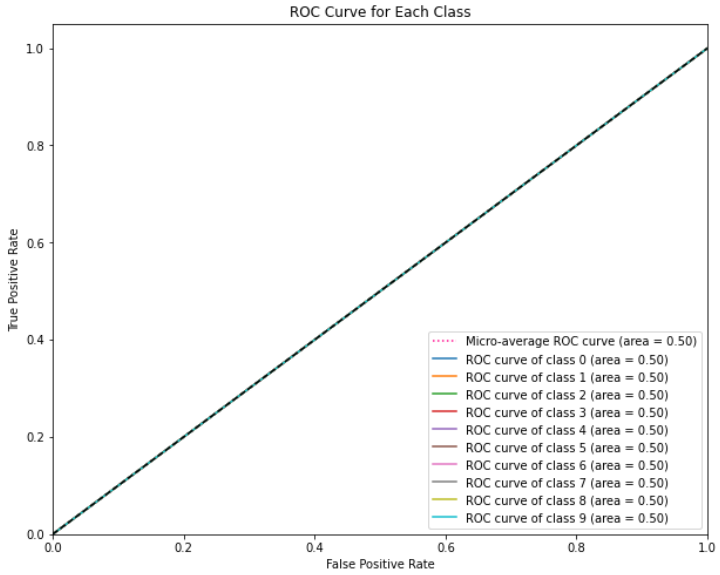


Fig. 27 ROC-AUC curve of the 2Head Transformer with k-fold

In the Fig. 27, ROC-AUC Curve shows that there 0.50 area for each of the class.

F1 score for 1head Transformer with k-fold is given as [0. 0. 0. 0. 0. 0. 0.18181818 0. 0. 0.]

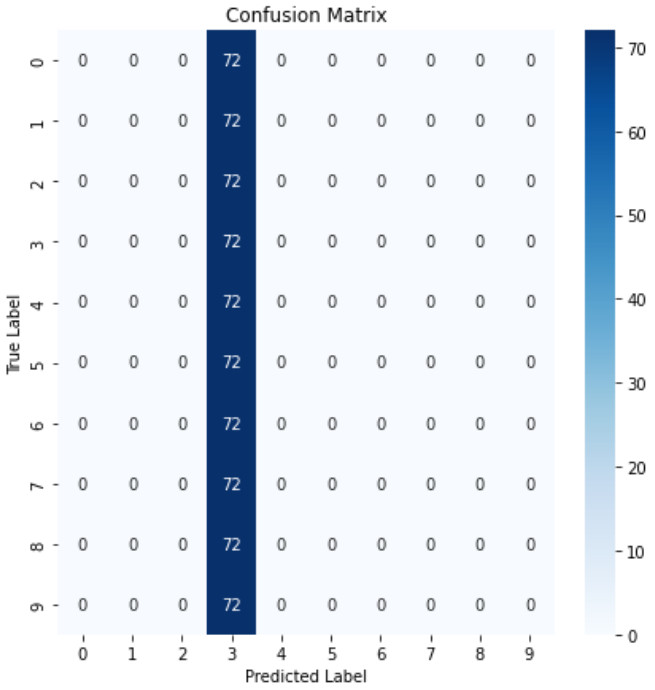


Fig. 28 Confusion matrix of the 2Head Transformer with Test Split

In the Fig. 28, confusion matrix has been drawn for the model 2head transformer, it shows a prediction for class 7 only which shows a biasness towards class 3.

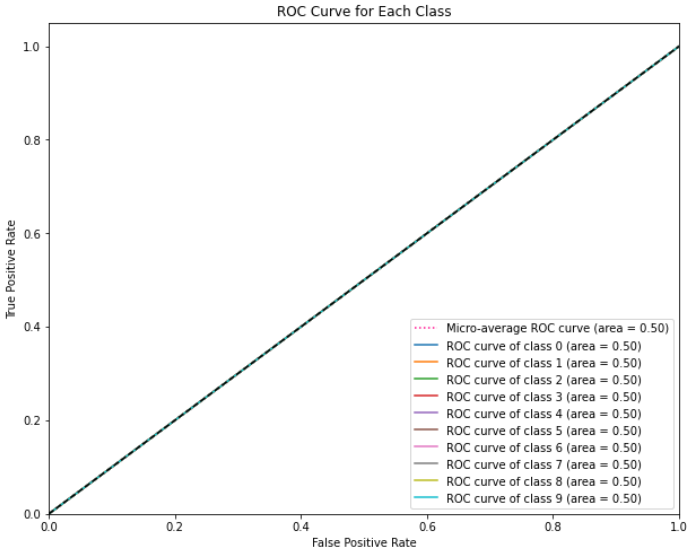


Fig. 29 ROC-AUC curve of the 2Head Transformer with Test Split

In Fig. 29, ROC-AUC curve for 2Head Transformer with Test Split configuration has been drawn.

Convolutional Transformer with Four Heads:

With four head, the model was not able to achieve a good accuracy. Approximately, 10 percent accuracy has been achieved on both training, validation, and testing as it can be observed in the Fig. 30, 35 and 33 respectively. By increasing the epochs, the loss has be observed to fluctuating around zeros in each of the case as can be seen in Fig. 31, 34 and 32 for Training loss, Validation loss and Testing loss.

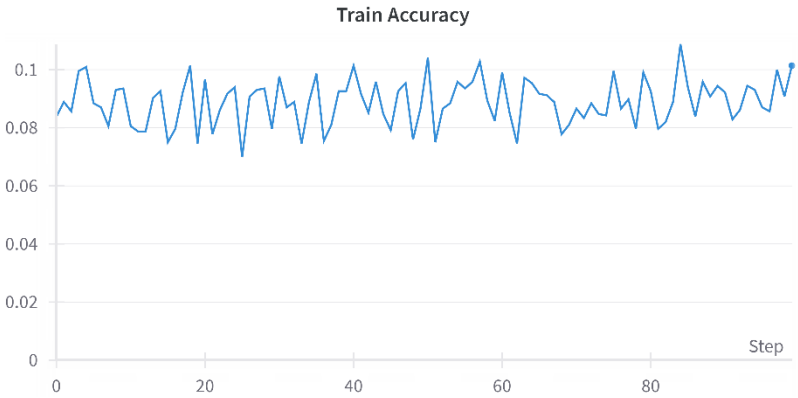


Fig. 30 Training Accuracy of the 2Head Transformer with k-fold

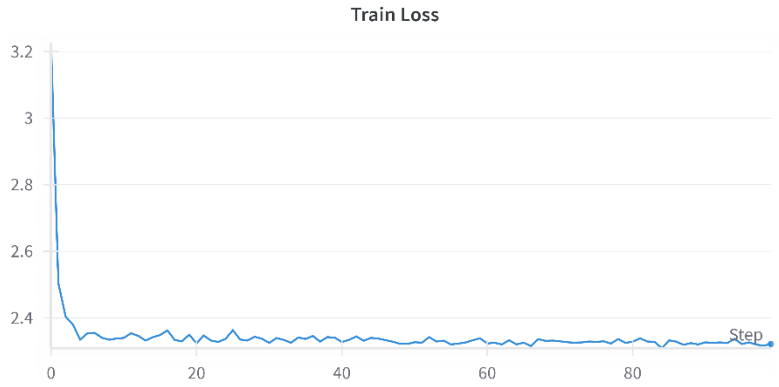


Fig. 31 Training Loss of the 2Head Transformer with k-fold

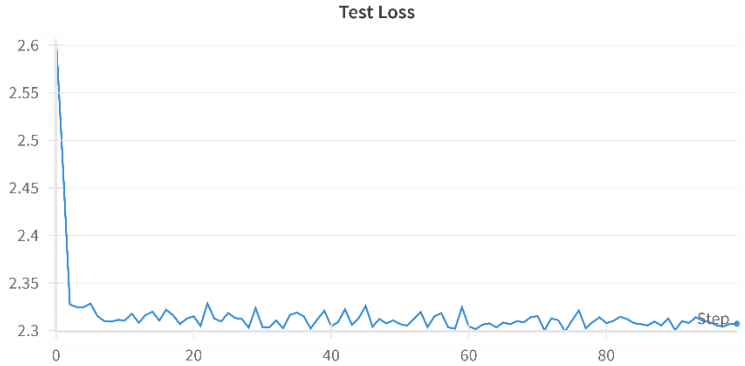


Fig. 32 Test Loss of the 2Head Transformer with k-fold

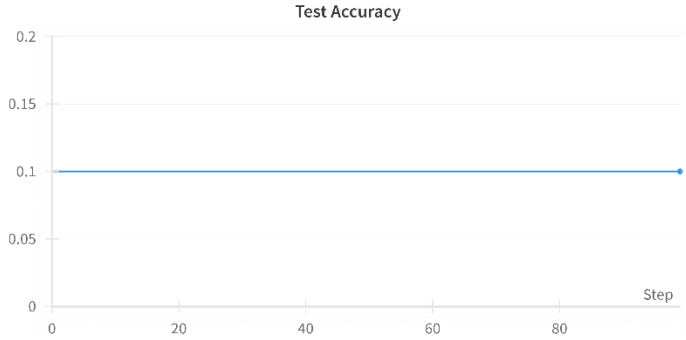


Fig. 33 Test Accuracy of the 2Head Transformer with k-fold

Fig. 34 Valid Loss of the 2Head Transformer with k-fold

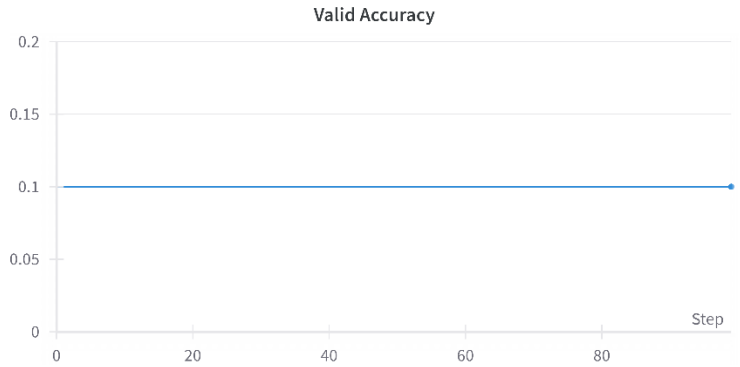


Fig. 35 Valid Accuracy of the 2Head Transformer with k-fold

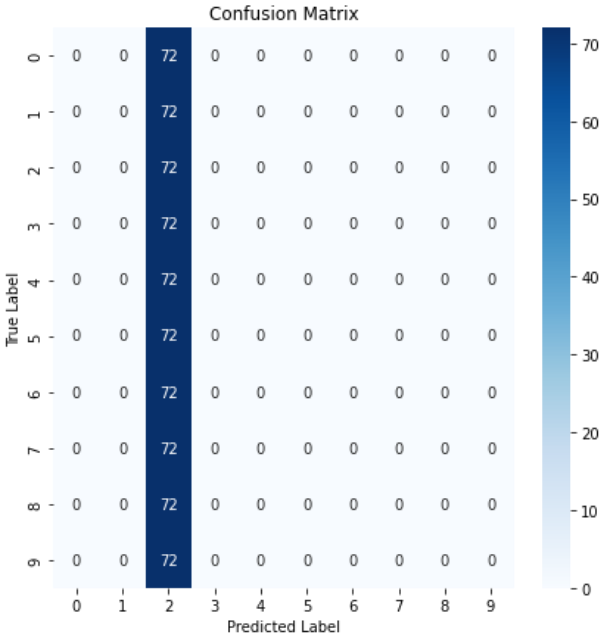


Fig. 36 Confusion matrix of the 4Head Transformer with k-fold

Fig. 36 shows the confusion matrix for the 4 head transformer with k-flod, it can be observed that the model is predicting all the values towards class 6 which shows that high biasness there of the towards class 2.

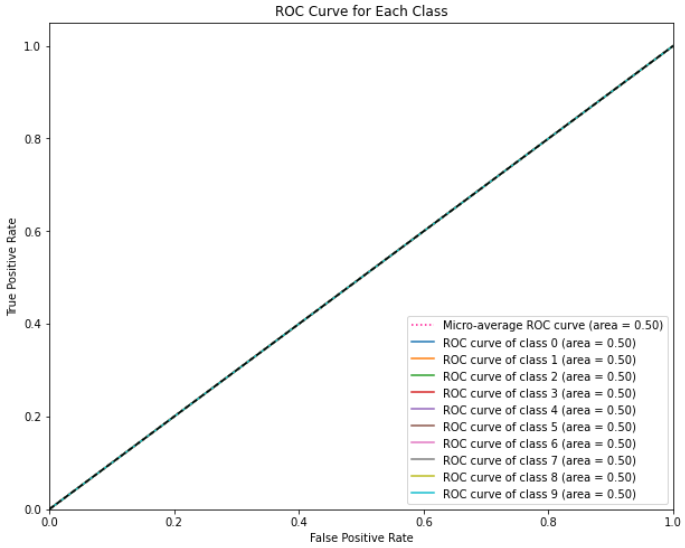


Fig. 37 ROC-AUC curve of the 2Head Transformer with k-fold

In the Fig. 37, ROC-AUC Curve shows that there 0.50 area for each of the class.

F1 score for 1head Transformer with k-fold is given as

[0. , 0. , 0.18181818, 0. , 0. ,0. , 0. , 0. , 0. , 0.]

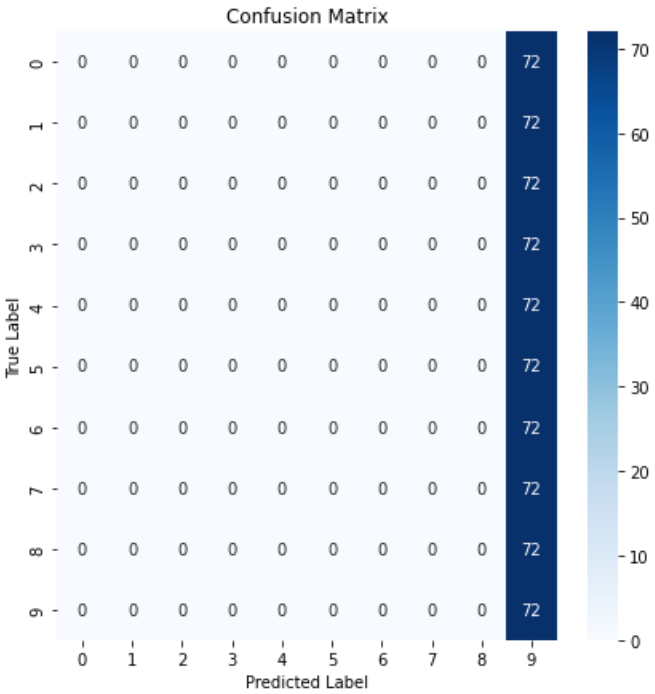


Fig. 38 Confusion matrix of the 4Head Transformer with Test Split

In the Fig. 38, confusion matrix has been drawn for the model 4head transformer, it shows a prediction for class 7 only which shows a biasness towards class 9.

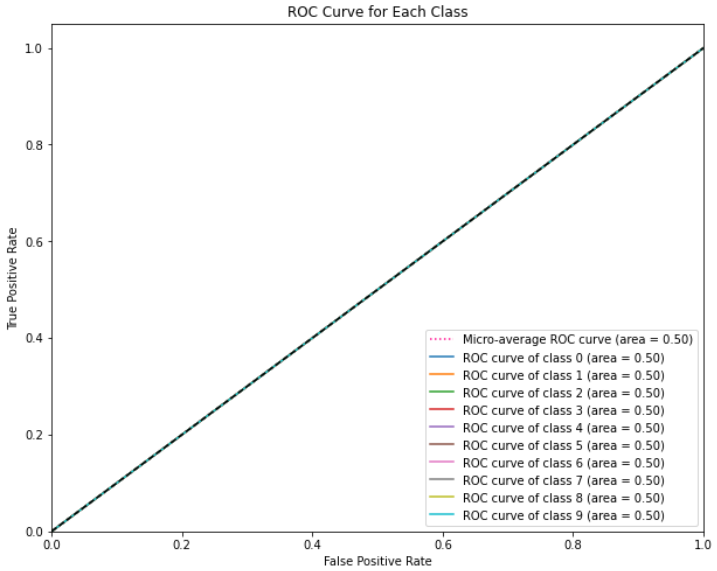


Fig. 39 ROC-AUC curve of the 2Head Transformer with Test Split

In Fig. 39, ROC-AUC curve for 2Head Transformer with Test Split configuration has been drawn.

Discussion and Conclusions:

In this task, many things have been inculcated to learn about the deep learning like CNN and Transformer architecture for the classification of audio. Also, concept of K-fold and Test Split configuration has been used for training the model. Along with F1 score, accuracy score, ROC-AUC curve etc. The accuracy with transformer is not achieved good in my case and with the CNN, only 42% testing accuracy has been achieved. But the training and validation accuracy of the CNN model is satisfactory. Also, since the CNN model is not able to achieve good accuracy during the testing the model seems to be overfitted. Also, I have tried the Xavier_uniform initialization for the encoder in the transformer but it seems that also not worked for achieving the good accuracy. In future, it seems to review the model of transformer and work on achieving the good accuracy of the same data.