# REPORT : ASSIGNMENT 1, DEEP LEARNING

**NAME : DIPAN MANDAL**
**ROLL NO.: M23CSA009**
**M.Tech Artificial Intelligence**

---

## Colab link:

https://colab.research.google.com/drive/1w5tJsuOP-1EpoFMJVqJlvaM6yQcozXcU?usp=sharing

**(Due to GPU constraints I have run two different notebooks to get the results, this notebook only contains the runs involving the transformers. The results are mentioned in the report)**

The objective of this assignment is to create 2 architectures for classification of a dataset containing 400 environmental audio recordings that are divided into 10 classes.

In the first architecture, we use a 1D convolution based neural network with a MLP as a classification head to perform the classification.

In the second architecture, we use a transformer encoder on top of the already existing convolutional neural network to get more meaningful features. On top of that we use an MLP head to carry out the classification task.

We train both the architectures for 100 epochs and use 4-fold cross-validation for getting more generalized results.

**Data Preparation:**

The code for the data preparation was already provided along with the problem statement. But we did make some changes in the collate function to stack all the 9 channels of the data (using `torch.stack()`) into one channel which made the final dimension of the input to the architectures as **(1, 144000).**

**Hyperparameter tuning:**

Mostly we experimented with the number of layers in the CNN, the kernel size of convolution layers and max pooling layers and the learning rate to find the most suitable parameters for our experiments.

## Architecture 1:

The architecture contains an overall 4 convolution, with each 1D convolution layer is followed by a ReLU, max pooling and a batch normalization layer.
These were the features of the convolution layers:

**Layer1:** in channels : 1, out channels : 8, kernel size : 11, padding : 5, max pooling kernel size : 5
**Layer2:** in channels : 8, out channels : 16, kernel size : 11, padding : 5, max pooling kernel size : 5
**Layer3:** in channels : 16, out channels : 32, kernel size : 7, padding : 3, max pooling kernel size : 5
**Layer4:** in channels : 32, out channels : 64, kernel size : 5, padding : 2, max pooling kernel size : 5

This was followed by a adaptive average pooling layer which made the output of the neural network to be **(64, 128)**

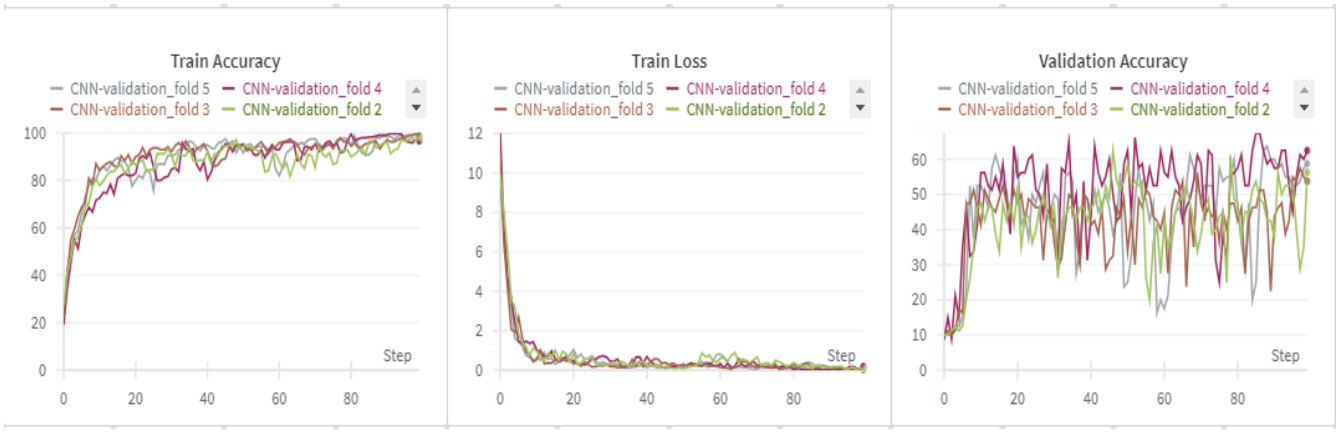
For the classification we used 2 fully connected layers in which the number of neurons in the first layer was 64 and the no. of neurons in the 2nd layer was equal to the number of classes i.e. 10.

## Results:

The above architecture was run for **100 epochs** using **4-fold cross validation**.
**Loss function:** categorical cross-entropy
**Optimizer:** Adam
**Learning rate:** 0.01
After 100 epochs there are the results that we got:

| Fold | Training loss | Training accuracy(%) | Validation accuracy(%) | Test accuracy(%) |
|---|---|---|---|---|
| **Fold 2** | 0.0709 | 97.92 | 56.25 | 46.25 |
| **Fold 3** | 0.0249 | 98.75 | 53.75 | 46.25 |
| **Fold 4** | 0.1871 | 96.67 | 62.50 | 48.75 |
| **Fold 5** | 0.0584 | 97.50 | 58.75 | 55.00 |

Here is the progression of the training error, training accuracy and validation accuracy over 100 epochs:

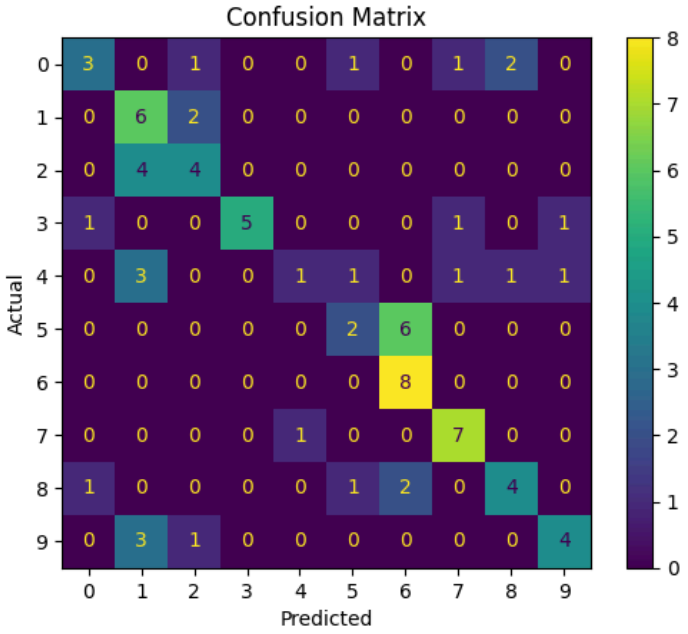


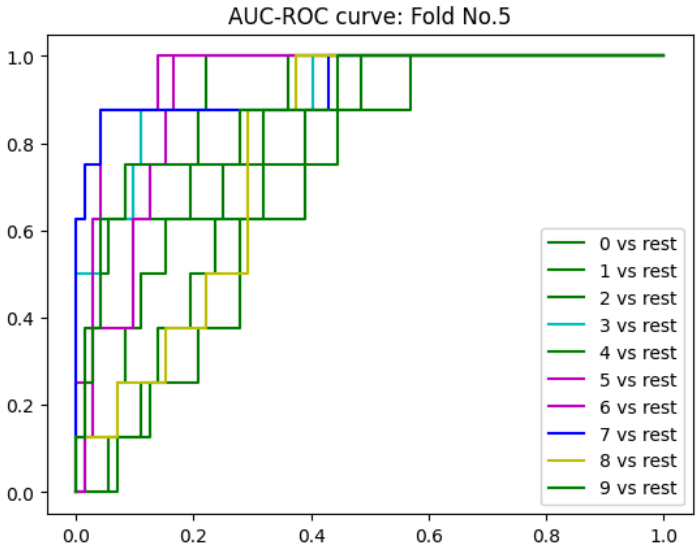

***Source: weights and biases platform***

Here are the **F1 score, confusion matrix and ROC curve** for the best performing fold (fold 5)

F1 score :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.60 | 0.38 | 0.46 | 8 |
| 1 | 0.40 | 1.00 | 0.57 | 8 |
| 2 | 0.50 | 0.12 | 0.20 | 8 |
| 3 | 0.29 | 0.25 | 0.27 | 8 |
| 4 | 0.40 | 0.25 | 0.31 | 8 |
| 5 | 0.00 | 0.00 | 0.00 | 8 |
| 6 | 0.41 | 0.88 | 0.56 | 8 |
| 7 | 0.86 | 0.75 | 0.80 | 8 |
| 8 | 0.40 | 0.75 | 0.52 | 8 |
| 9 | 1.00 | 0.25 | 0.40 | 8 |






**Observations:** As we can clearly see, the model is overfitting as the training accuracy is much higher than the validation or test accuracy. This phenomenon will be happening in all the future architectures.
One main reason for this behavior can be the **small size of the dataset** as there are **only 240 samples** in the training set after splitting the data into test and validation sets.

## Total number of trainable parameters in architecture 1: 540,682

## Architecture 2:

In the architecture 2 we have used a transformer encoder on top of the pre-existing CNN along with a 2 layer classification head.
For the multihead self attention blocks in the transformer architecture, we have experimented with 1,2 and 4 attention heads.
For each architecture, we have used 2 attention blocks to make the complete structure.

Also while doing the classification, instead of passing the whole output of the encoder, we have only passed a special **CLS_token** which is a trainable parameter and captures the information present in the encoder output.

**Loss function:** categorical cross entropy
**Optimizer:** Adam
**Learning rate:** 0.001

## Results:

These are the results that we got after 100 epochs for architecture with **4 self-attention heads:**

| Fold | Training loss | Training accuracy(%) | Validation accuracy(%) | Test accuracy(%) |
|---|---|---|---|---|
| **Fold 2** | 0.0022 | 100 | 47.50 | 51.25 |
| **Fold 3** | 0.0004 | 100 | 58.75 | 65.00 |
| **Fold 4** | 0.0006 | 100 | 48.75 | 56.25 |
| **Fold 5** | 0.0089 | 100 | 60.00 | 50.00 |

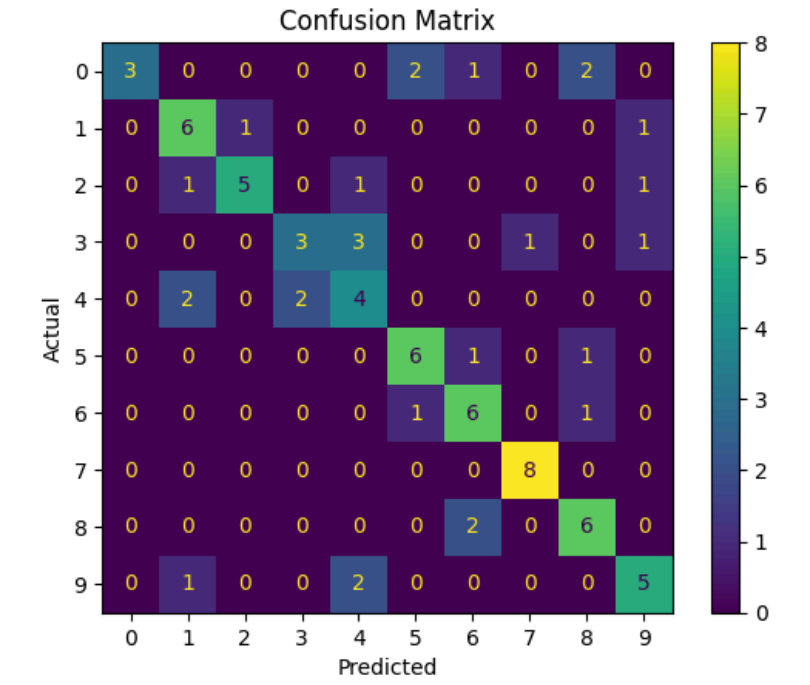
**Progression of the training error, training accuracy and validation accuracy over 100 epochs:**

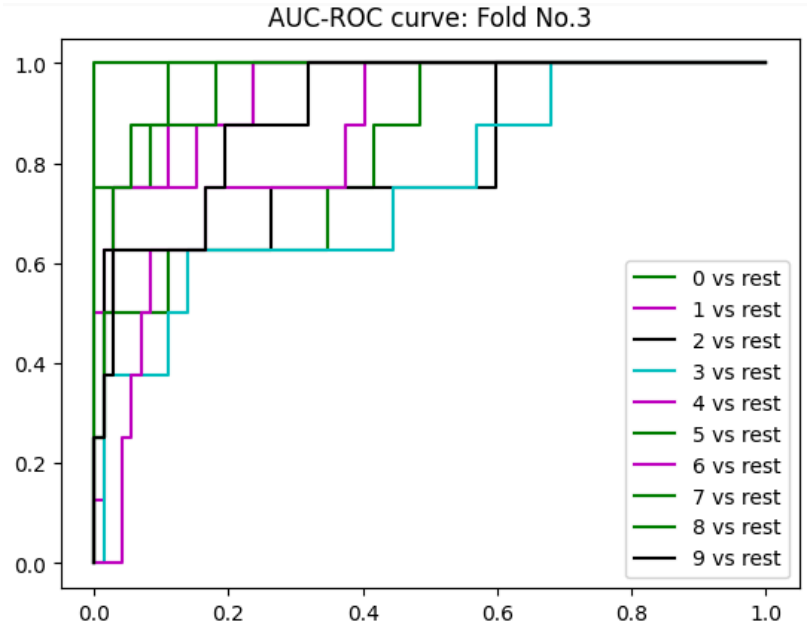

***Source: weights and biases platform***

**F1 score, confusion matrix and ROC curve(best performing fold):**

F1 score :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.38 | 0.55 | 8 |
| 1 | 0.60 | 0.75 | 0.67 | 8 |
| 2 | 0.83 | 0.62 | 0.71 | 8 |
| 3 | 0.60 | 0.38 | 0.46 | 8 |
| 4 | 0.40 | 0.50 | 0.44 | 8 |
| 5 | 0.67 | 0.75 | 0.71 | 8 |
| 6 | 0.60 | 0.75 | 0.67 | 8 |
| 7 | 0.89 | 1.00 | 0.94 | 8 |
| 8 | 0.60 | 0.75 | 0.67 | 8 |
| 9 | 0.62 | 0.62 | 0.62 | 8 |



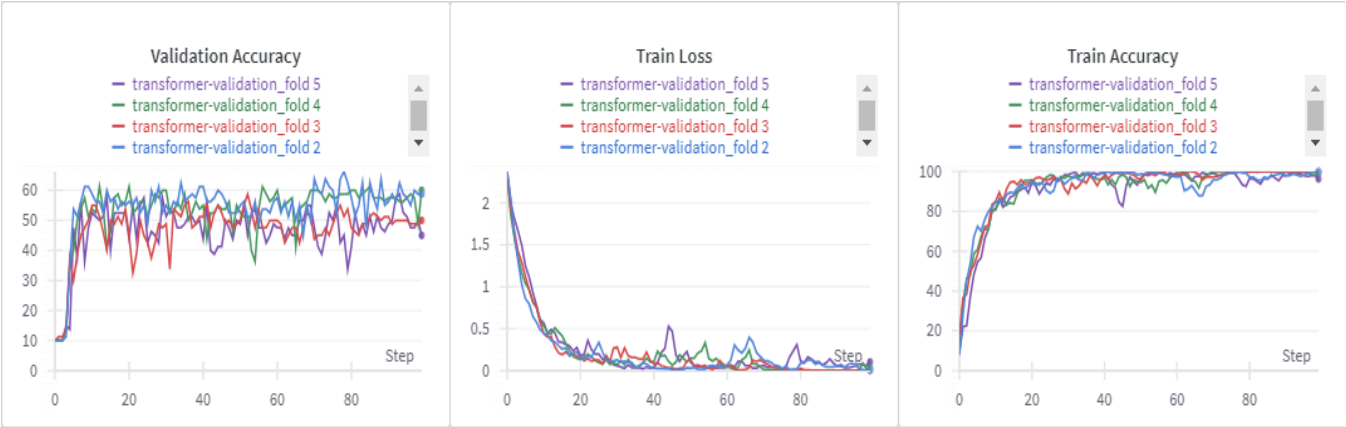

**Observations:**

Although the transformer model overfitted, the test accuracy is still better than the test accuracy of the CNN model. The number of parameters is also less in this architecture.

For architecture with **2 self-attention heads:**

| Fold | Training loss | Training accuracy(%) | Validation accuracy(%) | Test accuracy(%) |
|---|---|---|---|---|
| **Fold 2** | 0.0065 | 100 | 58.75 | 51.25 |
| **Fold 3** | 0.0008 | 100 | 50.00 | 52.50 |
| **Fold 4** | 0.0006 | 100 | 60.00 | 47.5 |
| **Fold 5** | 0.0089 | 100 | 45.00 | 47.5 |

**Progression of the training error, training accuracy and validation accuracy over 100 epochs:**

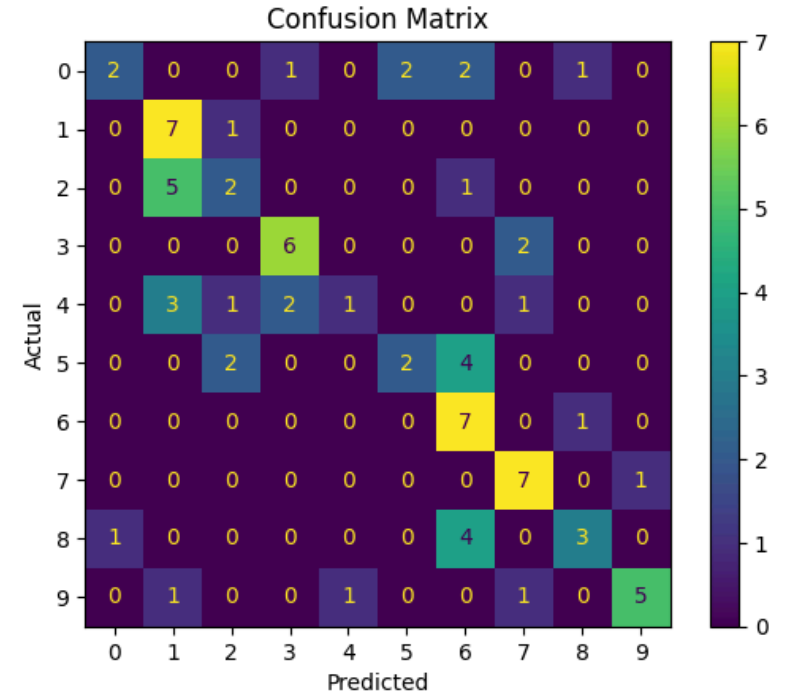


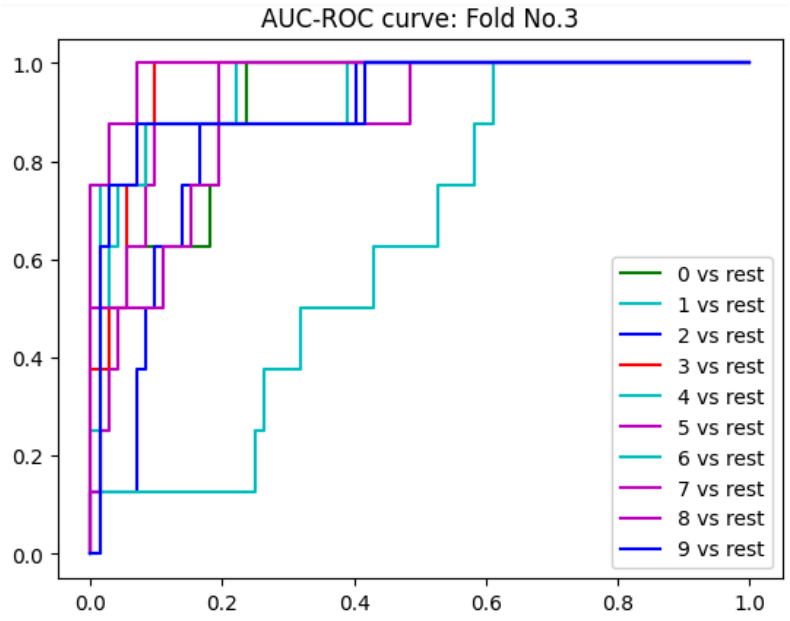

***Source: weights and biases platform***

**F1 score, confusion matrix and ROC curve(best performing fold):**

F1 score :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.67 | 0.25 | 0.36 | 8 |
| 1 | 0.44 | 0.88 | 0.58 | 8 |
| 2 | 0.33 | 0.25 | 0.29 | 8 |
| 3 | 0.67 | 0.75 | 0.71 | 8 |
| 4 | 0.50 | 0.12 | 0.20 | 8 |
| 5 | 0.50 | 0.25 | 0.33 | 8 |
| 6 | 0.39 | 0.88 | 0.54 | 8 |
| 7 | 0.64 | 0.88 | 0.74 | 8 |
| 8 | 0.60 | 0.38 | 0.46 | 8 |
| 9 | 0.83 | 0.62 | 0.71 | 8 |



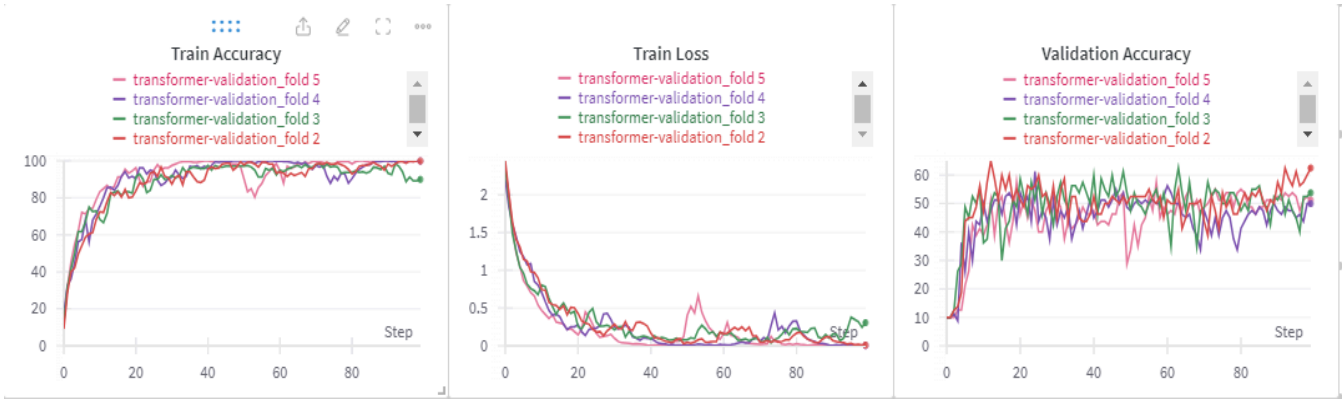

**Observation:**

The transformer model with 2 heads performs worse overall as compared to the model with 4 heads. But the performance was very much unpredictable, as we can see from the validation graph progression.

For the model with **1 self-attention head:**

| Fold | Training loss | Training accuracy(%) | Validation accuracy(%) | Test accuracy(%) |
|---|---|---|---|---|
| **Fold 2** | 0.0080 | 100 | 62.50 | 46.50 |
| **Fold 3** | 0.3060 | 90 | 53.75 | 36.25 |
| **Fold 4** | 0.0033 | 100 | 50.00 | 60.00 |
| **Fold 5** | 0.0014 | 100 | 51.25 | 50.00 |

**Progression of the training error, training accuracy and validation accuracy over 100 epochs:**

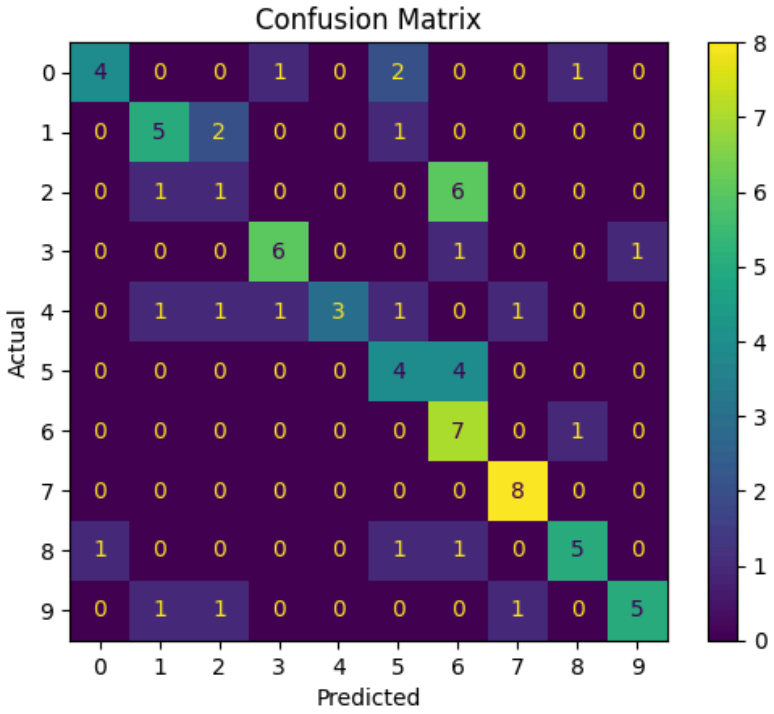


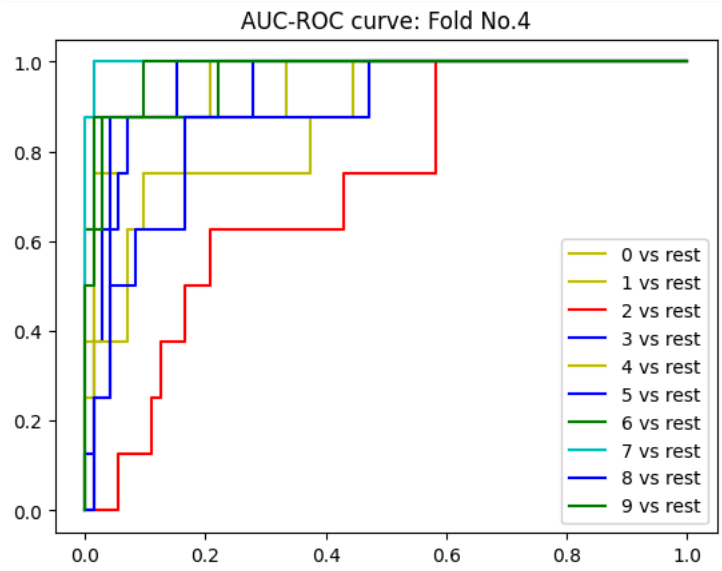

***Source: weights and biases platform***

**F1 score, confusion matrix and ROC curve(best performing fold):**

F1 score :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.80 | 0.50 | 0.62 | 8 |
| 1 | 0.62 | 0.62 | 0.62 | 8 |
| 2 | 0.20 | 0.12 | 0.15 | 8 |
| 3 | 0.75 | 0.75 | 0.75 | 8 |
| 4 | 1.00 | 0.38 | 0.55 | 8 |
| 5 | 0.44 | 0.50 | 0.47 | 8 |
| 6 | 0.37 | 0.88 | 0.52 | 8 |
| 7 | 0.80 | 1.00 | 0.89 | 8 |
| 8 | 0.71 | 0.62 | 0.67 | 8 |
| 9 | 0.83 | 0.62 | 0.71 | 8 |




**Observation:**

There were **fluctuations in the performance** of the model as in one fold it performed worst among all the models and in one fold it performed as good as the model with 4 self-attention heads.

## **Number of trainable parameters in architecture 2:** 421,130

(the number of parameters is **independent** of number of heads)

## Overall Comparison Between Models:

Here is a comparison between the best performance of all the models that we have used in the assignment. We have considered the fold that gave the best performance for each model.

| Model | Training loss | Training accuracy(%) | Validation accuracy(%) | Test accuracy(%) |
|---|---|---|---|---|
| **1D CNN** | 0.0584 | 97.50 | 58.75 | **55.00** |
| **Transformer: 1 head** | 0.0033 | 100 | 50.00 | 60.00 |
| **Transformer: 2 heads** | 0.0008 | 100 | 50.00 | 52.50 |
| **Transformer: 4 heads** | 0.0004 | 100 | 58.75 | **65.00** |

From the performance comparison we can clearly see that the transformer in general outperforms the CNN based architecture, despite having a lower number(almost half) of parameters.

This may happen due to a few reasons:

1. Transformers are inherently designed to capture **long-range dependencies** in sequences. They utilize self-attention mechanisms that allow each token to attend to all other tokens in the sequence, enabling them to effectively capture relationships between distant tokens. In contrast, CNNs have **limited receptive fields**, which may struggle to capture long-range dependencies.

2. Transformers can achieve **strong performance with fewer parameters compared to CNNs**, especially in tasks involving sequential data. This is because transformers share parameters across all positions in the sequence, whereas CNNs require separate parameters for each position in the input.

3. Transformers explicitly incorporate **positional information through positional encodings**. This helps the model understand the order of tokens in the sequence, which is crucial for tasks where the spatial information is important(like the one given in the assignment)