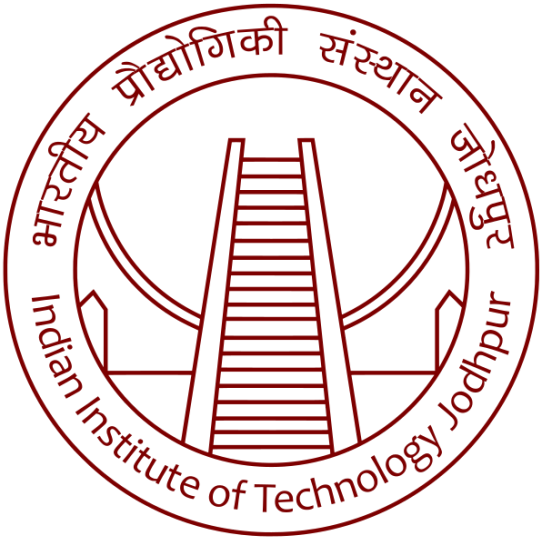
Deep Learning

Assignment 2



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Submitted by

Ratnesh Kumar Tiwari
(M23MAC011)

To

Dr. Deepak Mishra

Please visit the code for more information:
https://colab.research.google.com/drive/16mHnGGuIgD4eN7_BLhF627tuIG39jLk4?usp=sharing

1. Introduction:

The Environmental Sound Classification 10 (ESC10) dataset is a collection of environmental sounds with 10 distinct categories, primarily designed for audio classification tasks. Let us have a comprehensive overview of the ESC10 dataset, detailing its characteristics, loading procedure, and preprocessing steps

2. Dataset Overview:

The ESC10 dataset is derived from the larger ESC-50 dataset, which contains 50 sound classes. It comprises 10 classes of environmental sounds, including:

1. Dog	3. Pig	5. Frog	7. Hen	9. Sheep
2. Rooster	4. Cow	6. Cat	8. Insects	10. Crow

The dataset includes annotations specifying the fold number for cross-validation and whether the sound belongs to the ESC10 subset.
Each sound sample is provided as a separate audio file, with file names indicating the category and unique identifier.

3. Loading Procedure:

Parameters: Key parameters such as data directory, dataframe, fold information, and sampling rates are provided to instantiate the dataset object.

Data Filtering: The dataset is filtered based on the fold split for train, validation, or test sets. Additionally, filtering is applied based on the ESC10 flag to ensure only relevant samples are included.

Category Mapping: Categories are enumerated and mapped to integer indices for classification.

File Loading: Audio files are loaded using the torchaudio library, and resampled to a new sampling rate if necessary.

4. Preprocessing Steps:

Resampling: Audio files are resampled using torchaudio.transforms.Resample to ensure a consistent sampling rate across all samples.

Windowing: Audio signals are split into overlapping segments using a rolling window approach. The window size and step size are adjusted based on the desired sample length, with overlap to capture temporal information.

5. Experiment Setup:

To conduct experiments using the ESC10 dataset, the following setup is proposed:

Test Set: The parameter test_samp is set to 1, indicating that only a first fold will be used as test.
Validation Set: The parameter valid_samp is set to 3, allowing for 3rd fold to be validation set and rest as training set.
Batch Size: A batch size of 40 is selected, which is a commonly used value for training neural networks. Adjustments to this parameter can be made based on hardware limitations and model performance.
Custom Data Module: The CustomDataset class is instantiated with the provided parameters:
data_directory: Specifies the directory containing the audio files.
data_frame: Refers to the DataFrame containing metadata and annotations for the dataset.
esc_10_flag: Set to True to filter the dataset to include only samples from the ESC10 subset.
file_column: Specifies the column name in the DataFrame containing file paths.
label_column: Specifies the column name containing the category labels.
sampling_rate: Original sampling rate of the audio files (44100 Hz).
new_sampling_rate: Desired sampling rate for preprocessing (16000 Hz).
sample_length_seconds: Specifies the new length of input samples in seconds (2 second).

6. Execution and Evaluation:

After configuring the dataset and model training pipeline, experiments can be conducted to train and evaluate classification models using the ESC10 dataset.
The dataset will be automatically split into training and validation sets based on the specified folds, ensuring proper evaluation of model performance.

7. Architecture 1 (1D CNN):

The ConvNet architecture used in this assignment is designed for environmental sound classification using Convolutional Neural Networks (CNNs). Let us elaborate the architecture's components and their functionalities:

Convolutional Layers: Three convolutional layers are employed to extract hierarchical features from input audio.
Each convolutional layer consists of 1-dimensional convolutions followed by rectified linear unit (ReLU) activation functions and batch normalization.
The kernel sizes and strides of the convolutional layers decrease and increase progressively, allowing the network to capture features at multiple scales.
Pooling Layers: Three max-pooling layers are applied to reduce the spatial dimensions of feature maps while preserving important information.
Max-pooling operations with same kernel sizes and strides are utilized to downsample feature maps.
Fully Connected Layers: Two fully connected (dense) layers are employed for classification.
The output of the last convolutional layer is flattened and passed through two fully connected layers with ReLU activation functions and dropout regularization.
The final fully connected layer produces the logits for classification into the target number of classes.
Activation Functions and Regularization: ReLU activation functions are utilized throughout the network to introduce non-linearity.
Dropout regularization with a dropout probability of 0.25 is applied to the output of the first two fully connected layers to prevent overfitting.
Batch Normalization: Batch normalization layers are inserted after each convolutional layer to stabilize and accelerate the training process.

8. Architecture 2 (Transformer):

The Transformer architecture presented below is designed for environmental sound classification using self-attention mechanisms. The architecture created is dynamic in nature. It accepts the number of classes, number of heads, number of layers as parameters. This architecture can be used to classify using a user defined number of classes, heads and layers which gives more flexibility.

Convolutional Preprocessing Layers: Three convolutional layers are employed initially to extract features from the input audio spectrograms.
Each convolutional layer is followed by batch normalization and ReLU activation functions to introduce non-linearity and stabilize training.
Max-pooling operations are applied to reduce the spatial dimensions of feature maps.
This is the same baseline architecture that we have seen above.

The output volume obtained is in the form of (Batch Size , Output Channels, Output Seq. Length)
The output volume obtained is directly passed in the Transformer Encoder Block.

Transformer Encoder Blocks:

Positional Encoding: A CLS Token vector of the same dimension as **Output Channels** is firstly added at the front of inputs of each batch for classification purpose. Positional encodings are added to these new input embeddings to provide positional information to the Transformer model. This step is done before feeding the input to the multihead self attention layer.
Sine and cosine functions are utilized to encode the positions of tokens in the input sequence, ensuring the model can distinguish between tokens at different positions.

1. Multi-Head Self-Attention Mechanism: Each Transformer block includes a multi-head self-attention mechanism to capture contextual information from the input sequence.
Inputs are then linearly projected in smaller dimensions (in the factor of number of heads) as keys , queries and values to perform self-attention computations efficiently. This operation is performed as many times as the number of heads are there. This Multiple attention heads allow the model to focus on the input in different aspects simultaneously. Then attention is calculated using:

$$\textit{Attention}(Q, K, V) = \textit{Softmax}(QK/\sqrt{d})\, V$$

Here, Q is Query
K is Keys
V is Value
d is output channel

Now the output of each attention head is concatenated resulting in the same dimension of output before performing the self attention.

2. Residual Connection: Now a residual connection is added from input to output of the multihead attention block to promote better flow of gradient and avoid vanishing gradient problem.

3. Layer Normalization: Layer Normalization is applied at each output vector individually facilitating stable training.

4. Multilayer Perceptron : Now, the output vector is passed individually through a MLP Layer to introduce non-linearity and enable the model to learn complex patterns.
The MLP Layer consists of two linear transformations separated by a ReLU activation function, followed by dropout regularization to prevent overfitting.

5. Residual Connection: The input and output of the MLP layer is again joined with a residual connection.

6. Layer Normalization: Layer Normalization is again applied at each output vector individually.

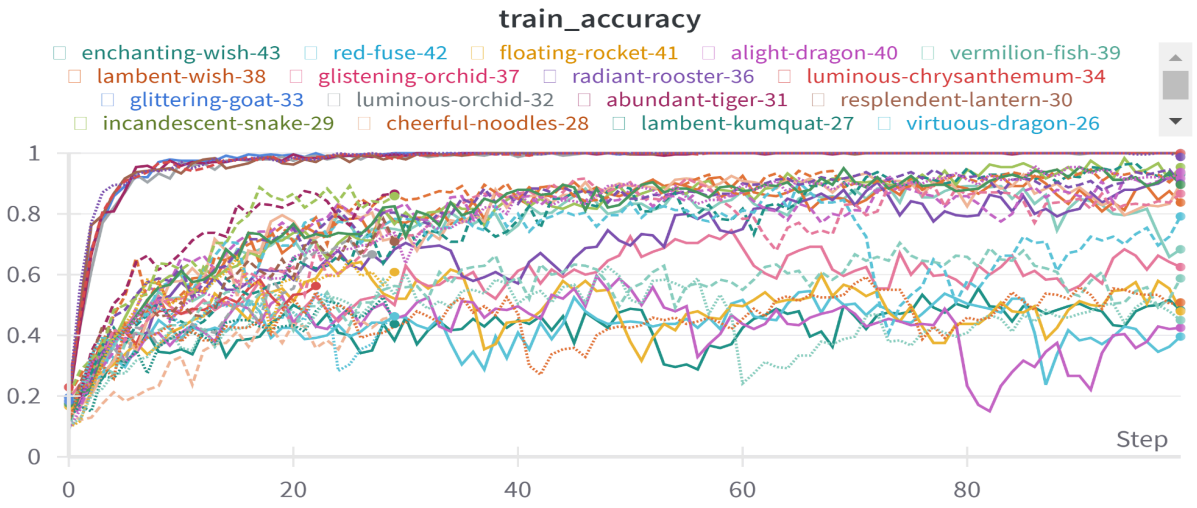
Multiple Encoder Layers: The Transformer architecture consists of multiple encoder layers stacked sequentially consisting of all steps (1 to 6 mentioned above), allowing for the hierarchical extraction of features from the input sequence.
Each encoder layer repeats the same structure, enabling the model to learn increasingly abstract representations as information flows through the network.

Classification Head: At the end, we extract the first output of each batch as we have added a CLS Token initially. This was followed by positional encoding which ensured the position of every input including CLS Token.

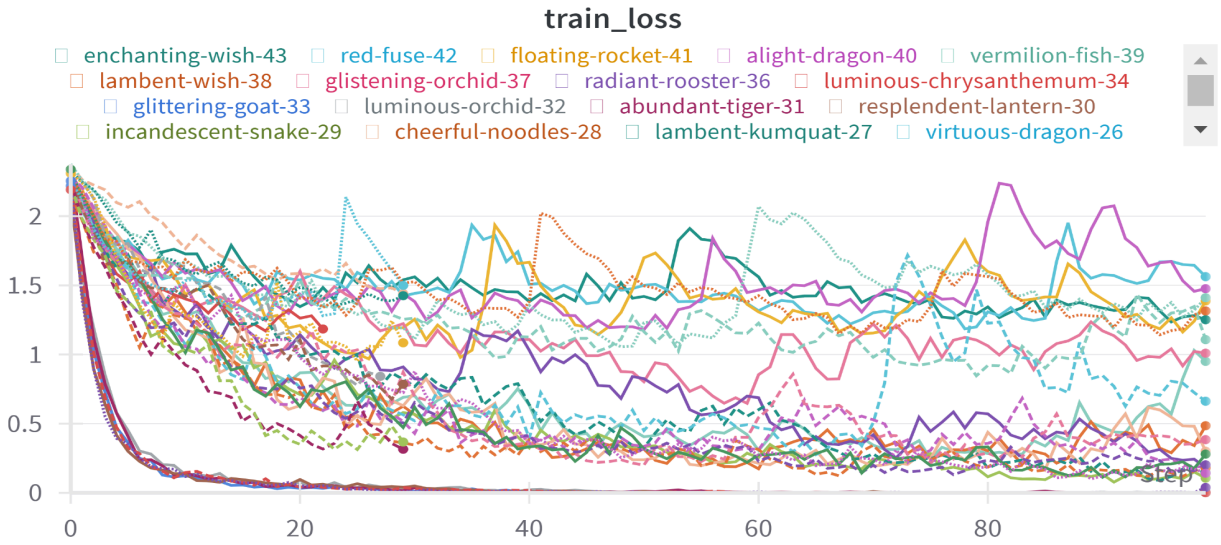
Fully Connected Layer: The extracted CLS Tokens from each batch is a D dimensional vector, where D is the number of channels in the input of Transformer Encoder. It is then passed through a fully connected layer with ReLU activation to produce the final classification logits.
Dropout regularization is applied to the output of the fully connected layer to prevent overfitting during training.

Note: Optimizer used in calculating the result is Adam Optimizer with learning rate = 0.001.

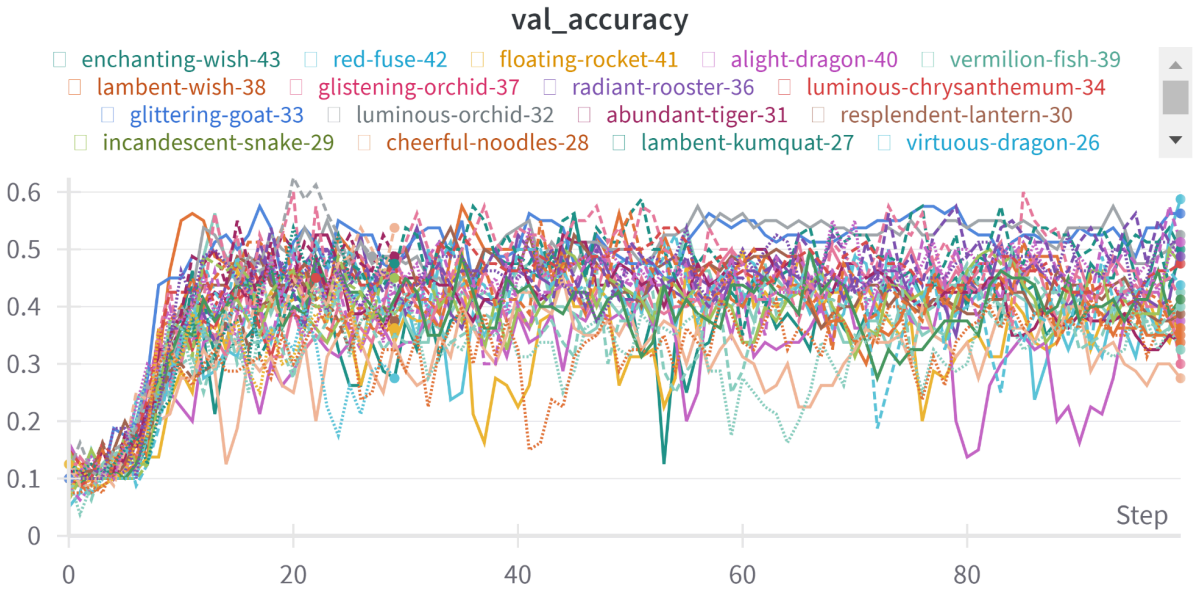
Results:
Plots of training accuracies per epoch of complete architectures.



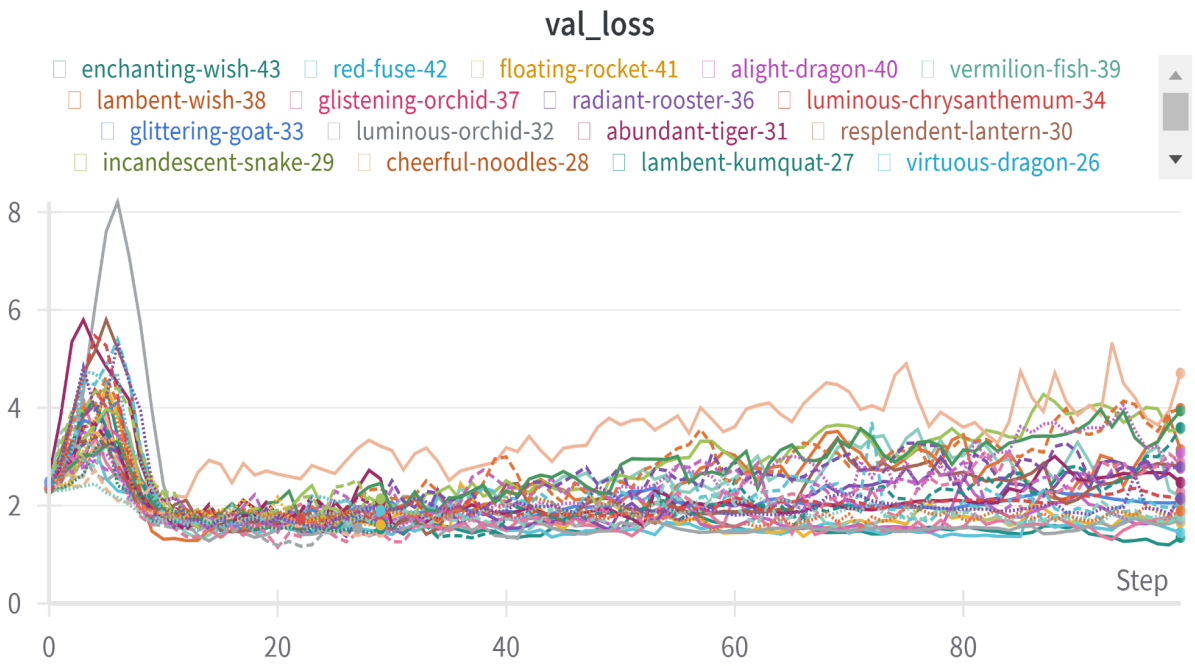
Plots of training losses per epoch of complete architectures.



Plots of Validation Accuracies per epochs:



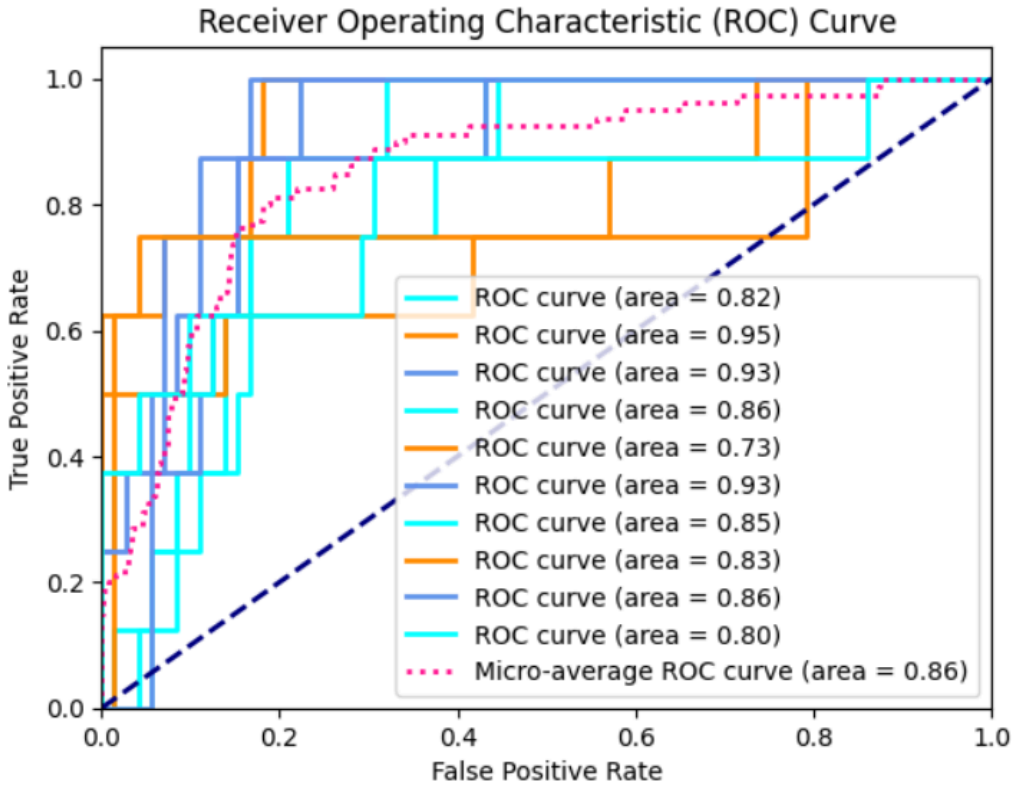
Plots of Validation loss per epochs of all architecture:



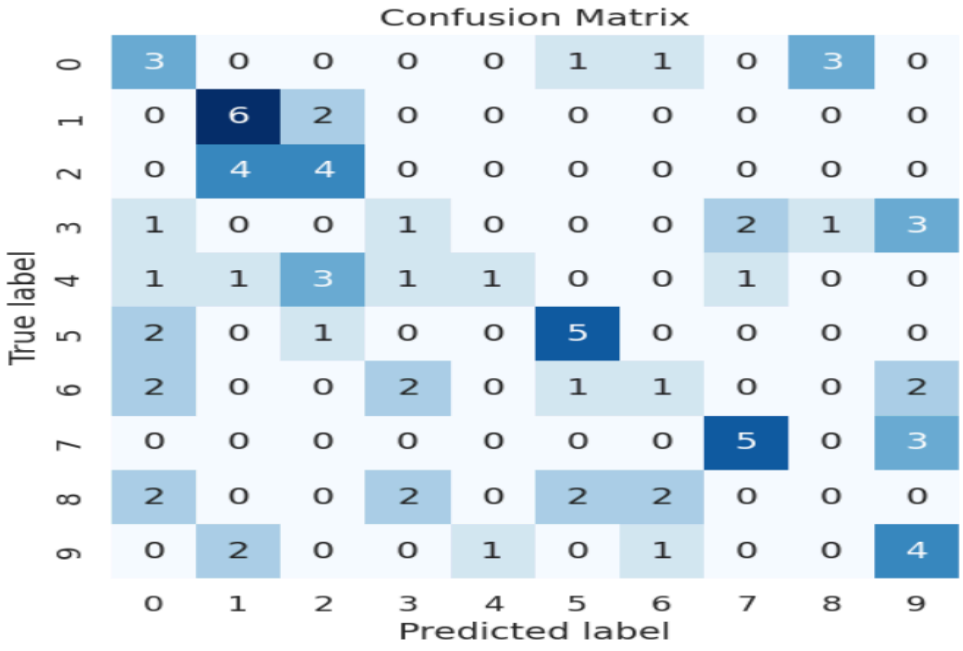
1. Using Standard Train and Test Split: The dataset provided contains fold attributes in range of 1 to 5. Using that, all samples, whose fold value is 1, are selected as a testing set. Out of the remaining, a random value is selected for the validation set and the rest is selected as a training set.

a. 1D-CNN:

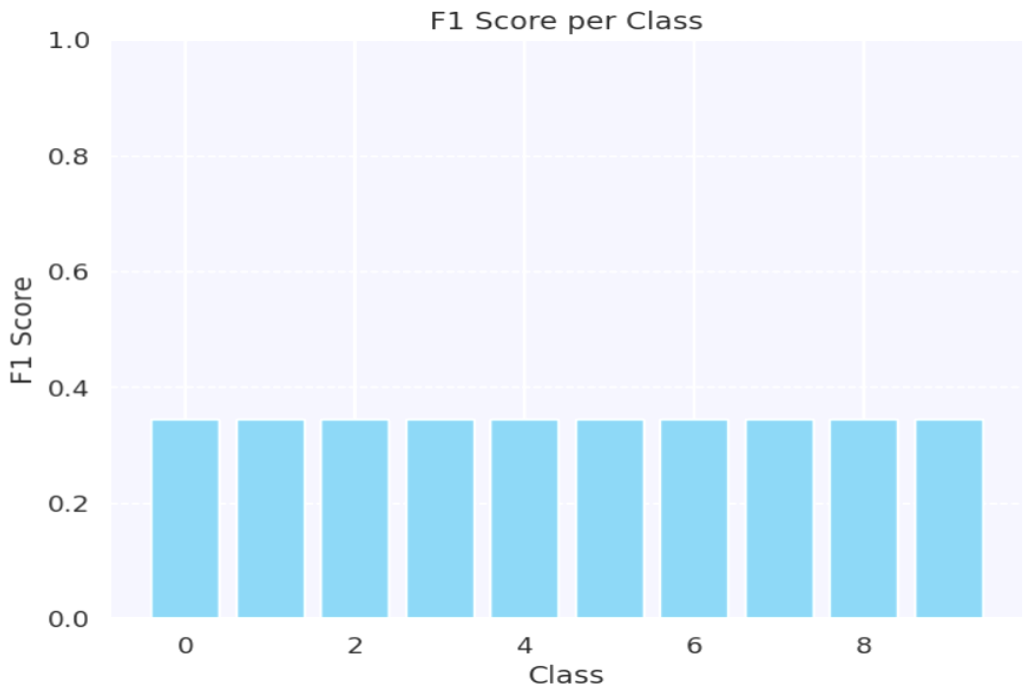
I. AUC ROC Curve:



II. Confusion Matrix:

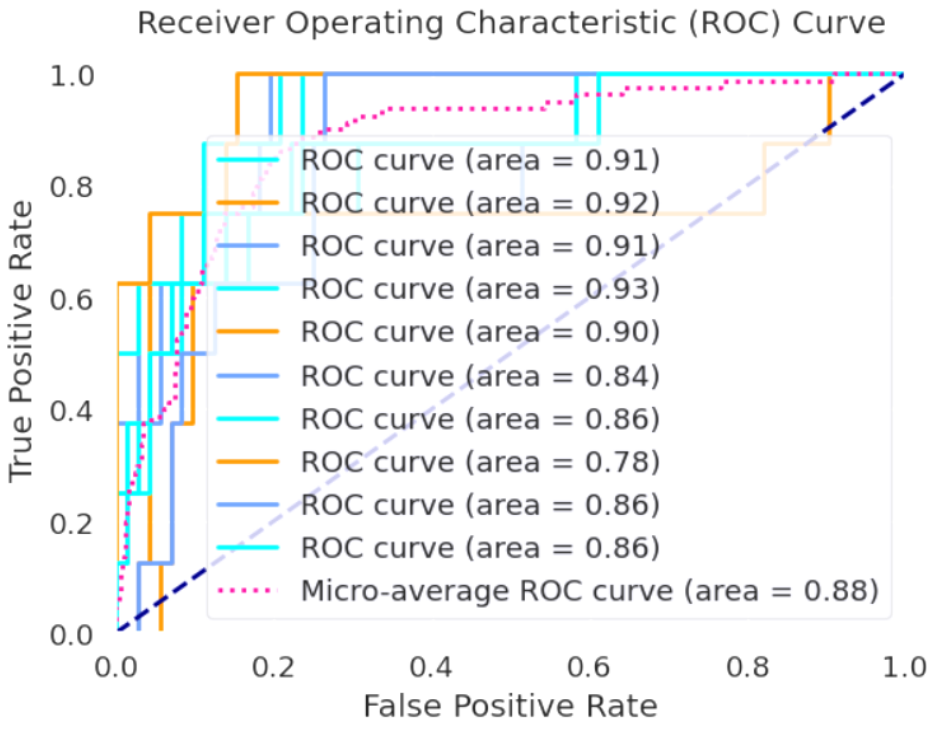


III. F1 Score:

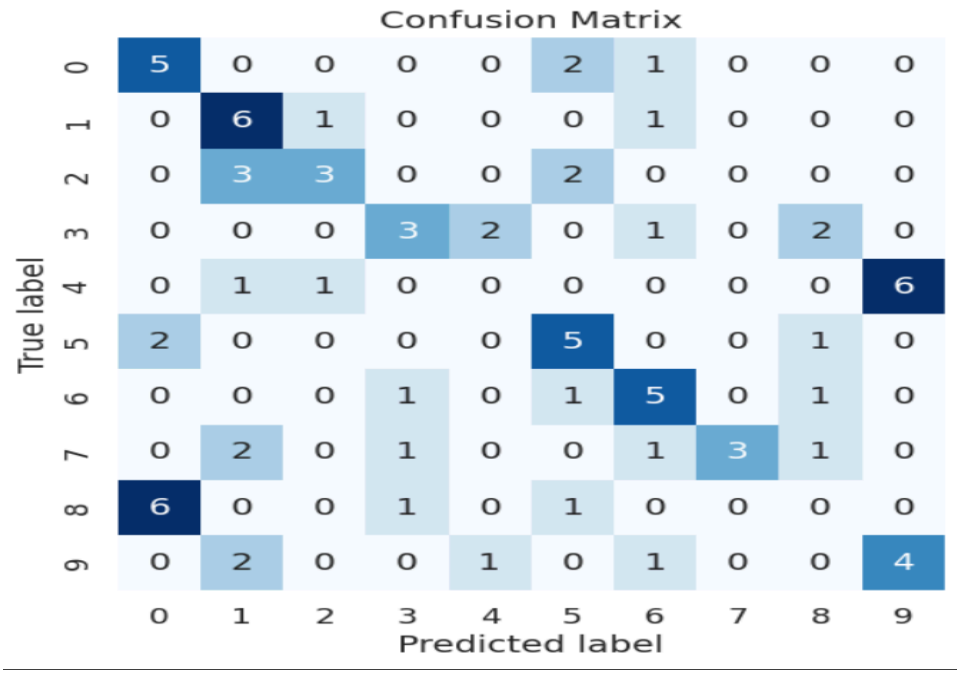


b. Transformer with 1 head and 2 layers:

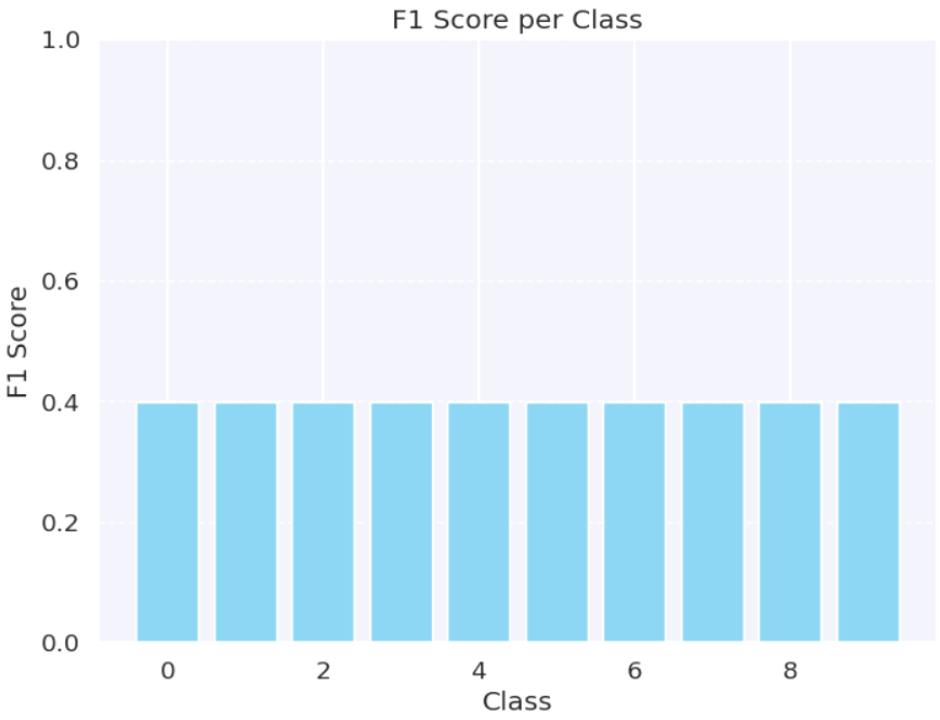
I. AUC ROC Curve:



II. Confusion Matrix:

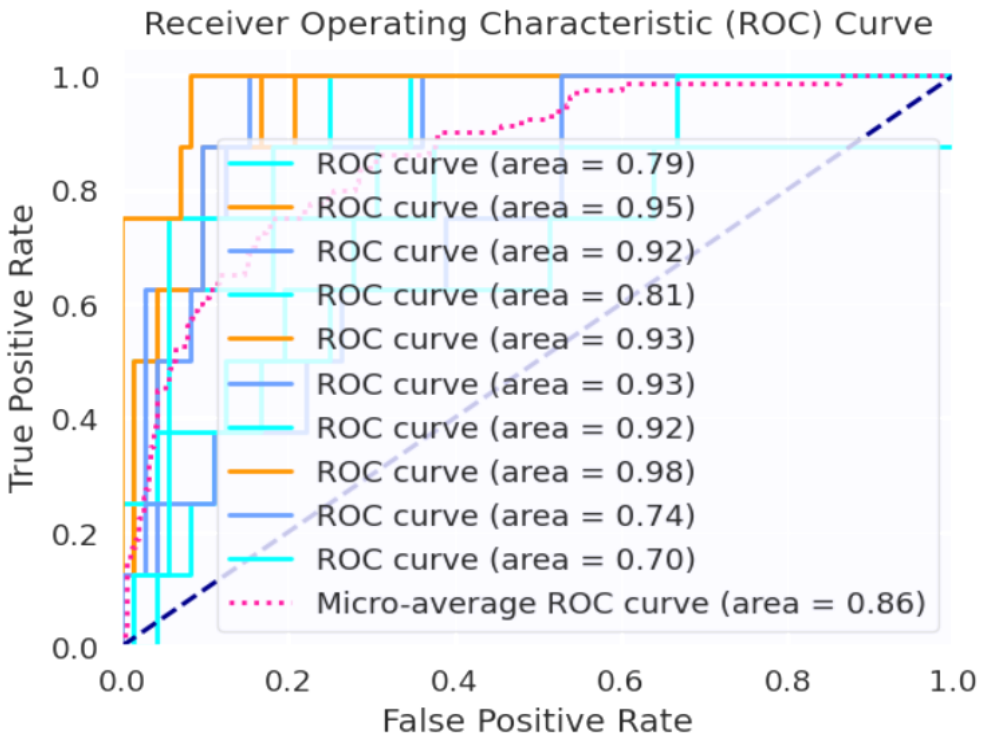


III. F1 Score:

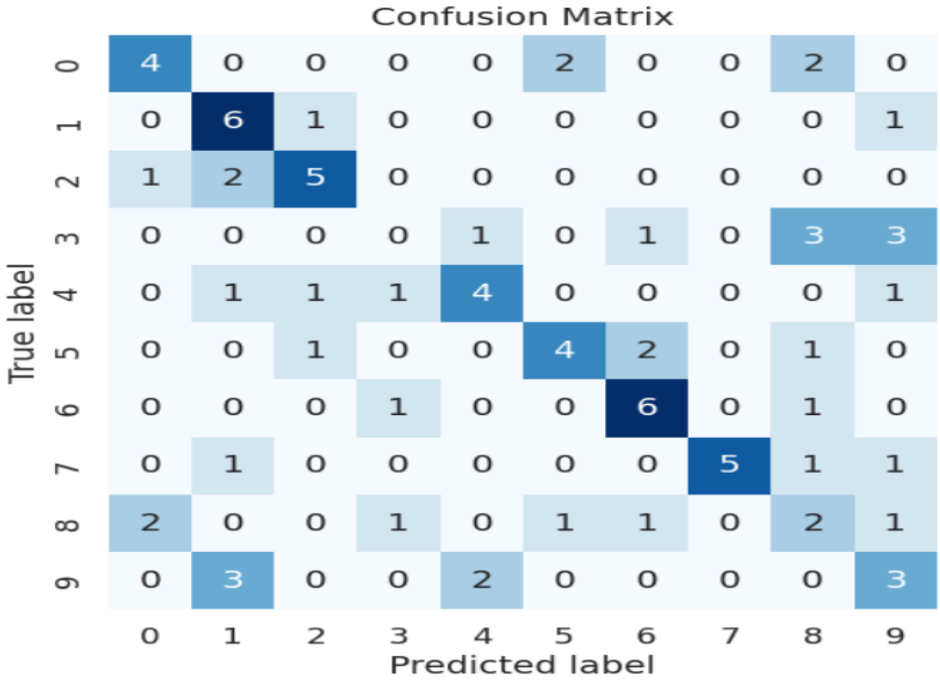


c. Transformer with 2 head and 2 layers:

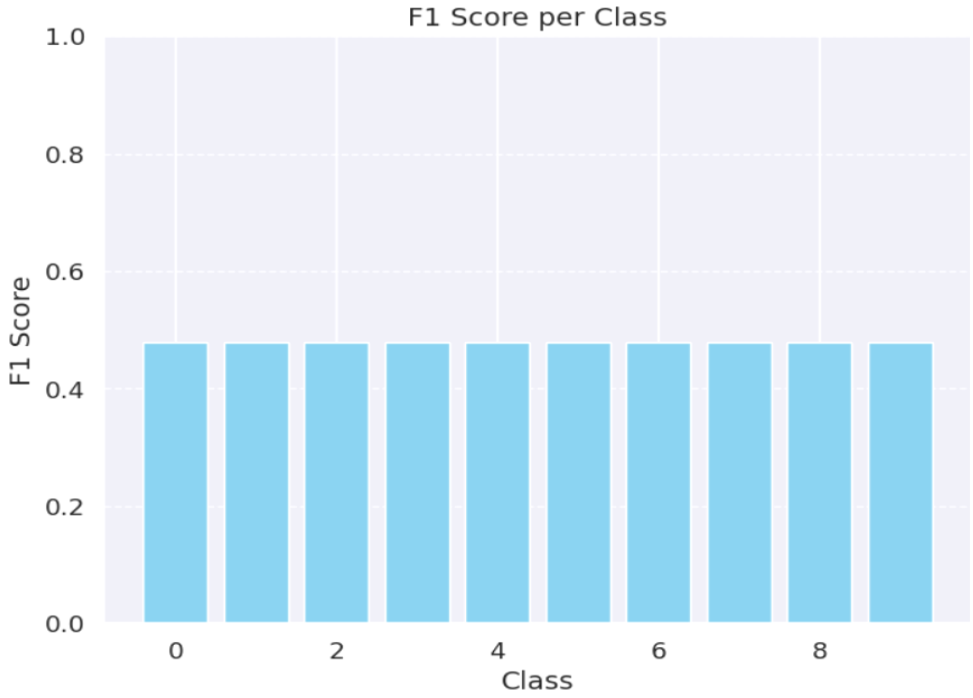
I. AUC ROC Curve:



II. Confusion Matrix

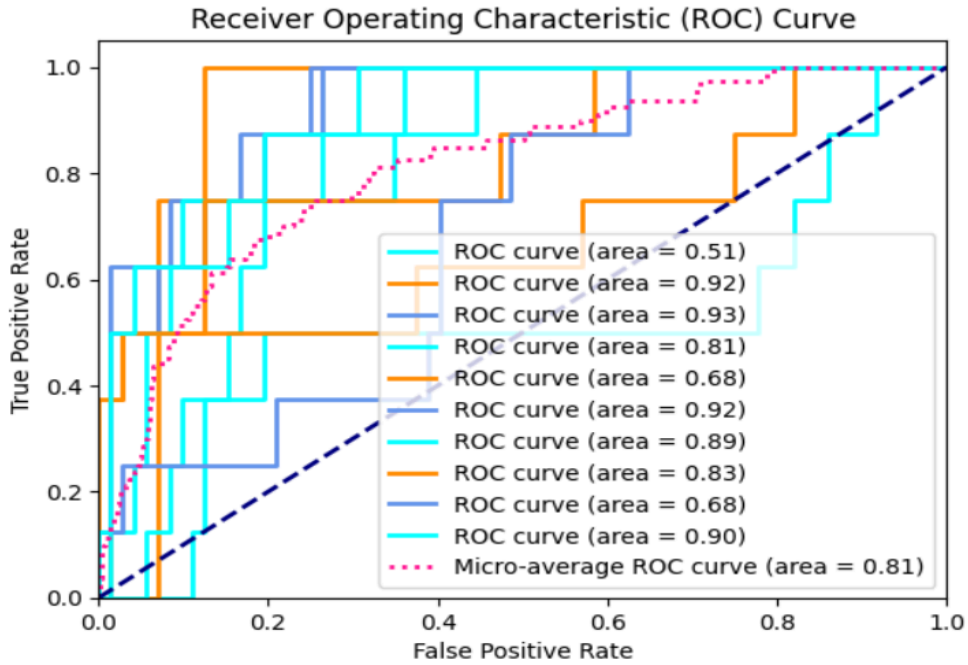


III. F1 Score

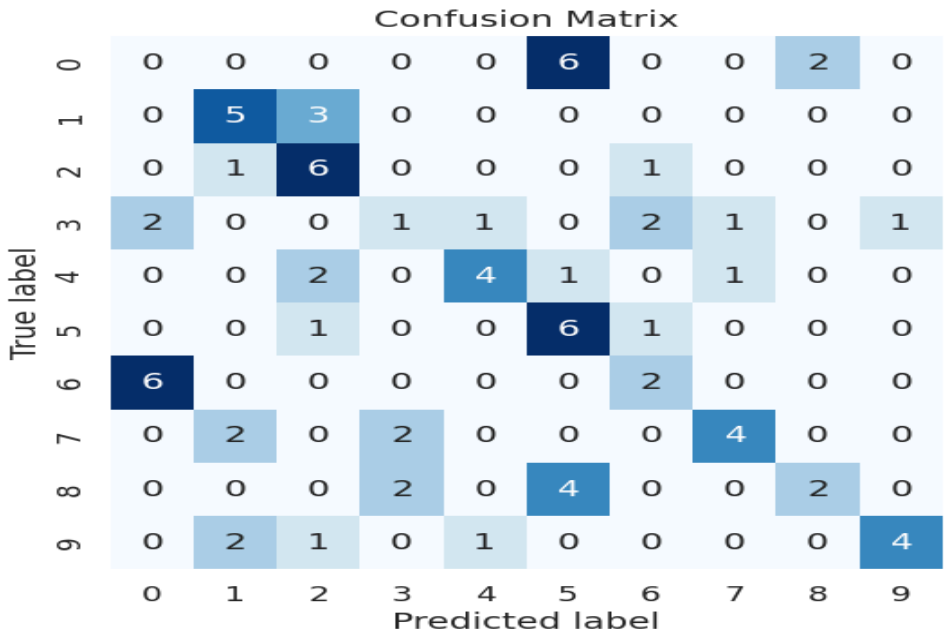


d. Transformer with 4 head and 2 layers:

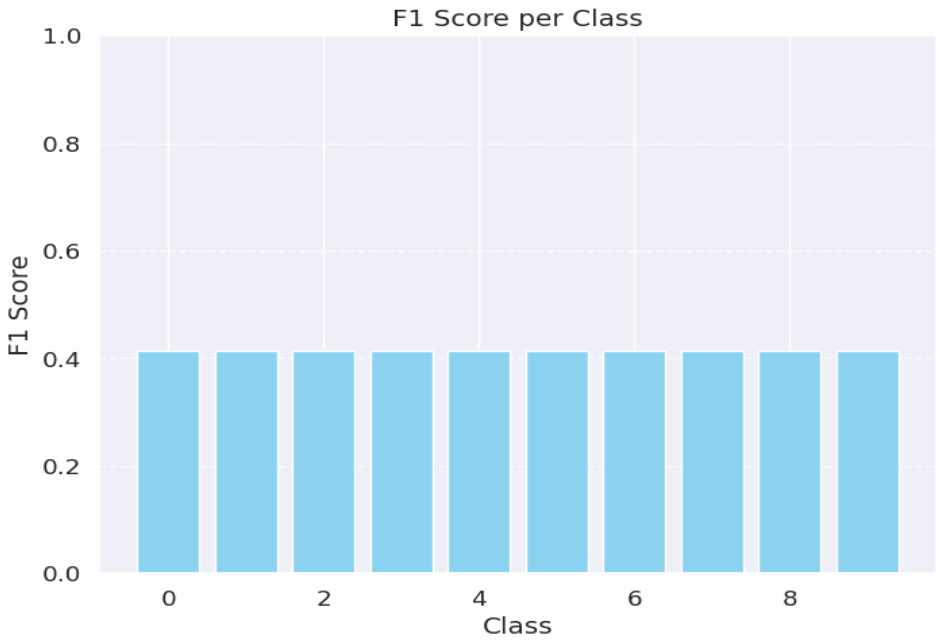
I. AUC ROC Curve



II. Confusion Matrix



III. F1 Score

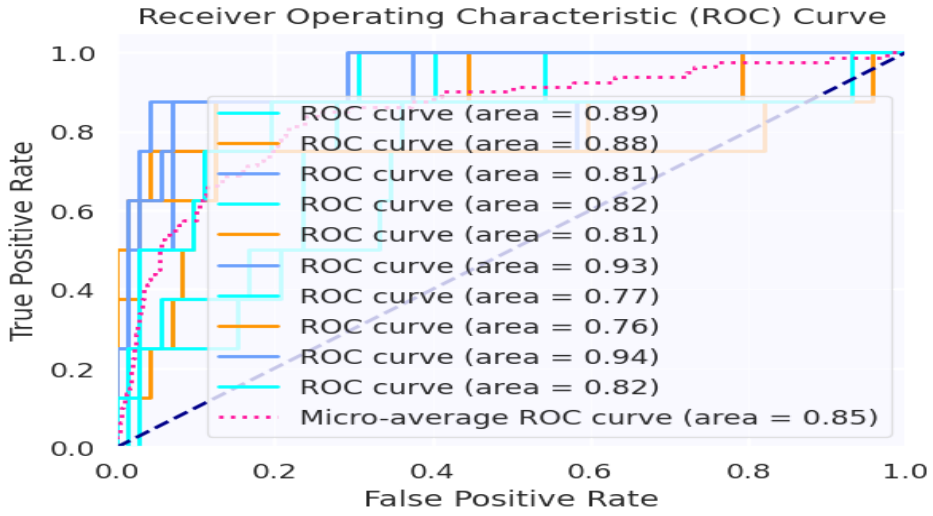


2. Using KFold validation: K fold cross-validation is a way to check how well a model can predict things. We split our data into "k" groups, and each time, one group is used to validate the model while the rest are used to train it. This is repeated "k" times, and then the results are compared to see how good the model is overall. In our case, K is chosen as 4.
A loop has been run from 2 to 5 resulting in 4 different fold values of the validation set. The fold 1 is already fixed for the testing set initially.
In this loop, we initialize our model every time and run it for 100 epochs and check the training, validation and testing metrics.

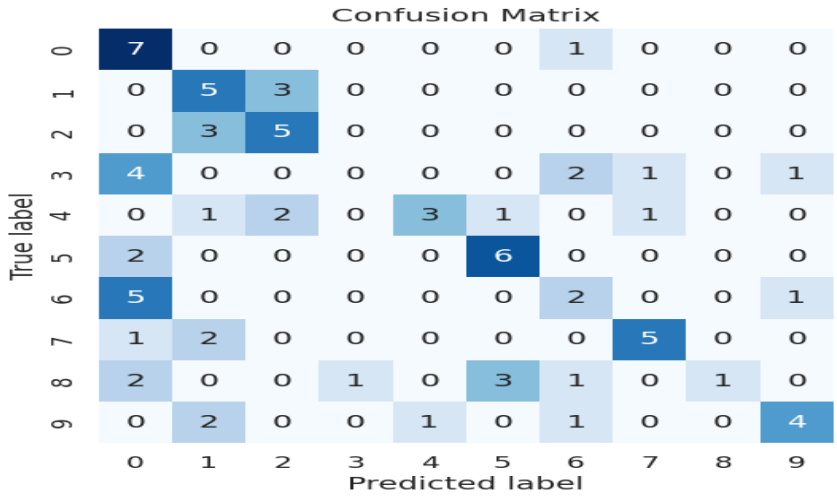
a. 1D-CNN:

When Validation Fold is 2:

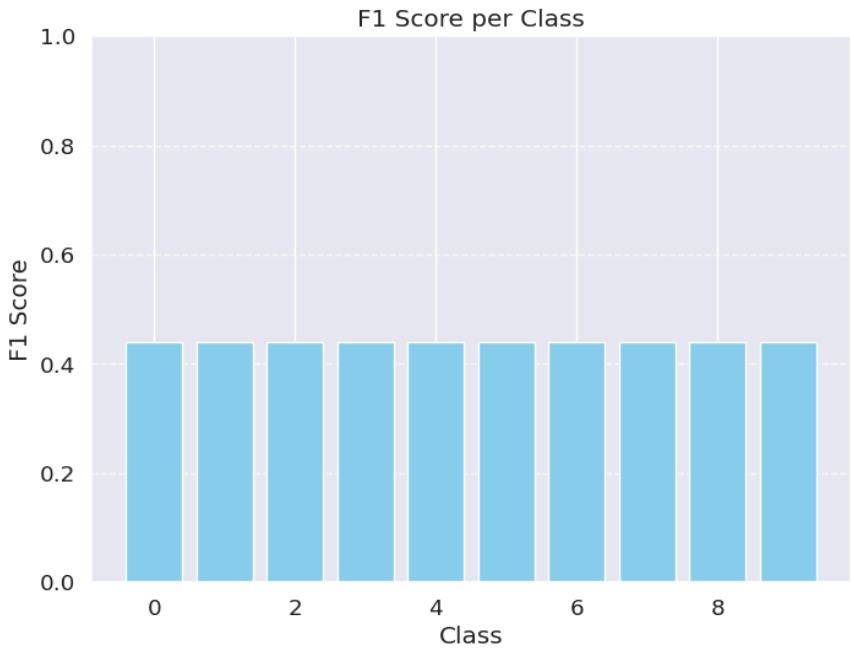
AUC ROC Curve



Confusion Matrix

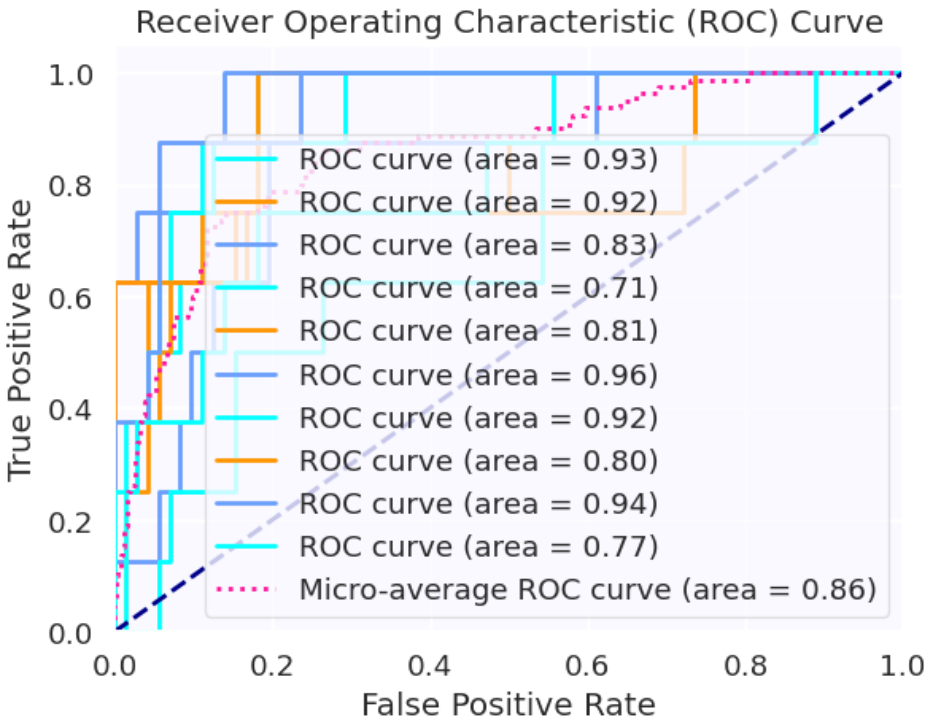


F1 Score

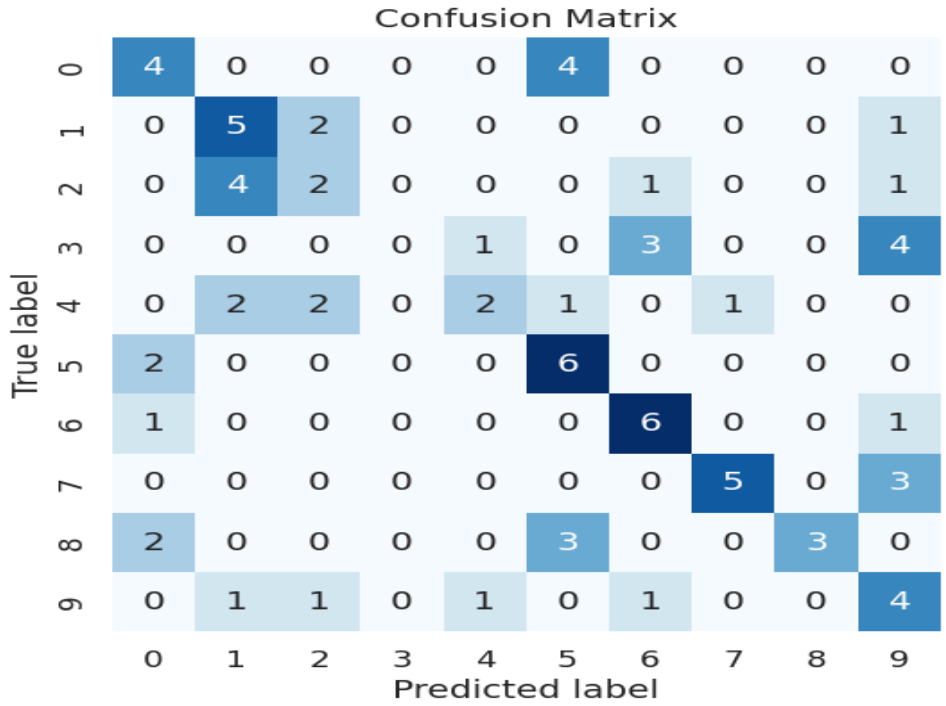


When Validation Fold is 3:

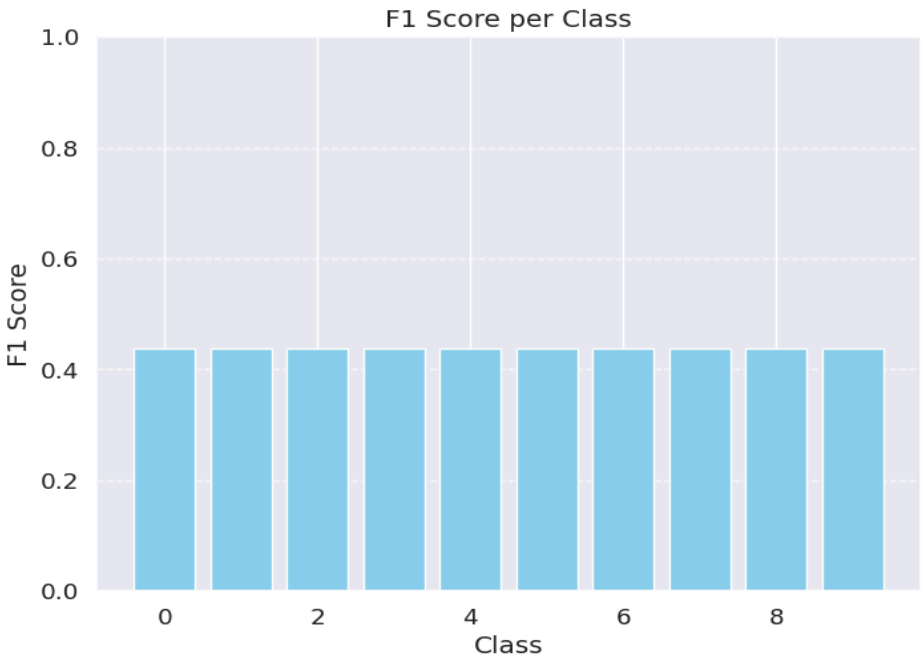
AUC ROC Curve



Confusion Matrix

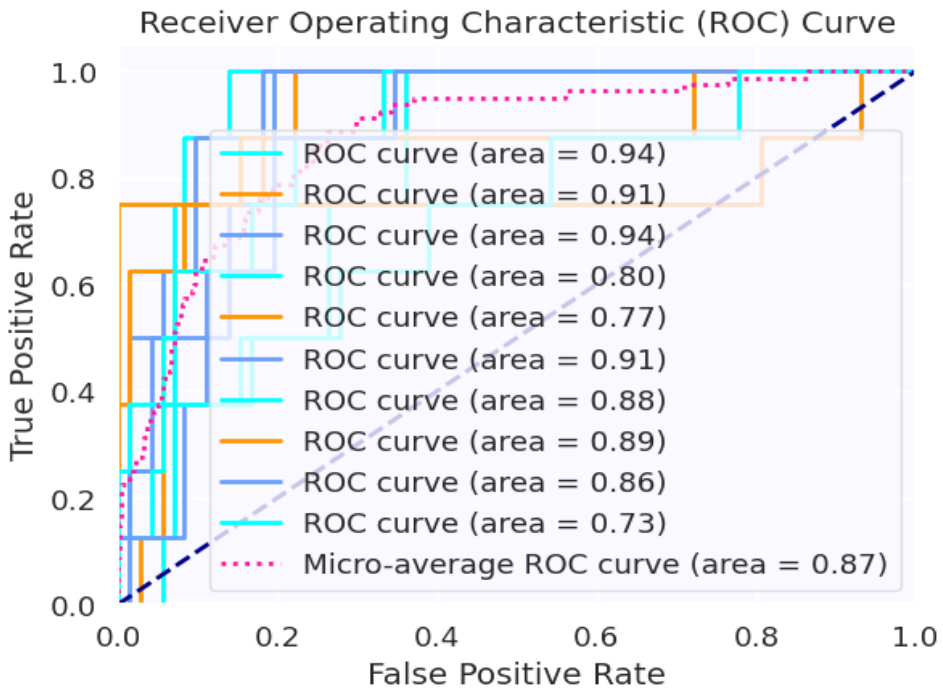


F1 Score

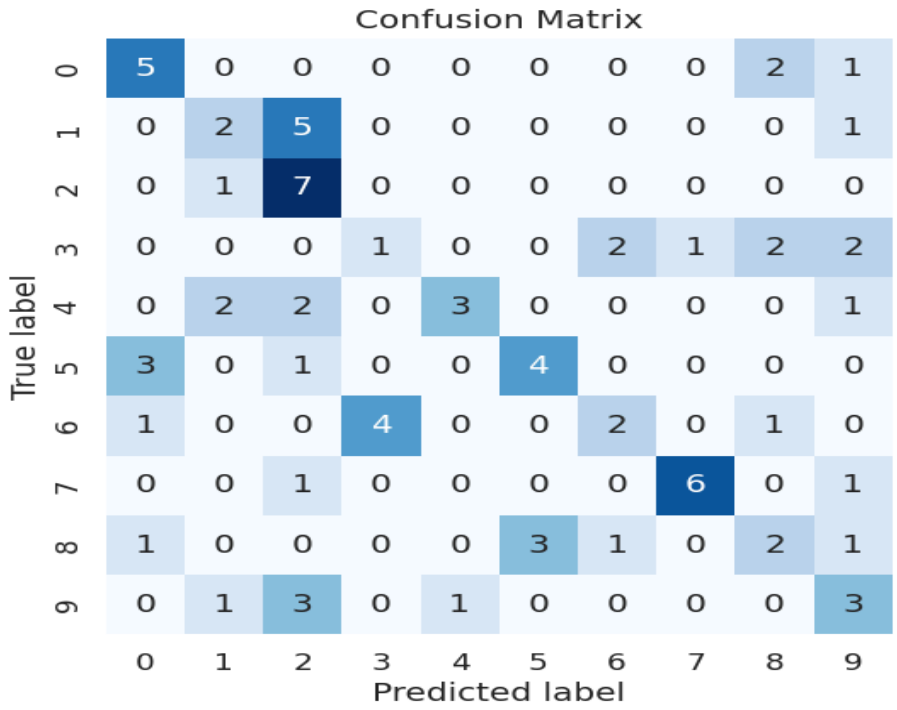


When Validation Fold is 4:

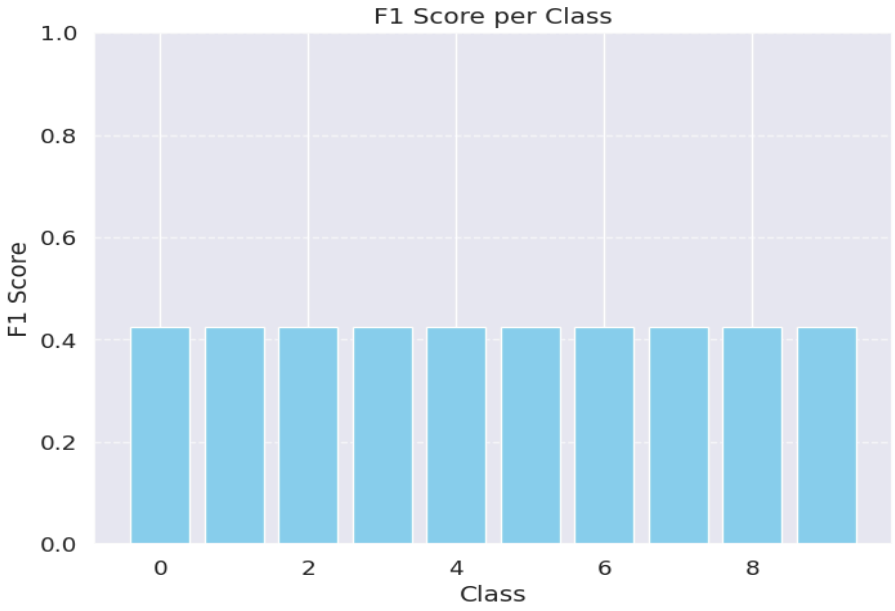
AUC ROC Curve



Confusion Matrix

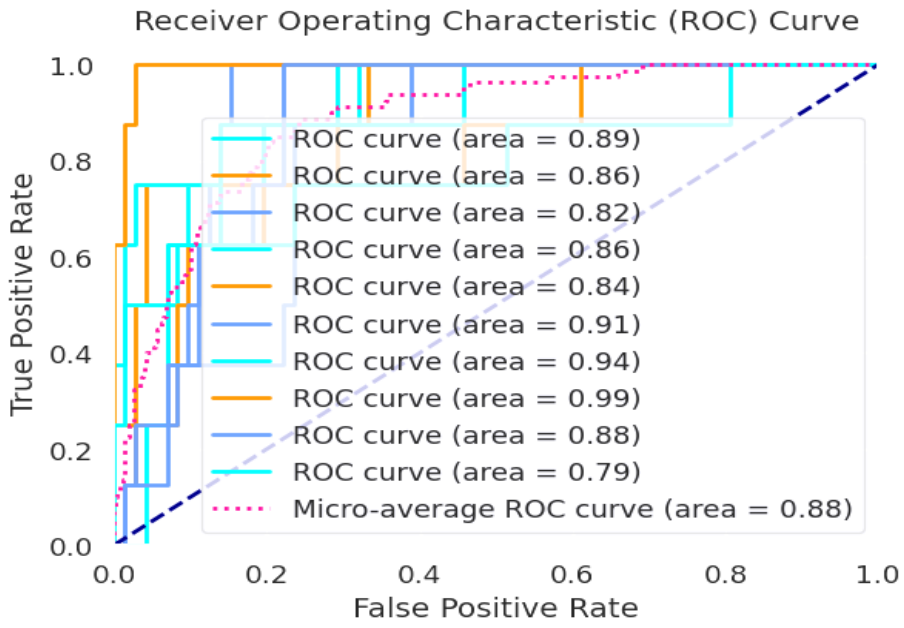


F1 Score

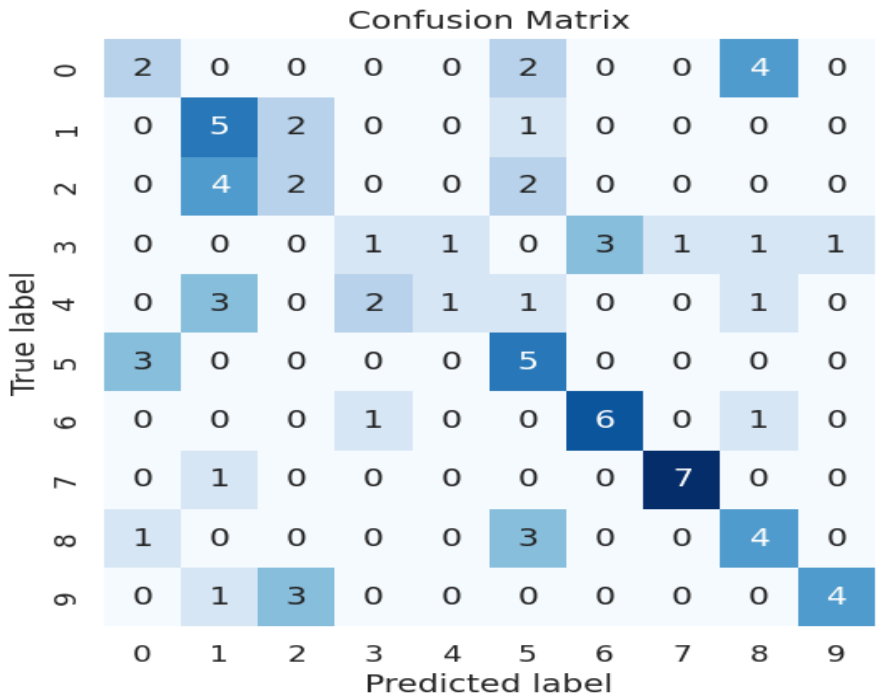


When Validation Fold is 5:

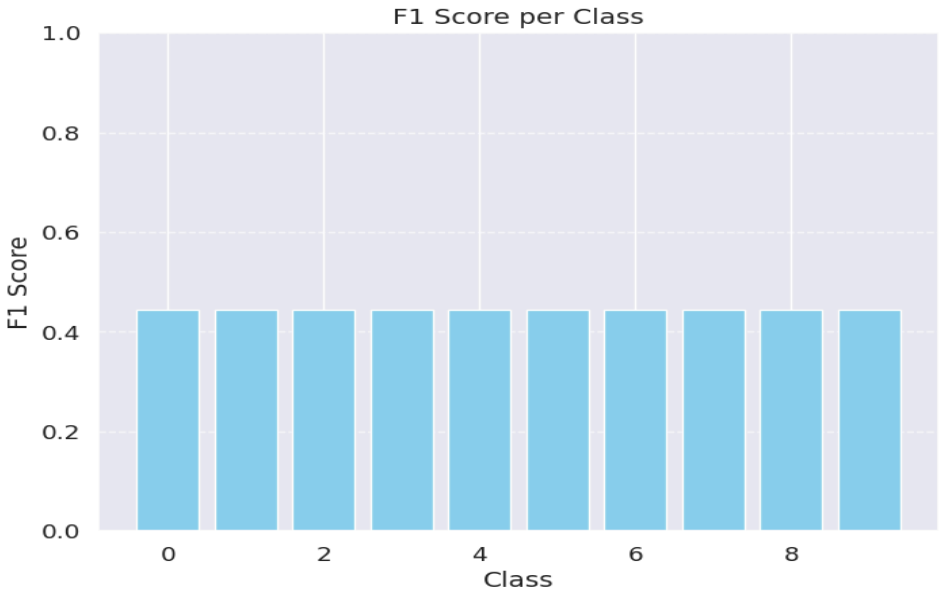
AUC ROC Curve



Confusion Matrix



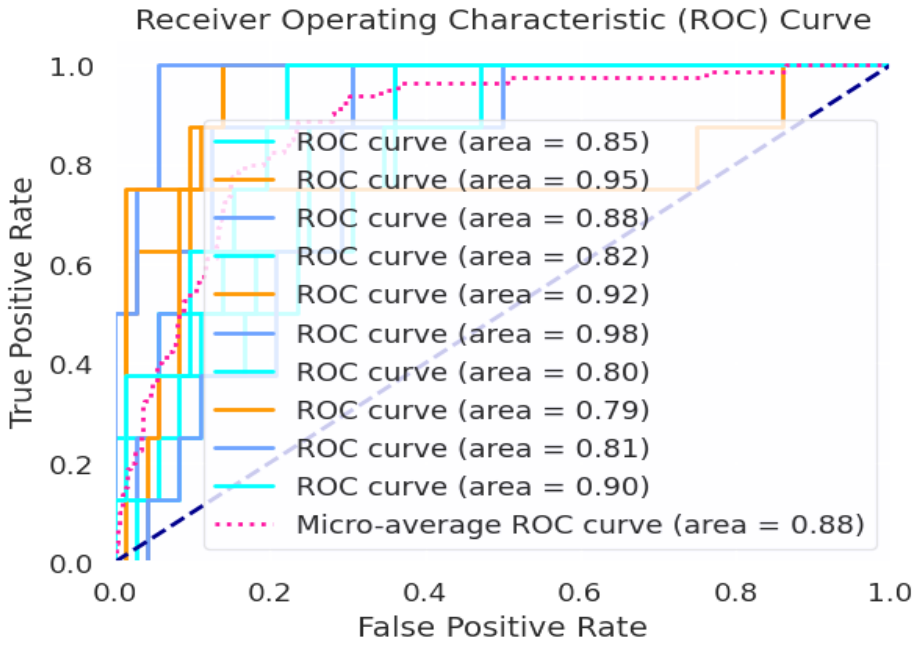
F1 Score



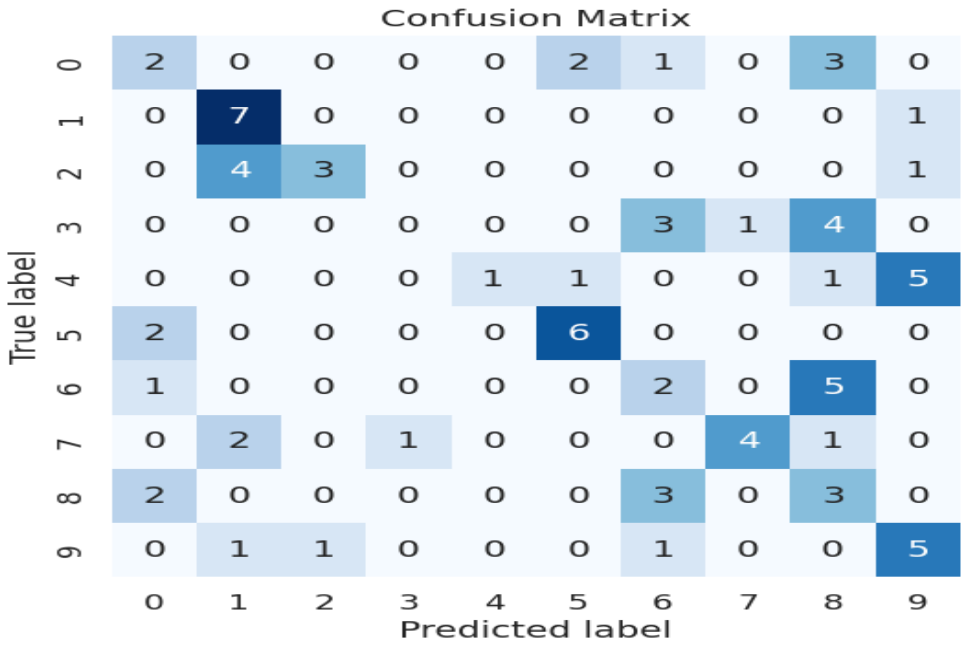
b. Transformer with 1 head and 2 layers:

When Validation Fold is 2:

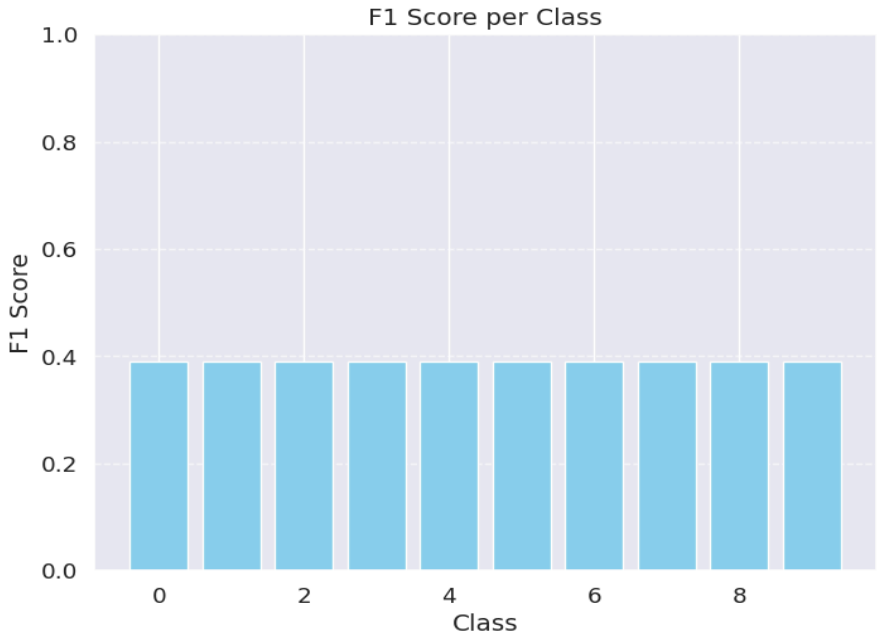
AUC ROC Curve



Confusion Matrix

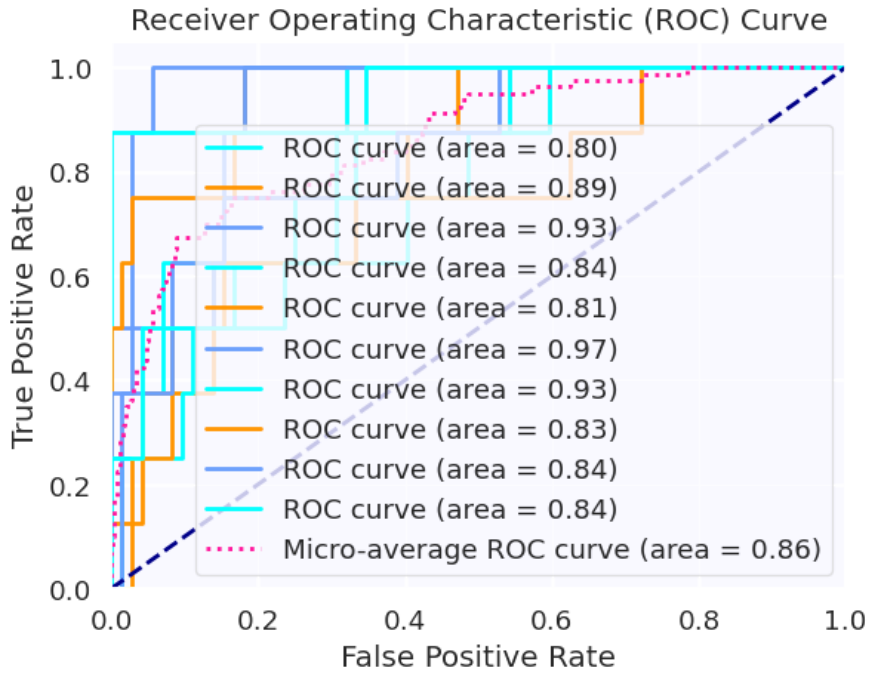


F1 Score

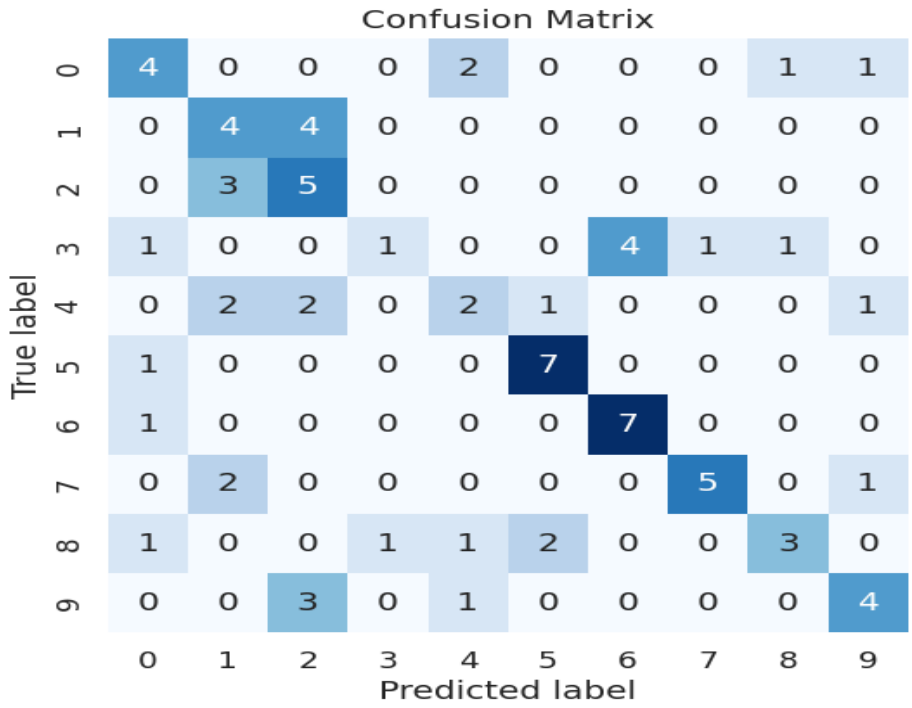


When Validation Fold is 3:

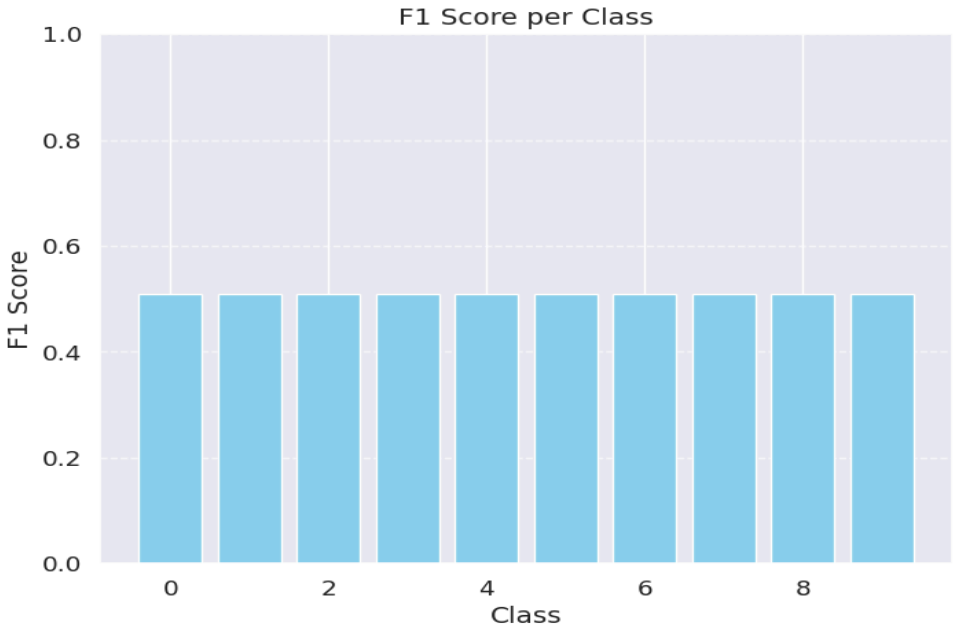
AUC ROC Curve



Confusion Matrix

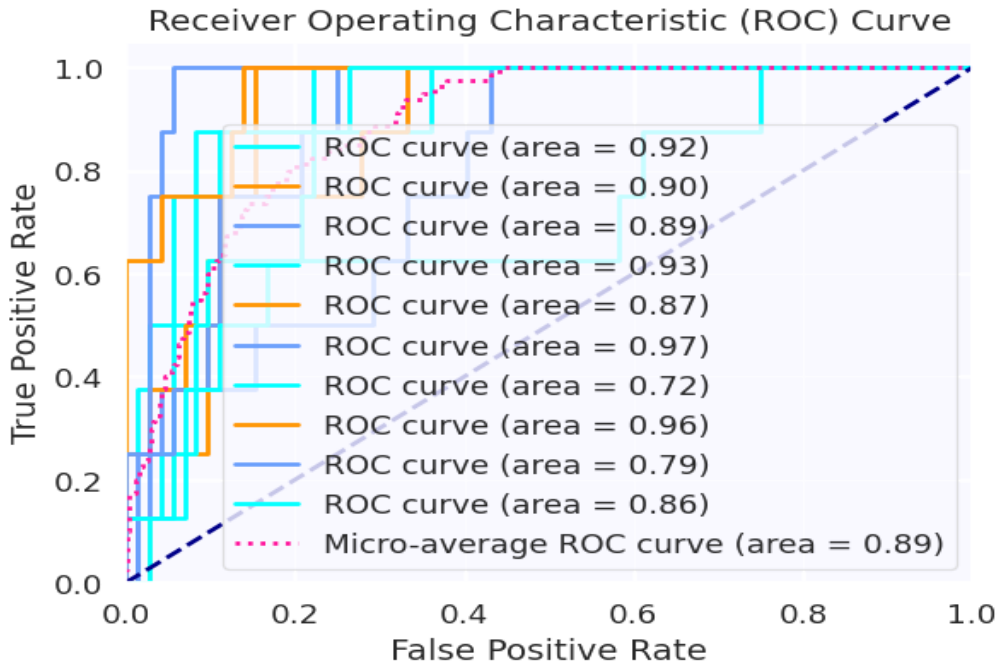


F1 Score

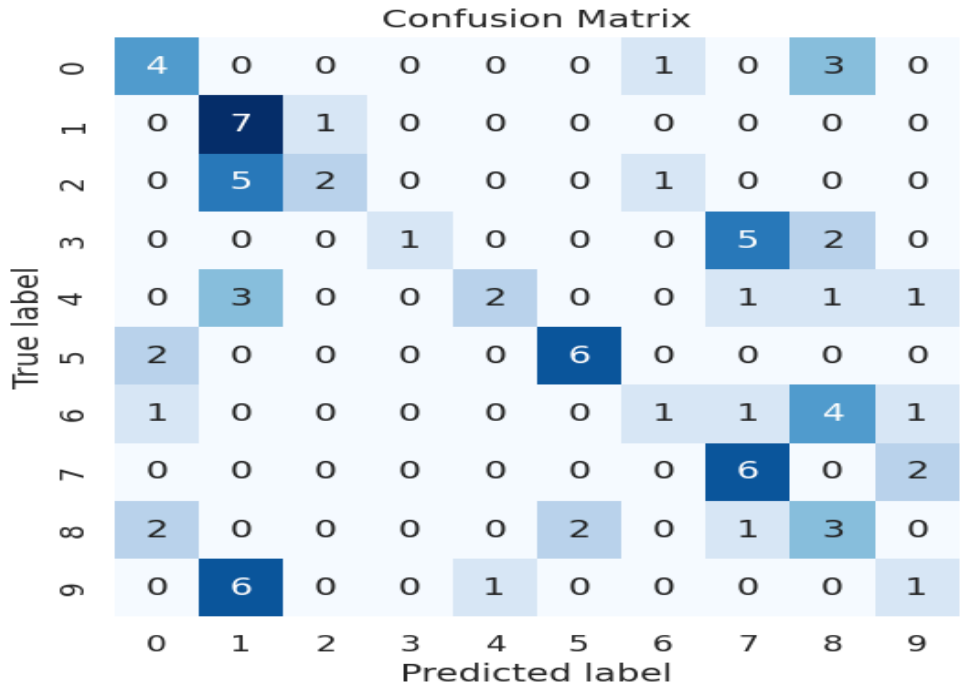


When Validation Fold is 4:

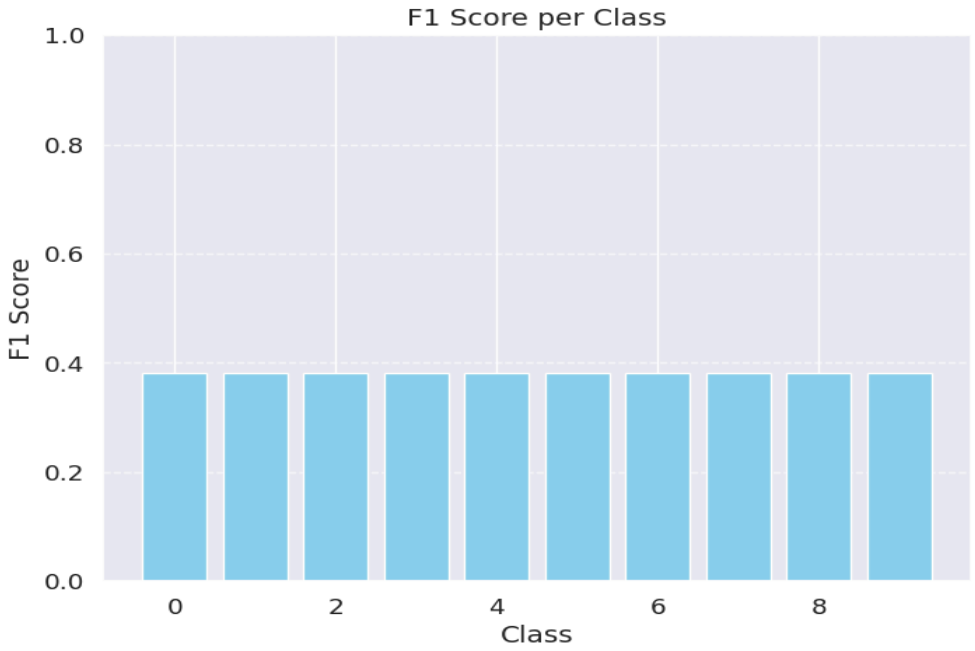
AUC ROC Curve



Confusion Matrix

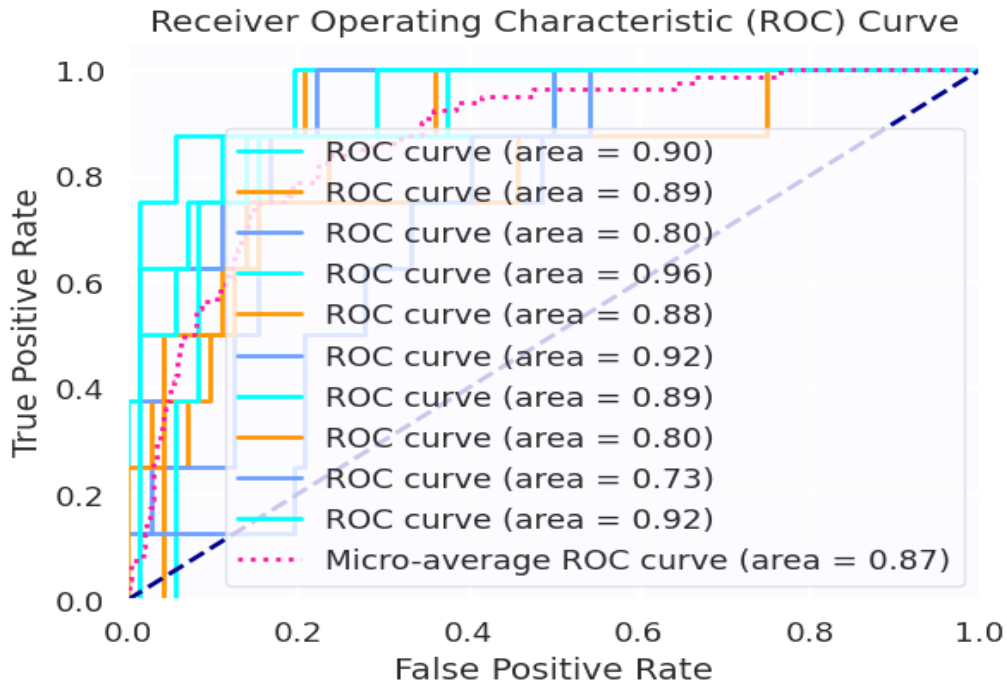


F1 Score

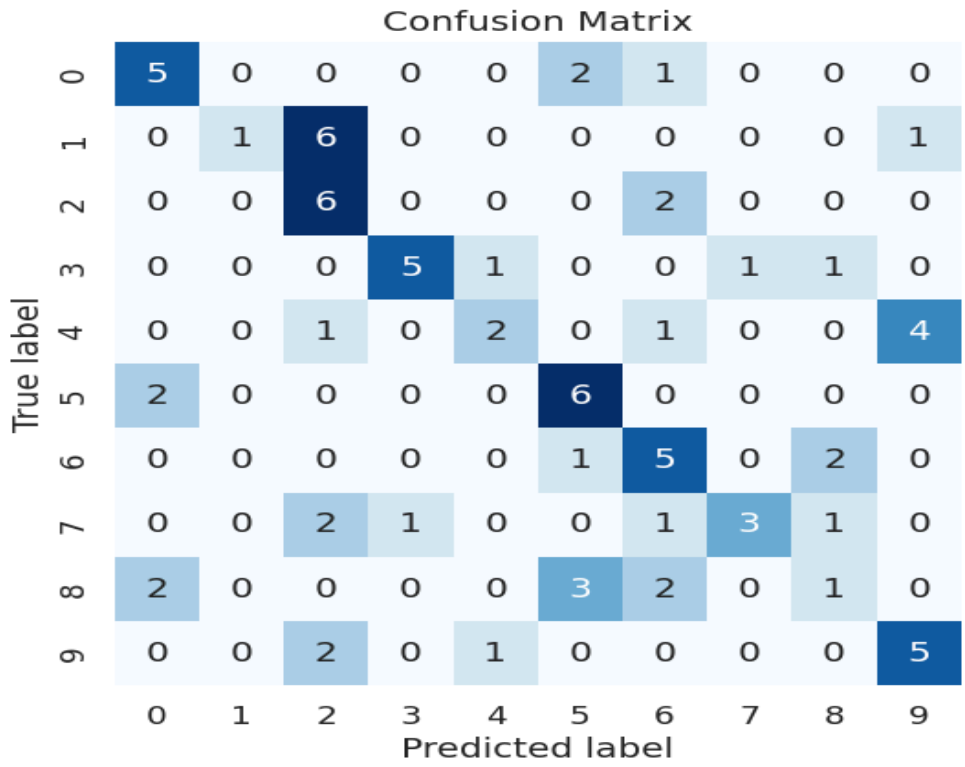


When Validation Fold is 5:

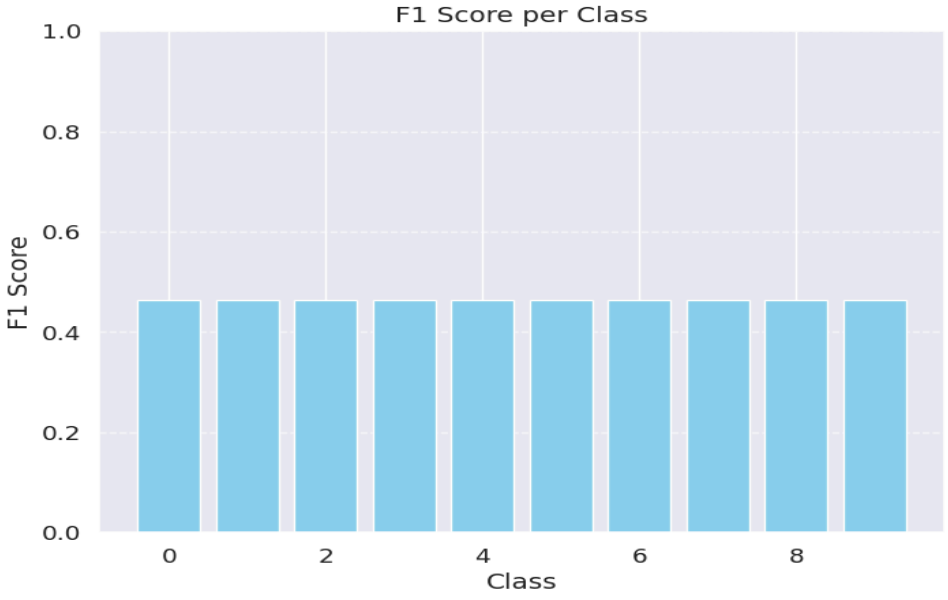
AUC ROC Curve



Confusion Matrix



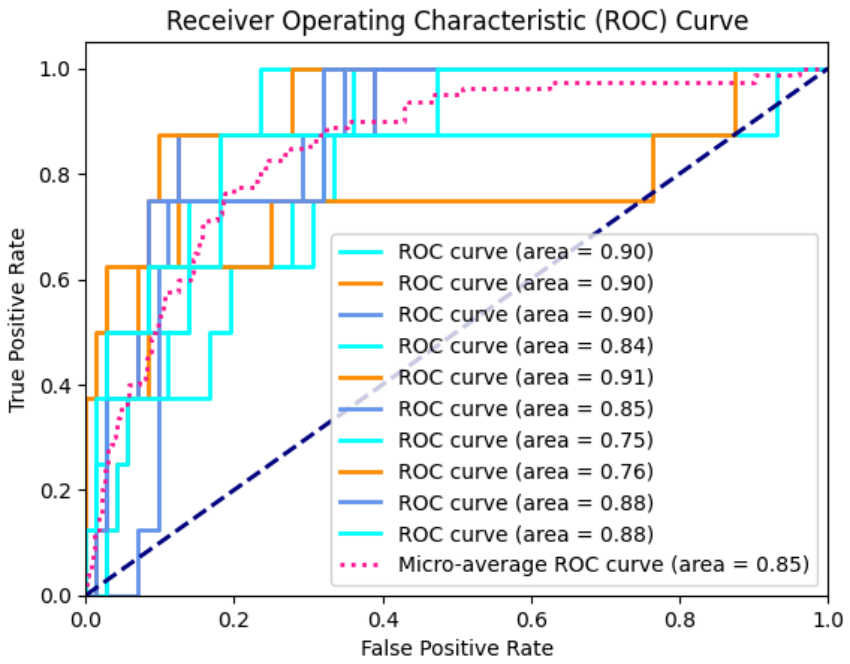
F1 Score



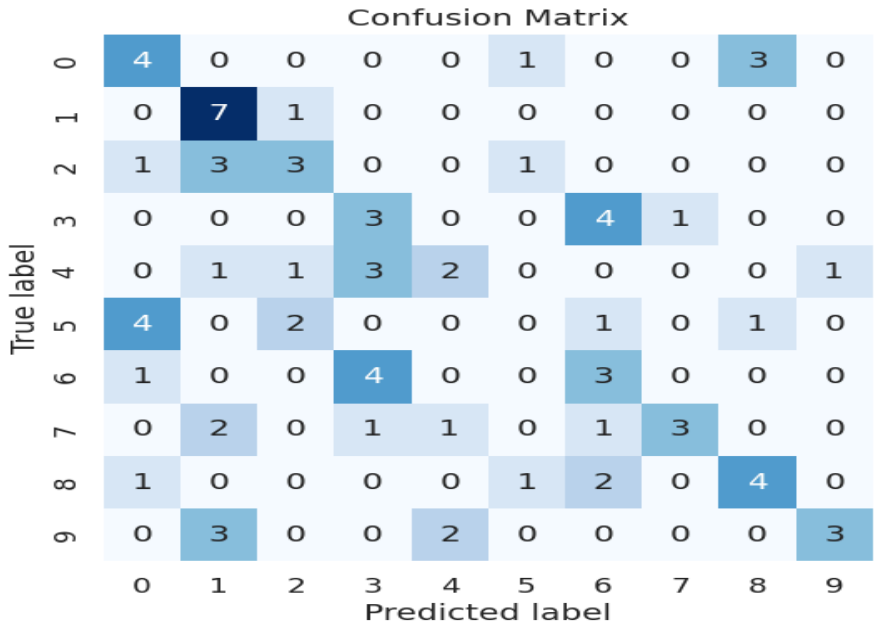
c. Transformer with 2 head and 2 layers:

When Validation Fold is 2:

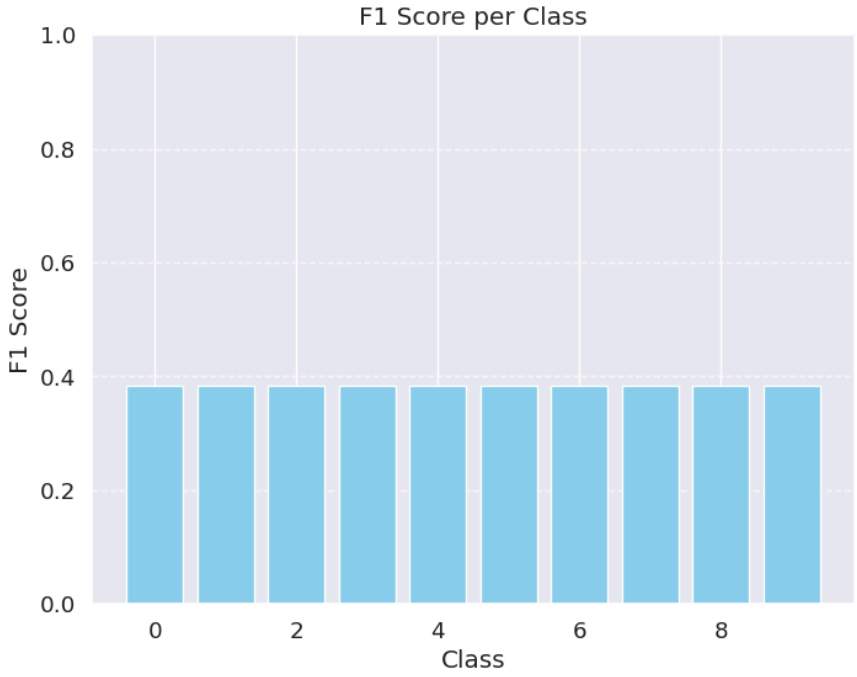
AUC ROC Curve



Confusion Matrix

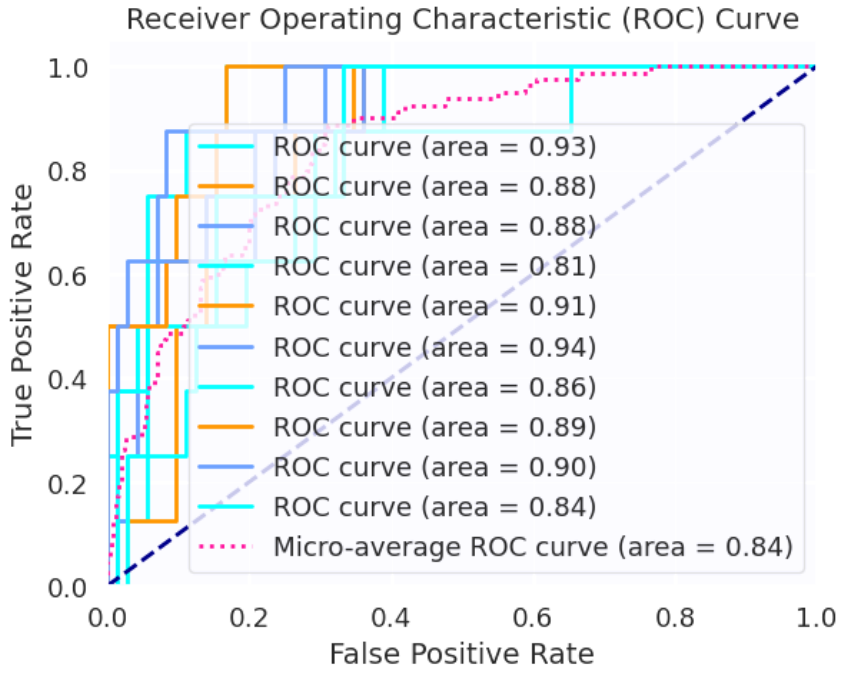


F1 Score

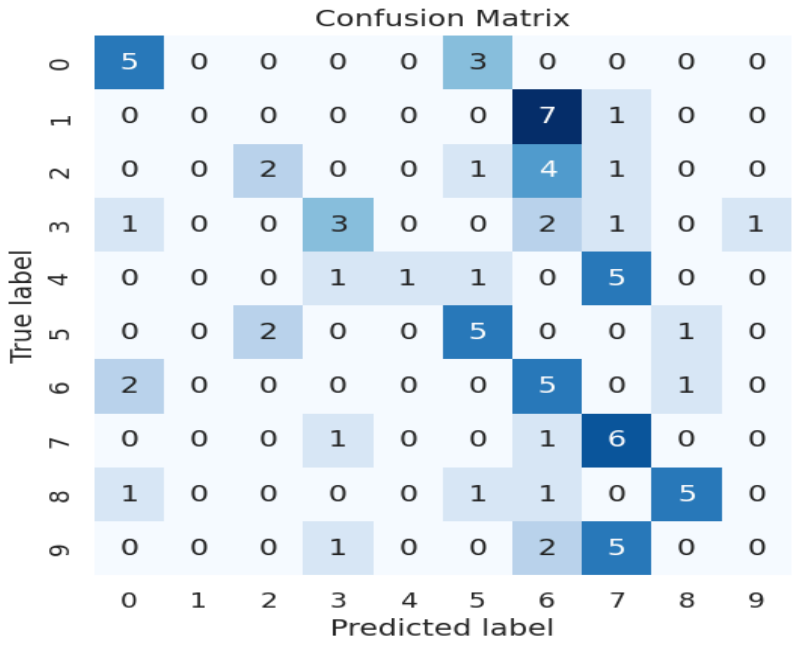


When Validation Fold is 3:

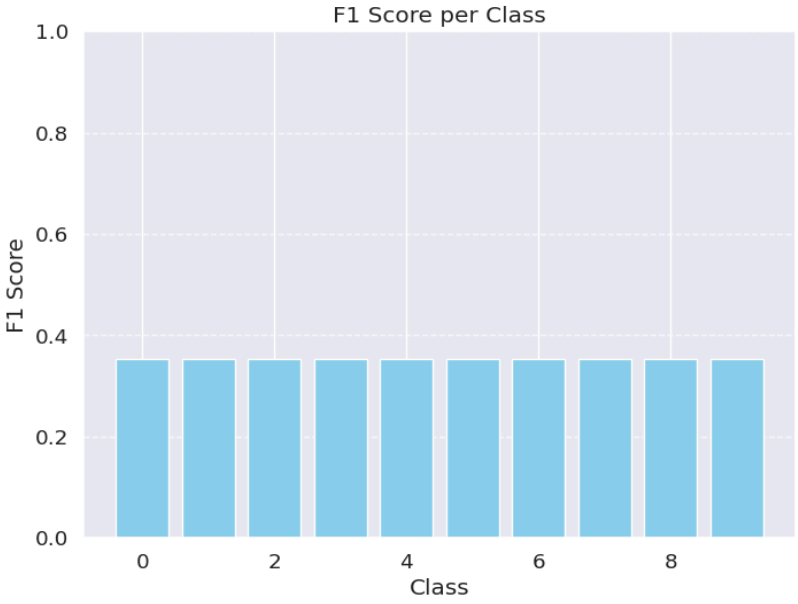
AUC ROC Curve



Confusion Matrix

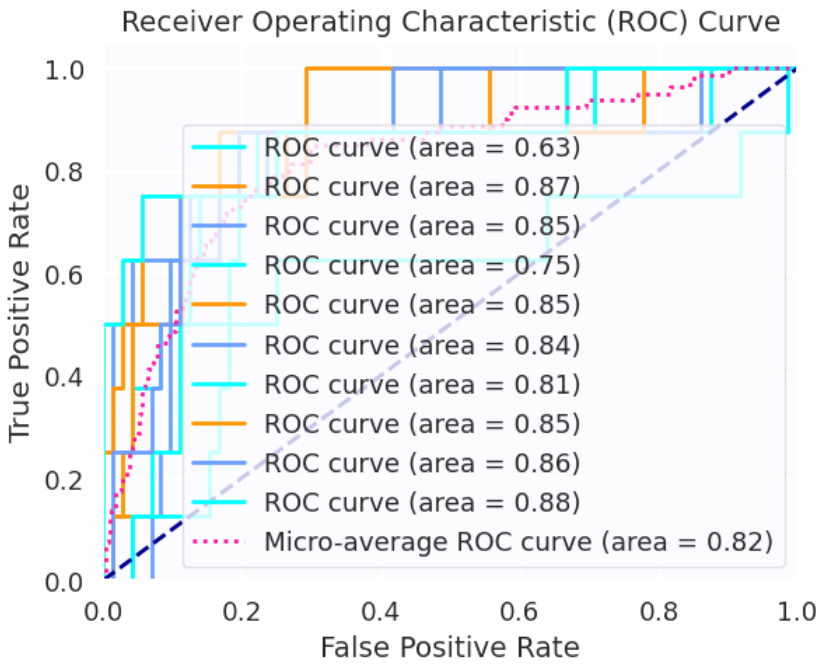


F1 Score

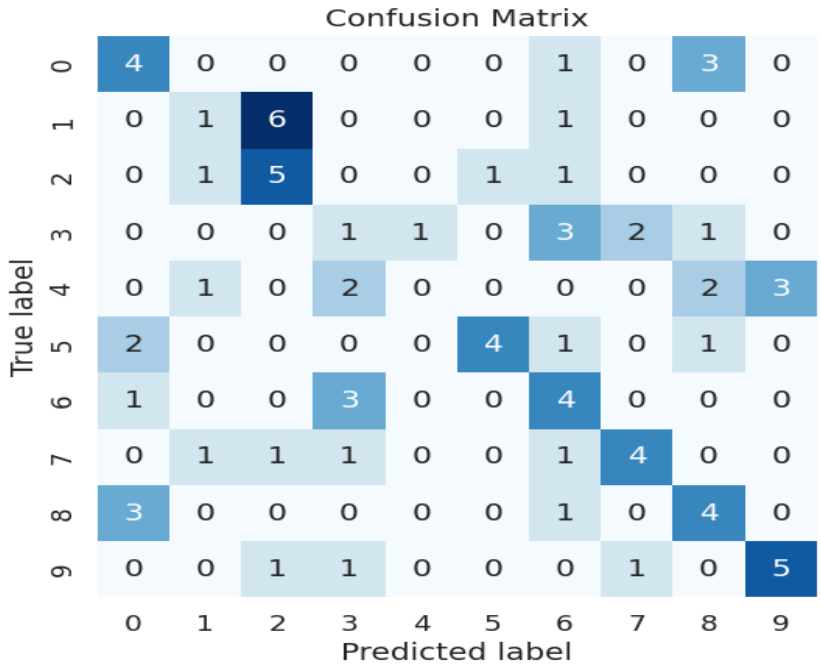


When Validation Fold is 4:

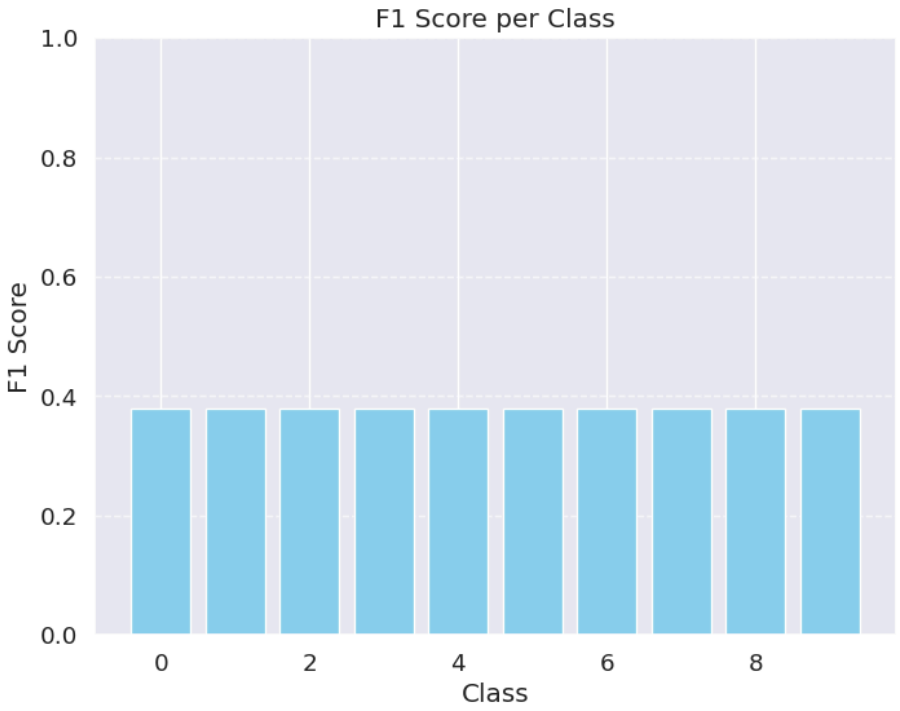
AUC ROC Curve



Confusion Matrix

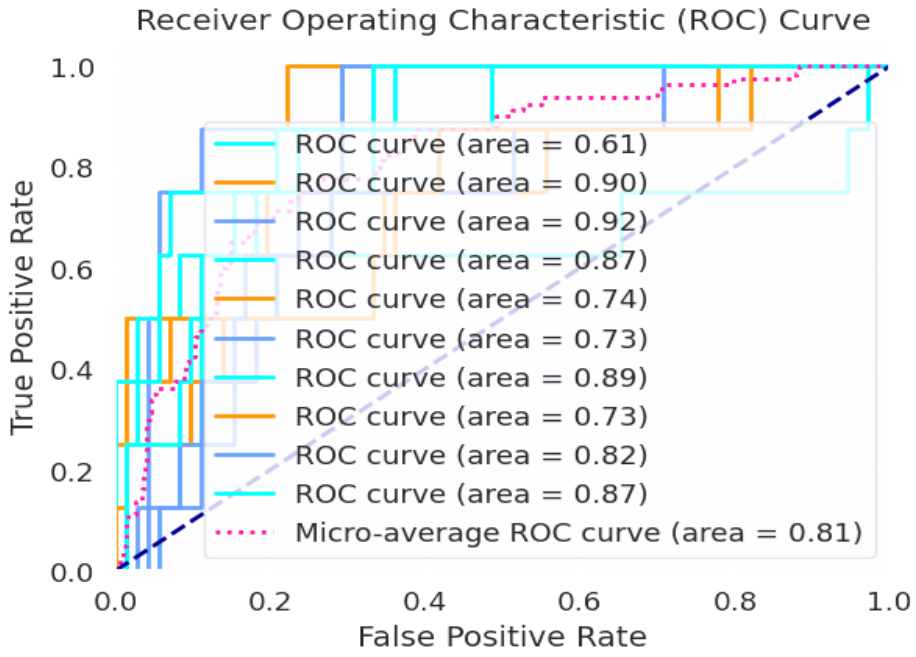


F1 Score

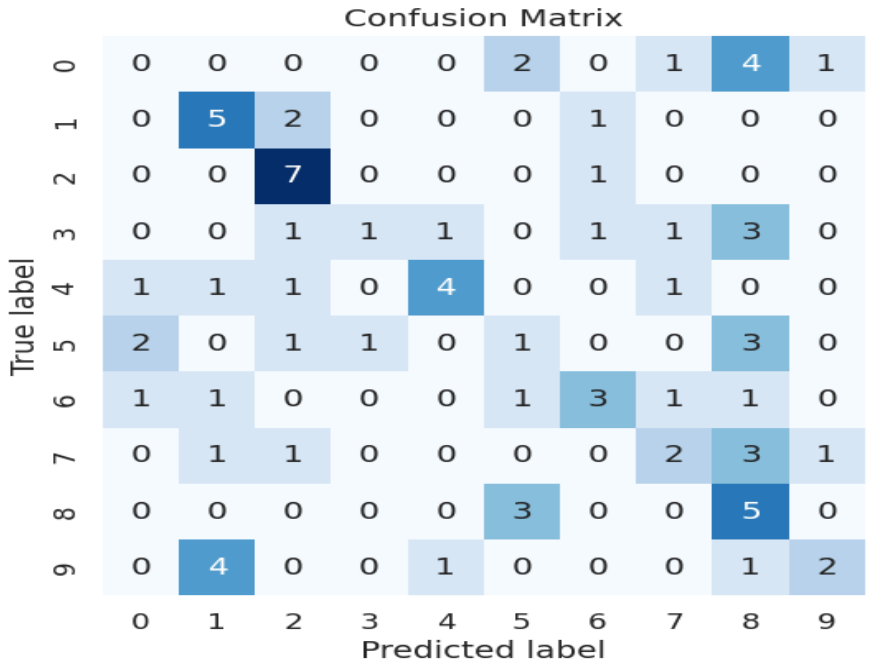


When Validation Fold is 5:

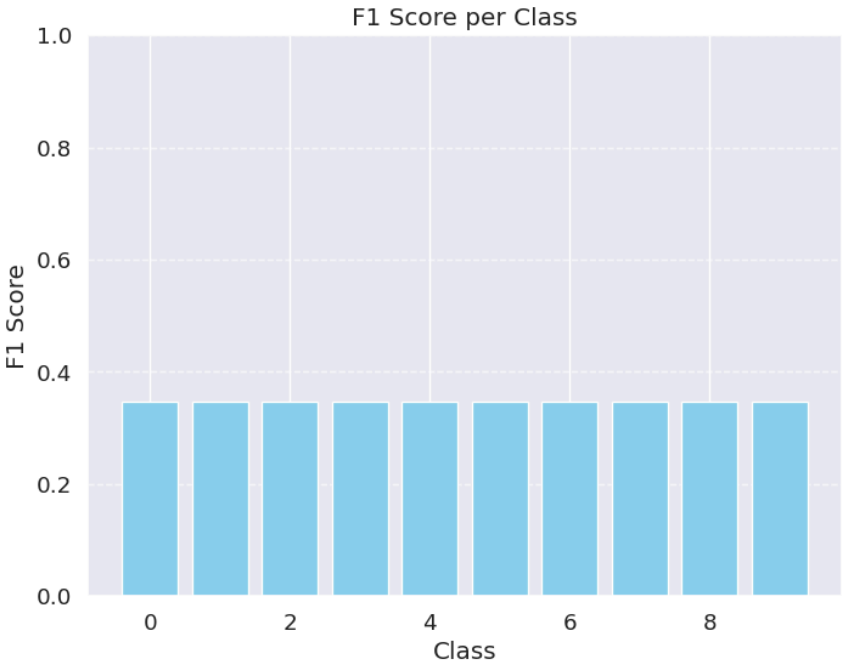
AUC ROC Curve



Confusion Matrix



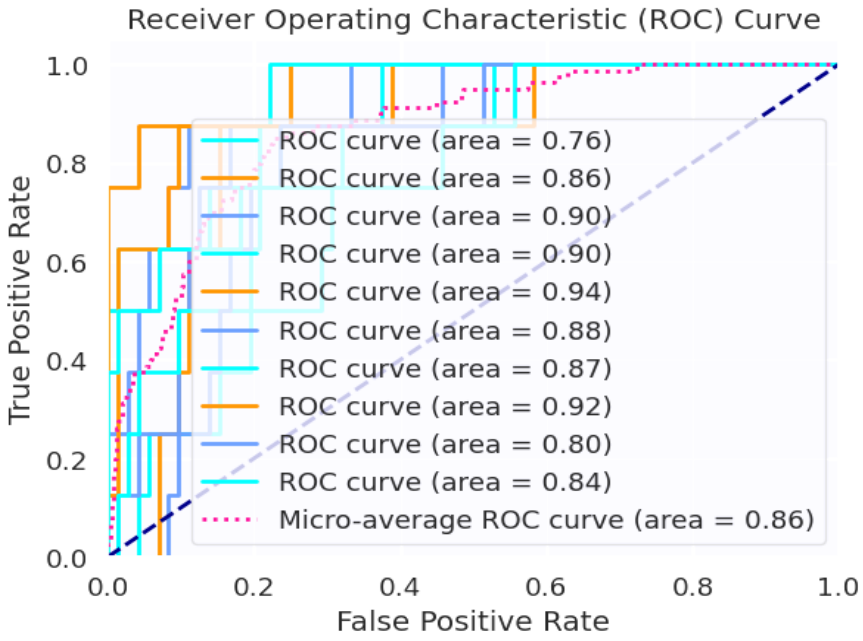
F1 Score



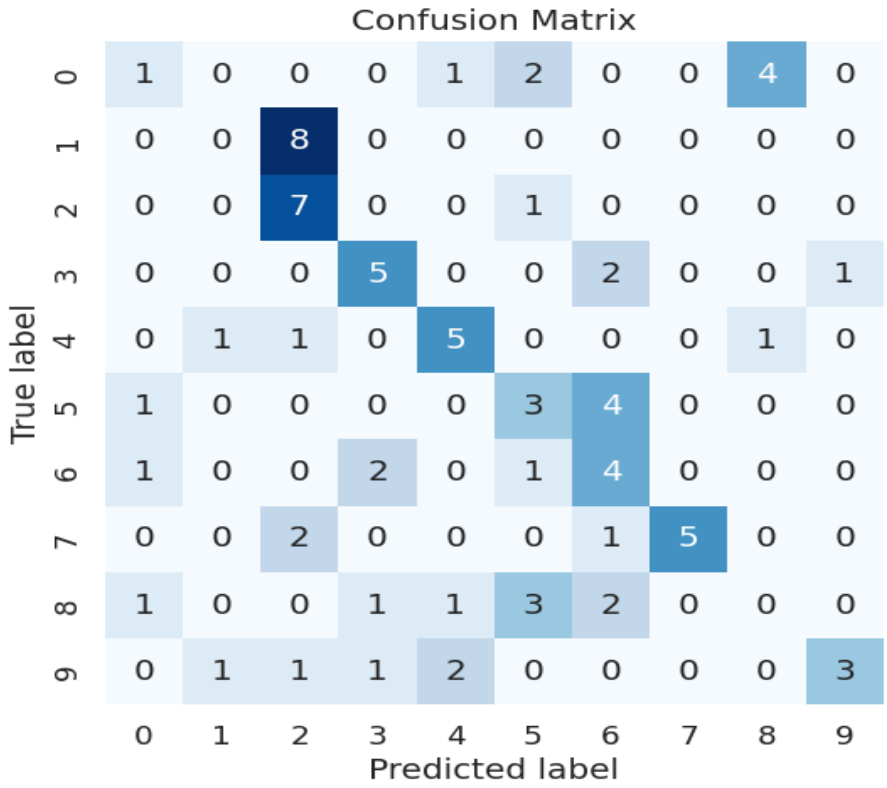
d. Transformer with 4 head and 2 layers:

When Validation Fold is 2:

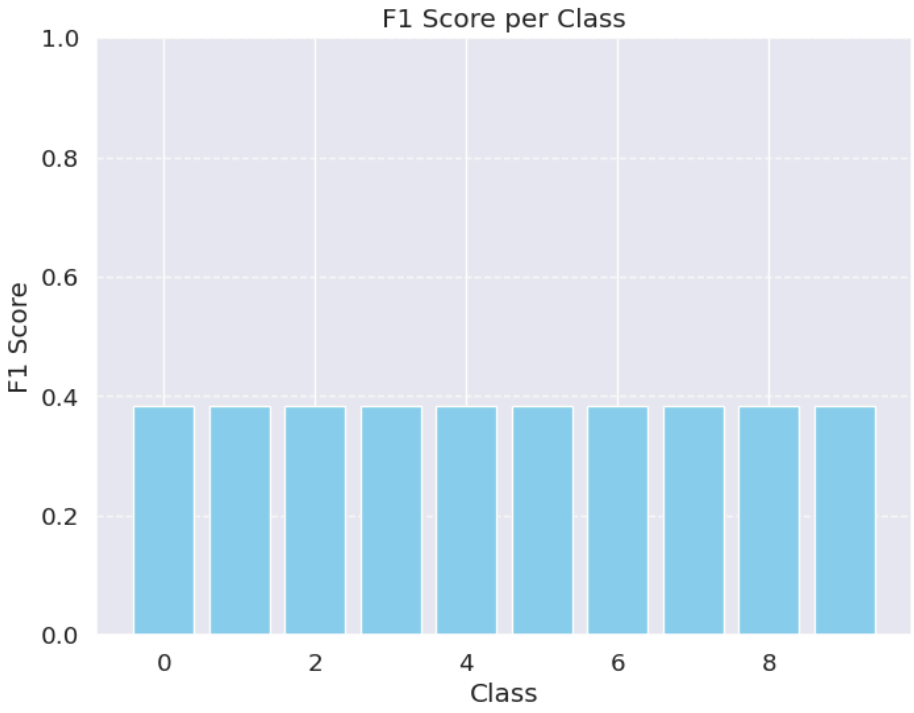
AUC ROC Curve



Confusion Matrix

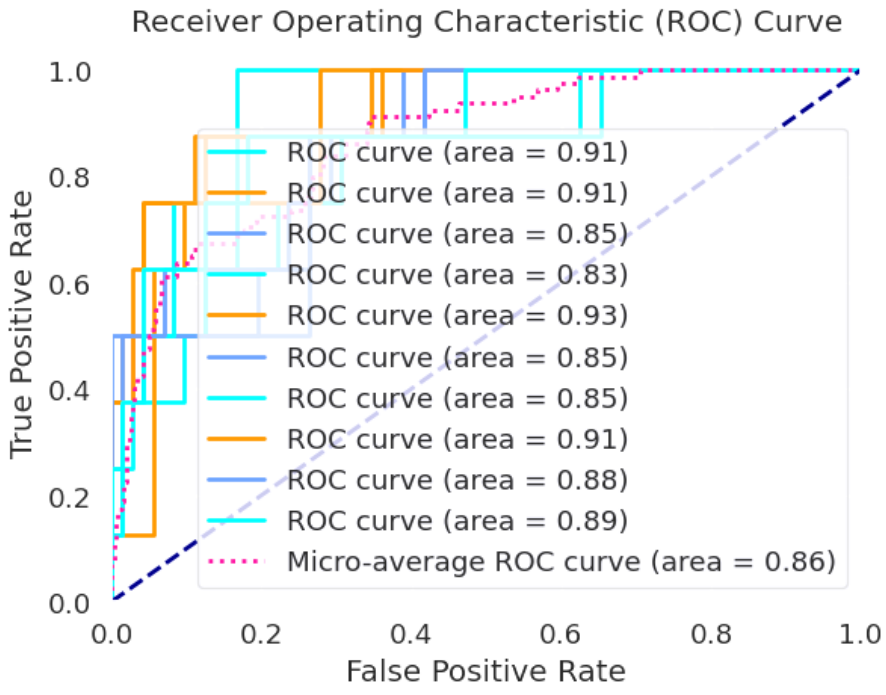


F1 Score

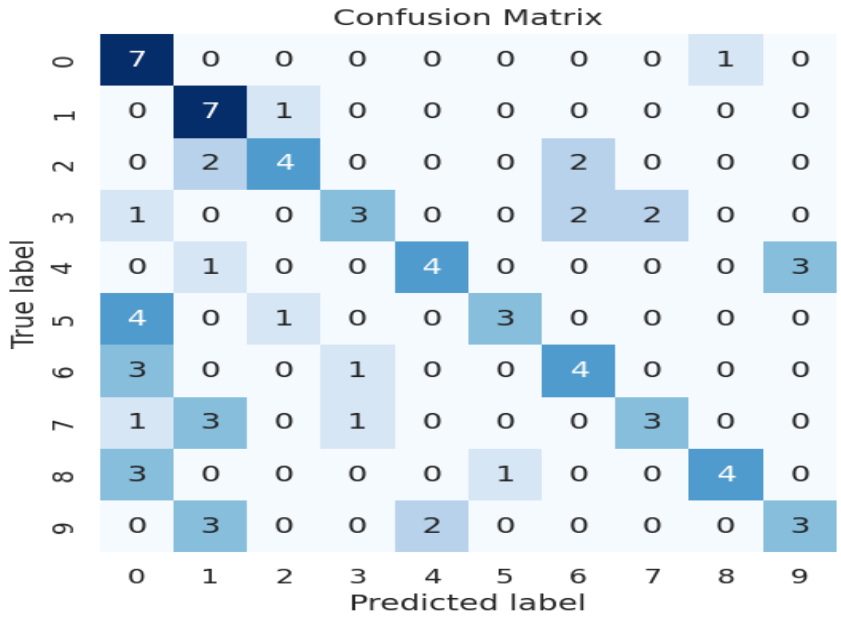


When Validation Fold is 3:

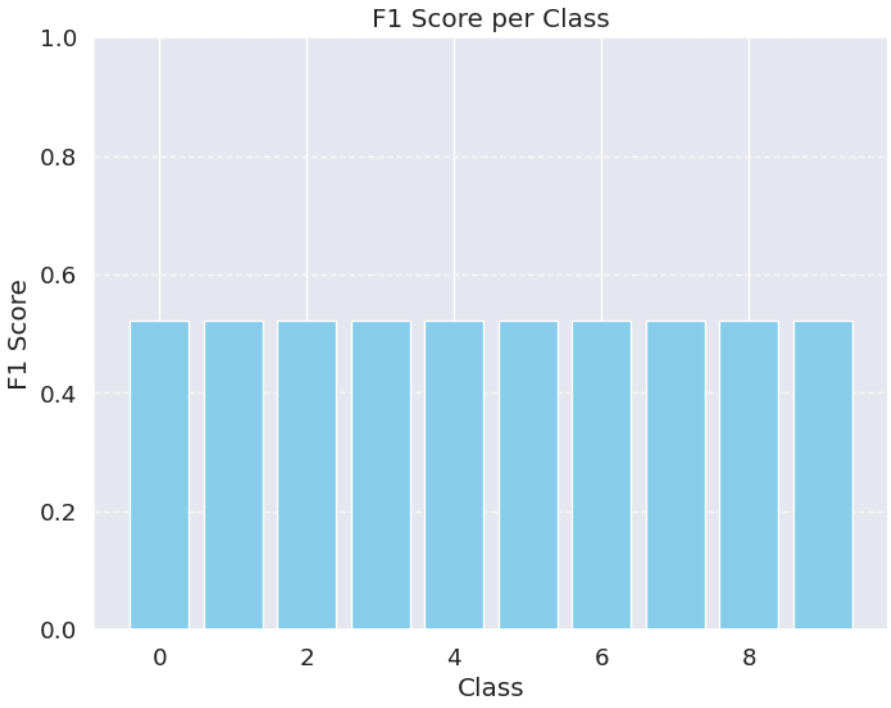
AUC ROC Curve



Confusion Matrix

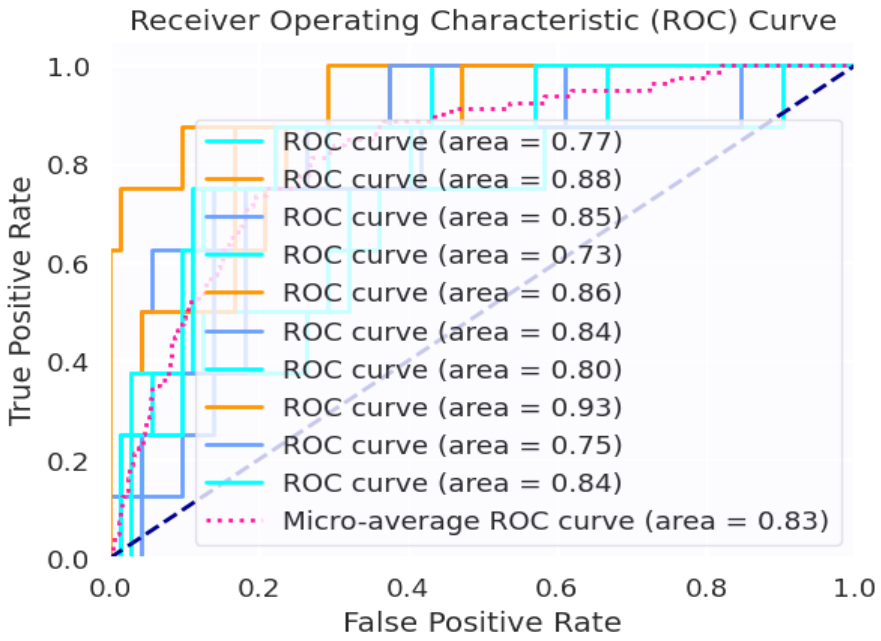


F1 Score

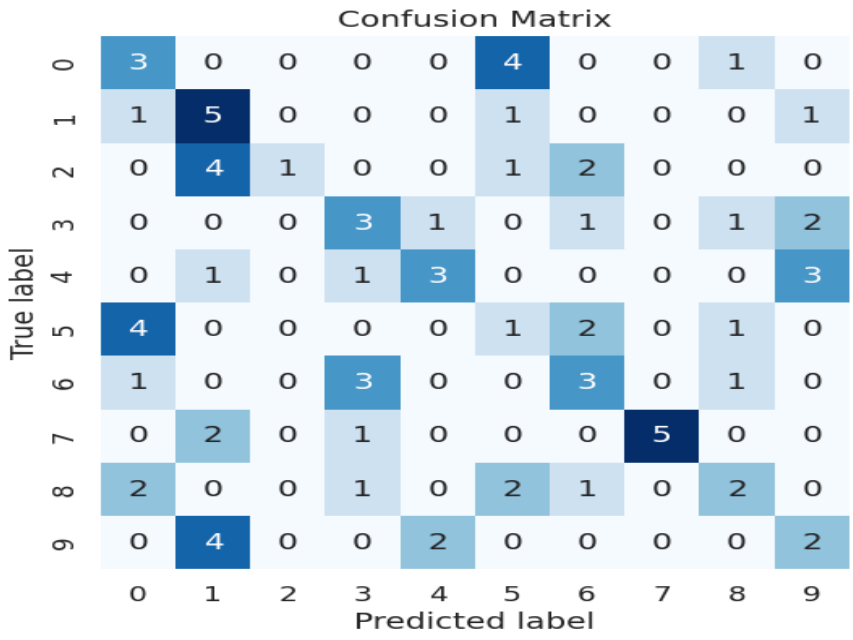


When Validation Fold is 4:

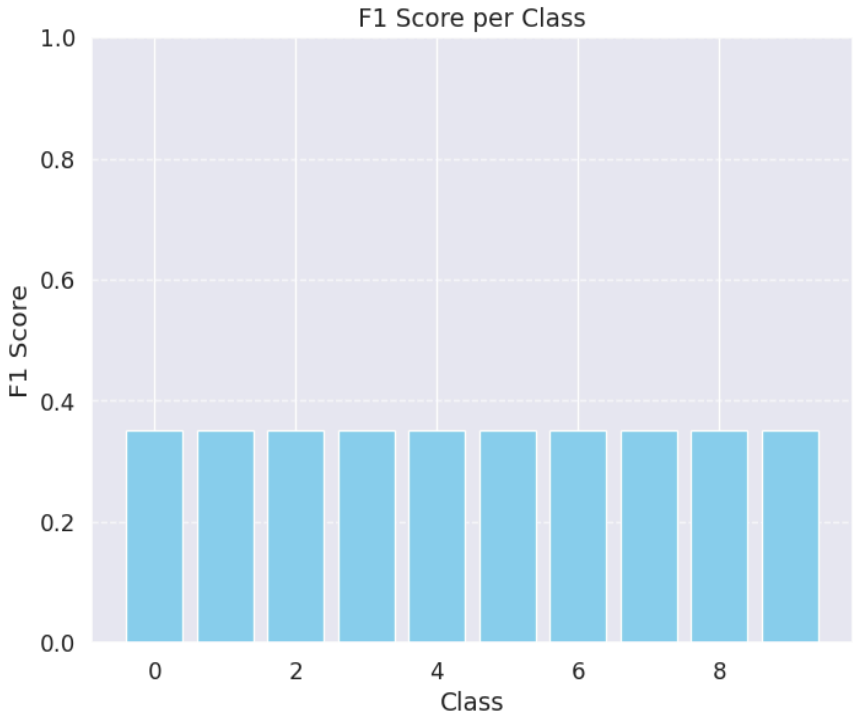
AUC ROC Curve



Confusion Matrix

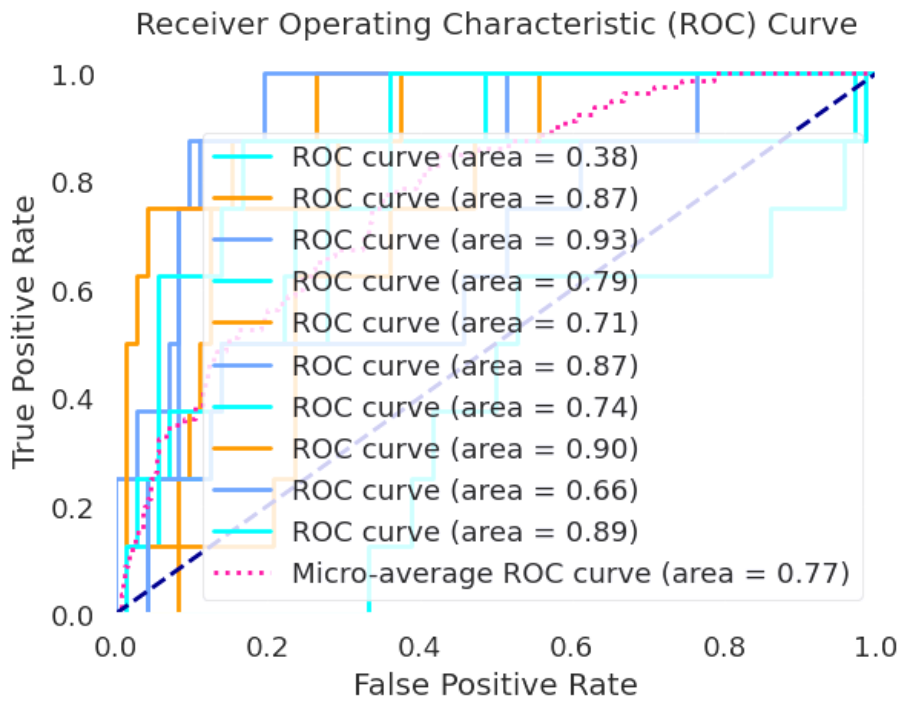


F1 Score

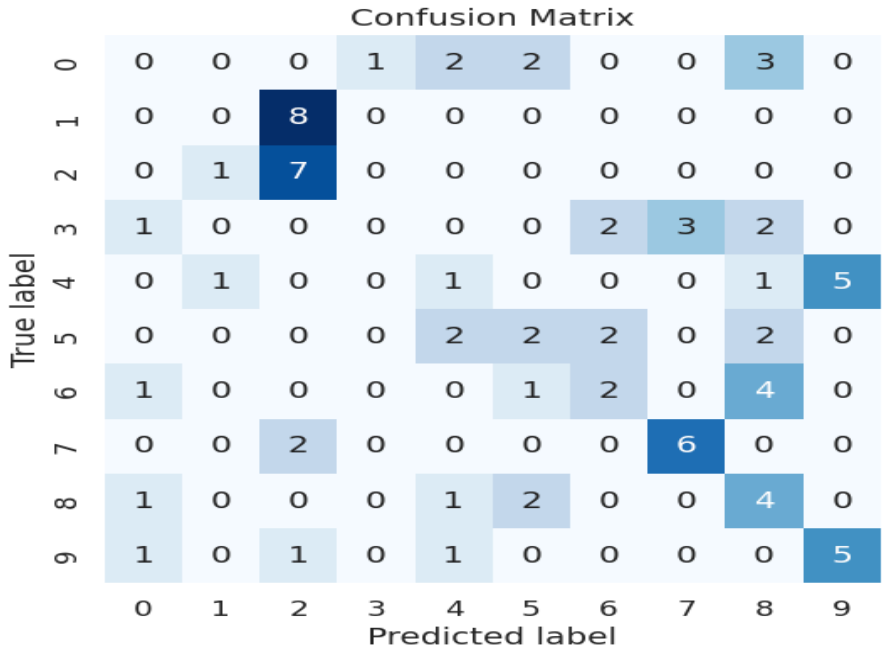


When Validation Fold is 5:

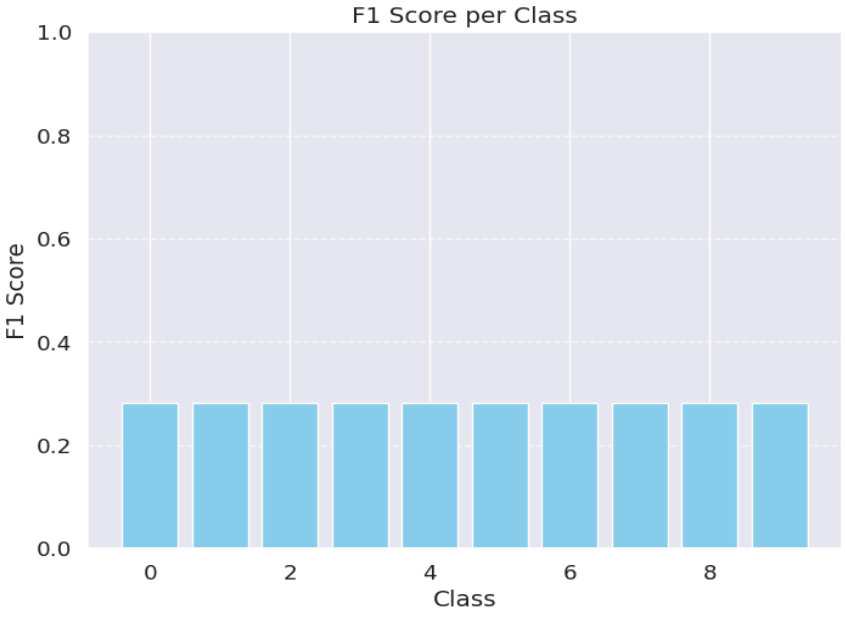
AUC ROC Curve



Confusion Matrix



F1 Score



Model Summary:

	No. of Trainable Parameters	No. of non Trainable Parameters
1d CNN	352354	0
Transformer (1 head)	417686	0
Transformer (2 head)	232430	0
Transformer(4 head)	139802	0

Model parameters will remain the same in Normal Train Test split and KFold validation. Because these two techniques are linked with the dataset rather than model itself.

Average Loss and Accuracy Table

	Avg. Train Loss(last epoch)	Avg. Valid. Loss(last epoch)	Avg. Test Loss(last epoch)	Avg. Train Accu.(last epoch)	Avg. Valid Accu.(last epoch)	Avg. Test Accu.(last epoch)
CNN	0.0010	2.1284	2.5475	100%	47.50%	37.50%
Transformer (1 head)	1.1075	1.7828	1.5763	58.75%	40.00%	42.5%
Transformer (2 head)	0.1829	2.7641	2.6626	91.67%	51.25%	48.75%
Transformer (4 head)	0.1062	3.5677	3.8441	95.42%	47.50%	42.5%

CNN (KFold)	0.0018	2.2673	2.3265	100%	45.31%	45.93%
Transformer (1 head, KFold)	1.3523	1.4523	1.4812	45.32%	47.35%	45.89%
Transformer (2 head, KFold)	1.1823	2.5628	2.4421	79.81%	40.08%	39.8%
Transformer (4 head, KFold)	0.3011	3.2564	3.6432	74.45%	43.54%	41.8%

Note: The average accuracies and losses in standard train and test splits are considered as the average accuracy and loss of 100th epoch. While in KFold validation, the average accuracy and loss are considered as the average loss and accuracy of the last epoch of every fold.

Hyperparameter Tuning:

The only hyperparameter considered in all four architectures is learning rate.
Learning rates vs Validation accuracy at 15th epoch is tabulated below:

		LR = 0.0001	**LR = 0.0005**	**LR = 0.001**	**LR = 0.005**
1d CNN	**Val. Accuracy**	38.75	43.75%	42.5%	45%
	Val. Loss	1.509	1.553	1.6404	2.372
Transformer (1 head)	**Val. Accuracy**	18.75	41.25	38.75%	32.5%
	Val. Loss	2.0378	1.804	1.817	1.8739
Transformer (2 head)	**Val. Accuracy**	26.25%	43.75%	30.00%	23.75%
	Val. Loss	1.949	1.849	1.609	2.050
Transformer(4	**Val. Accuracy**	31.25%	47.5	36.25	21.25

head)	**Val. Loss**	1.849	1.490	1.750	1.902

For 1d CNN, we can see, 0.0005 is outperforming all learning rate by making balanced accuracy with lesser loss.
For 1 head transformer, we can observe again 0.0005 is better choice.
Same we can see in 2 heads and 3 heads as well.
So the best learning rate is 0.0005.

Observation:

We can see that in CNN architecture, the model is highly overfitted. The train accuracy is about 100 percent while test and validation accuracies are significantly low. Hence, out model is not considered appropriate. When we had used KFold and measured the average accuracies, it came out to be approximately the same.

In the case of the Transformer, where head is equal to 1, we can observe that our training set , testing set and validation set are somewhat closer in terms of accuracy and losses in both normal train test split and K fold. Although accuracy and loss are not very good, we can conclude that our model is not very overfitted. Since transformers are heavy architecture, they require an ample amount of dataset to train. But in our case the dataset is very small. Hence the accuracy is lower.

In the case of 2 head and 4 head transformers, we can observe that they are also highly overfitted. 2 head and 4 head architectures are more complex than 1 head. Hence, they memorized the whole training data and performed badly on the test and validation set.

Conclusion:

In this assignment, we learned how to build a 1-dimensional convolutional neural network (CNN) and a transformer model from scratch. These models are commonly used in handling sequential data like audio. By implementing them ourselves, we gained a deeper understanding of how they work.

We also learned about KFold validation, which helps us evaluate our models more effectively by splitting the data into different folds for training and testing.

Additionally, we explored hyperparameter tuning, which involves adjusting parameters like learning rate or number of layers to improve the performance of our models.

Overall, this assignment provided us with valuable insights into the inner workings of CNNs and transformers, as well as important techniques for evaluating and optimizing our models.