Deep Learning Assignment 2

Bikram Majhi (M23CSA007)

February 2024

Google Colab Link

1D CNN Model

The proposed 1D-CNN architecture is effective for tasks like audio classification due to its ability to learn hierarchical feature representations, its parameter efficiency, and its translation invariance. The fully connected layer at the end allows for multi-class classification. The architecture is flexible, allowing adjustments to suit specific tasks.

Figure 1: 1D-CNN Model Training loss, fold=2

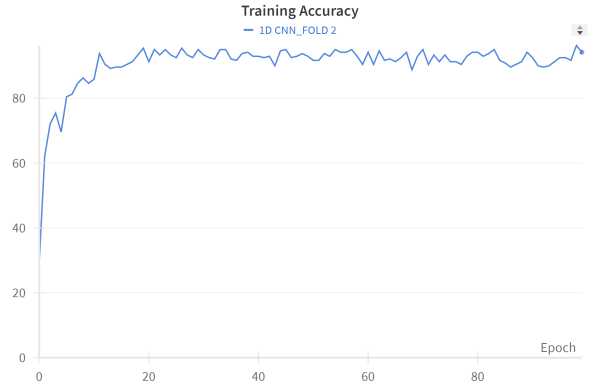

Figure 2: 1D-CNN Model Training accuracy, fold=2

Figure 3: 1D-CNN Model Training loss, fold=3

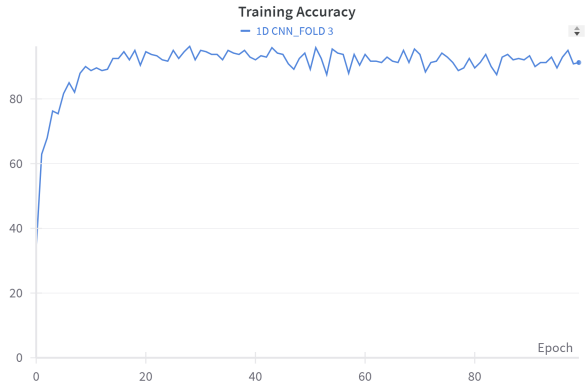

Figure 4: 1D-CNN Model Training accuracy, fold=3

Figure 5: 1D-CNN Model Training loss, fold=4

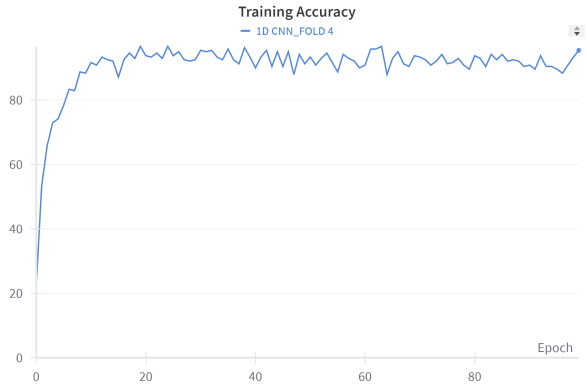

Figure 6: 1D-CNN Model Training accuracy, fold=4

Figure 7: 1D-CNN Model Training loss, fold=5

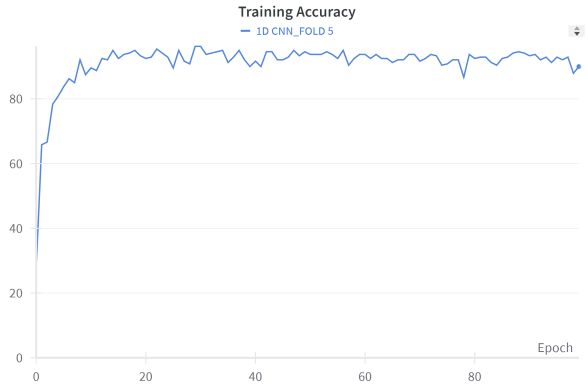

Figure 8: 1D-CNN Model Training accuracy, fold=5

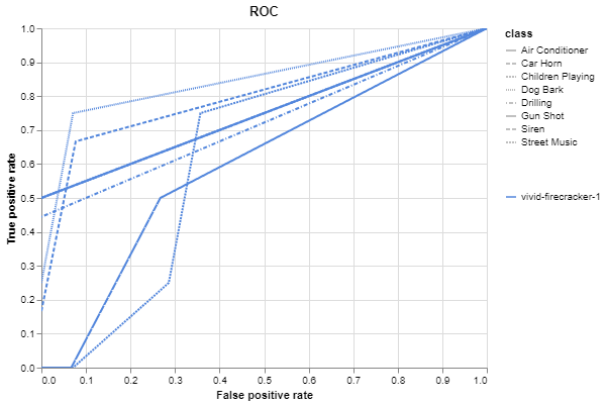

Figure 9: 1D-CNN Model ROC curve

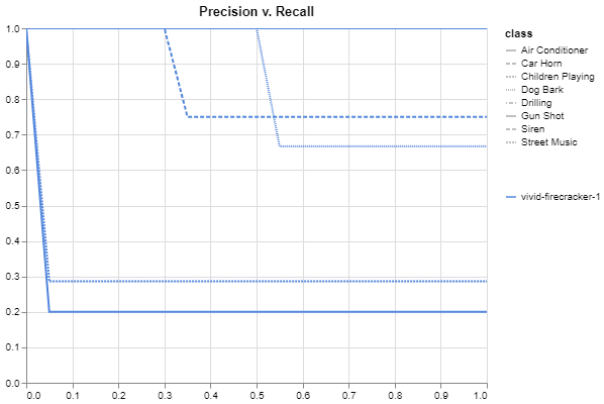

Figure 10: 1D-CNN Model Precision-Recall curve

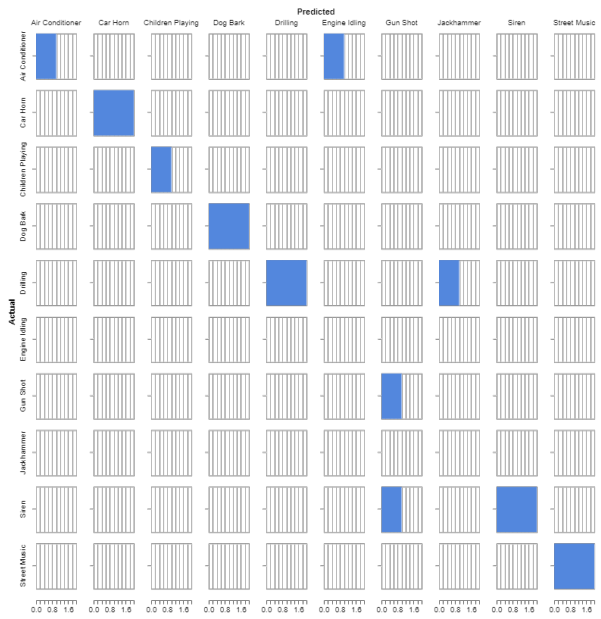

Figure 11: 1D-CNN Model Confusion Matrix

Figure 12: 1D-CNN Transformer Model Training loss, fold=2, head=1

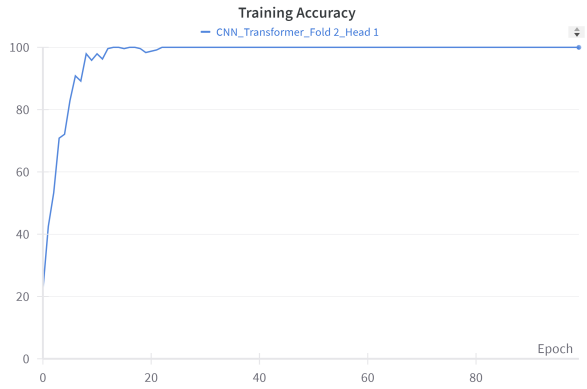

Figure 13: 1D-CNN Transformer Model train accuracy,fold=2,head=1

Figure 14: 1D-CNN Transformer Model Training loss, fold=2, head=2

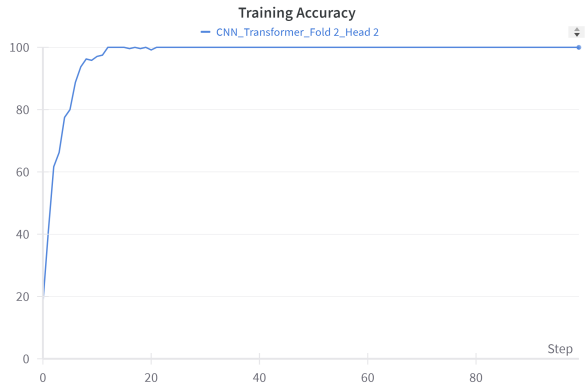

Figure 15: 1D-CNN Transformer Model train accuracy, fold=2,head=2

Figure 16: 1D-CNN Transformer Model Training loss, fold=3, head=1

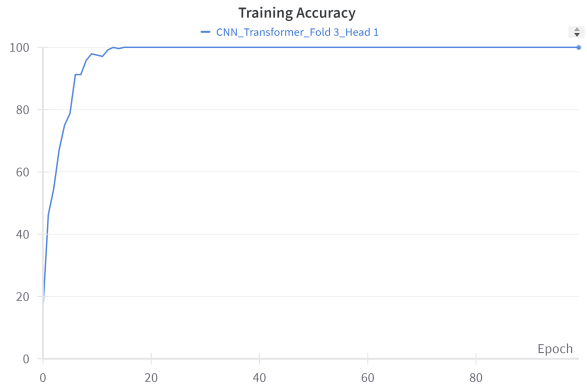

Figure 17: 1D-CNN Transformer Model train accuracy, fold=3,head=1

Figure 18: 1D-CNN Transformer Model training loss, fold=3, head=2

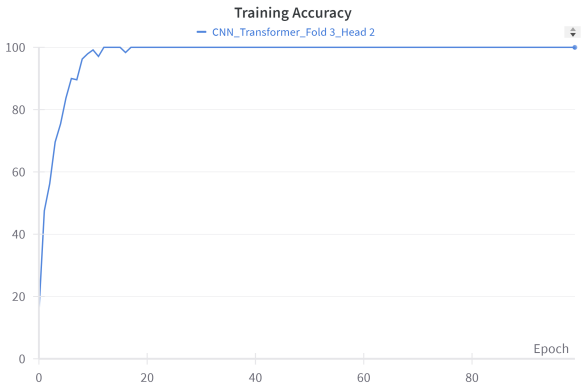

Figure 19: 1D-CNN Transformer Model train accuracy, fold=3,head=2

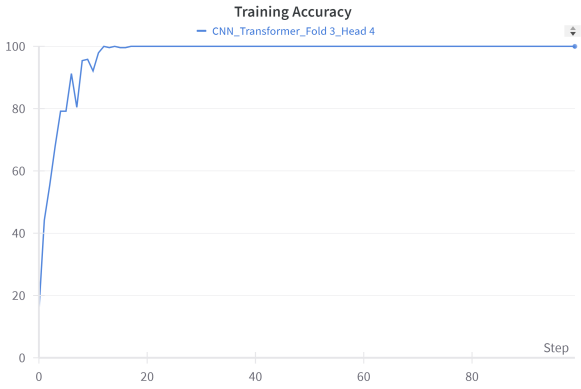

Figure 20: 1D-CNN Transformer Model train accuracy, fold=3, head=4

Figure 21: 1D-CNN Transformer Model training loss, fold=3, head=4

Figure 22: 1D-CNN Transformer Model Training loss, fold=4, head=1

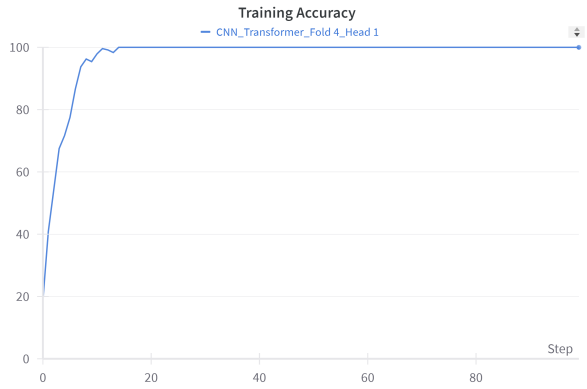

Figure 23: 1D-CNN Transformer Model train accuracy, fold=4,head=1

Figure 24: 1D-CNN Transformer Model training loss, fold=4, head=2

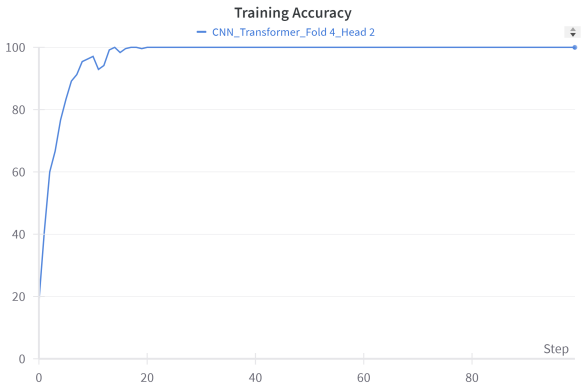

Figure 25: 1D-CNN Transformer Model train accuracy, fold=4,head=2

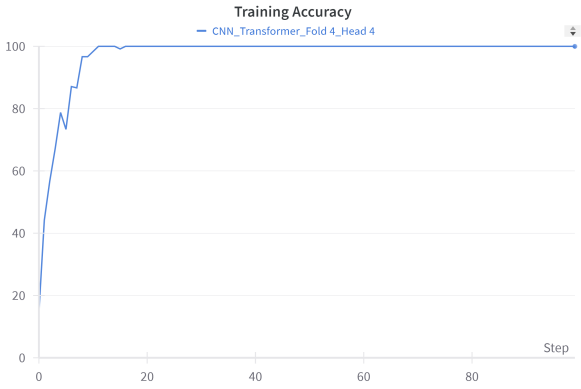

Figure 26: 1D-CNN Transformer Model train accuracy, fold=4, head=4

Figure 27: 1D-CNN Transformer Model training loss, fold=4, head=4

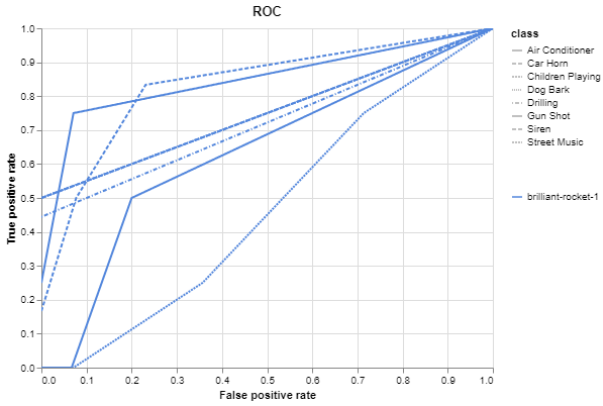

Figure 28: 1D-CNN Model ROC curve

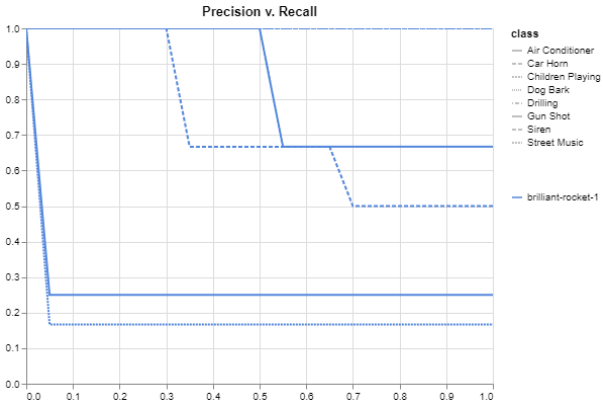

Figure 29: CNN Transformer Model Precision-Recall curve

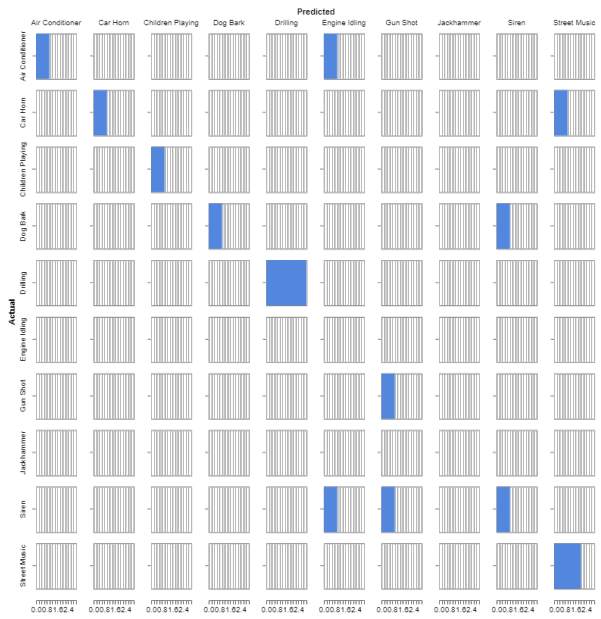

Figure 30: CNN Transformer Model Confusion Matrix

Figure 31: 1D-CNN Transformer Model Training loss, fold=2, head=1

Table 1: Run Summary of 1D-CNN for Fold 2, Fold 4, and Fold 5

Metric	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**
Epoch	99	99	99	99
Test Accuracy (%)	42.5	60.0	52.5	52.5
Test F1 Score	0.67708	0.72292	0.69583	0.8625
Test Loss	8.41874	5.36506	6.65996	7.1779
Training Accuracy (%)	94.16667	91.25	95.41667	90
Training Loss	2.2967	3.27856	1.72893	3.5525
Validation Accuracy (%)	42.5	50.0	51.25	51.25
Validation Loss	8.64688	8.26779	6.71342	7.3346
Average Val Acc (%)	**42.5**	**46.25**	**47.91667**	**48.75**

Table 2: Performance Summary of CNN-Transformer architecture for Fold 2

Metric	**Head 1**	**Head 2**	**Head 4**
Epoch	99	99	99
Test Accuracy (%)	57.5	60.0	55.0
Test F1 Score	0.69375	0.67083	0.56667
Test Loss	9.15975	8.919	10.32059
Training Accuracy (%)	100.0	100.0	100.0
Training Loss	0.00643	0.00578	0.0094
Validation Accuracy (%)	61.25	65.0	63.75
Validation Loss	7.87568	6.95919	8.32973
Average Val Acc (%)	**61.25**	**63.125**	**63.33333**

1D CNN + Transformer Model

The proposed architecture uses 1D convolutions for temporal feature extraction from audio signals. The transformer encoder network, with its self-attention mechanism, captures long-range dependencies in the data. The multi-head self-attention allows the model to focus on different parts of the input simultaneously. The MLP head performs the final classification task. The architecture's flexibility allows it to adapt to different complexity levels of the classification task. The model is flexible, robust to overfitting, and scalable. The architecture allows for adjustments in the number of layers, filters, kernel size, stride, pooling type, and other parameters to suit specific tasks. This makes it adaptable to a wide range of problems. The use of convolutional layers and weight sharing reduces the number of parameters, making the model less prone to overfitting. This means the model can generalize better to unseen data. The architecture can handle inputs of varying lengths, making it scalable to different audio clip durations. This is particularly useful in audio classification tasks where the length of audio clips can vary widely.

Table 3: Performance Summary of CNN-Transformer architecture for Fold 3

Metric	**Head 1**	**Head 2**	**Head 4**
Epoch	99	99	99
Test Accuracy (%)	53.75	60.0	51.25
Test F1 Score	0.50417	0.6125	0.73125
Test Loss	9.62385	8.29306	10.02546
Training Accuracy (%)	100.0	100.0	100.0
Training Loss	0.00916	0.00536	0.00551
Validation Accuracy (%)	56.25	63.75	63.75
Validation Loss	8.58272	8.55683	7.45118
Average Val Acc (%)	**61.5625**	**62.0**	**62.29167**

Table 4: Performance Summary of CNN-Transformer architecture for Fold 4

Metric	**Head 1**	**Head 2**	**Head 4**
Epoch	99	99	99
Test Accuracy (%)	67.5	61.25	57.5
Test F1 Score	0.86042	0.68958	0.49196
Test Loss	7.20512	8.17299	7.29623
Training Accuracy (%)	100.0	100.0	100.0
Training Loss	0.00646	0.00537	0.00655
Validation Accuracy (%)	60.0	56.25	57.5
Validation Loss	6.45722	8.20615	7.65826
Average Val Acc (%)	**61.96429**	**61.25**	**60.83333**

Total trainable and non-trainable parameters.

For CNN Architecture

- Total Params: 641034
- Trainable Params: 641034
- Non-Trainable Params: 0

For CNN+Transformer Architecture

- Total Params: 3272714
- Trainable Params: 3272714
- Non-Trainable Params: 0

Table 5: Performance Summary of CNN-Transformer architecture for Fold 5

Metric	**Head 1**	**Head 2**	**Head 4**
Epoch	99	99	99
Test Accuracy (%)	65.0	57.5	58.75
Test F1 Score	0.75208	0.71667	0.675
Test Loss	7.56061	8.85283	8.29552424
Training Accuracy (%)	100.0	100.0	100
Training Loss	0.00512	0.00554	0.007601104
Validation Accuracy (%)	68.75	67.5	66.25
Validation Loss	7.72767	7.97602	7.32637465
Average Val Acc (%)	**61.625**	**62.15909**	**62.5**

Hyperparamter tuning using Grid Search

Grid Search is a traditional method for hyperparameter tuning. It works by defining a grid of hyperparameters and then evaluating model performance for each point on the grid. You can then choose the point that produces the best model.

- **Pros:**
  - It's simple and easy to implement.
  - It's exhaustive in nature, meaning it gives the best performance for the given hyperparameters.
- **Cons:**
  - It can be computationally expensive, especially with a large number of hyperparameters.
  - It doesn't work well with continuous hyperparameters or when the function to optimize is not smooth.

For this specific task, using Grid Search could be a good idea because it allows you to systematically work through multiple combinations of hyperparameters to find the best one. It's particularly useful when you have a small number of hyperparameters and each one has a relatively small number of values to test.

1D-CNN

The best result was achieved with a weight decay of 0.01 and Xavier weight initialization, yielding a training accuracy of 92.08% and a validation accuracy of 58.75%. On the other hand, the worst result was observed with a weight decay of 0.05 and random weight initialization, resulting in a training accuracy of 93.33% but a lower validation accuracy of 53.75%.

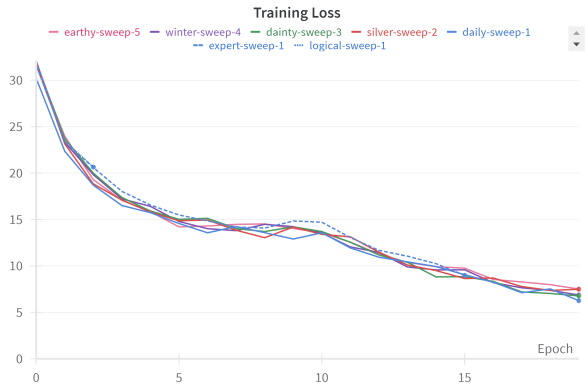

Figure 32: 1D-CNN Model hyperparamter tuning train loss

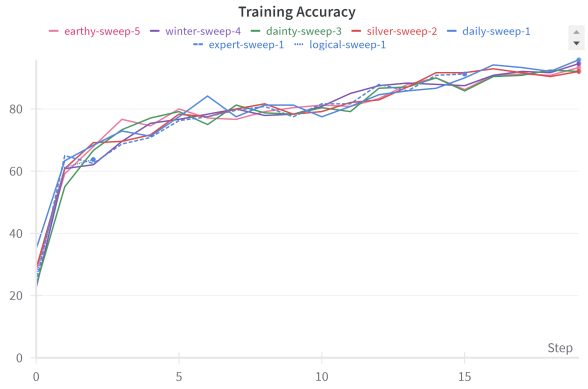

Figure 33: 1D-CNN Model hyperparamter tuning train accuracy

Figure 34: 1D-CNN Model hyperparamter tuning validation loss

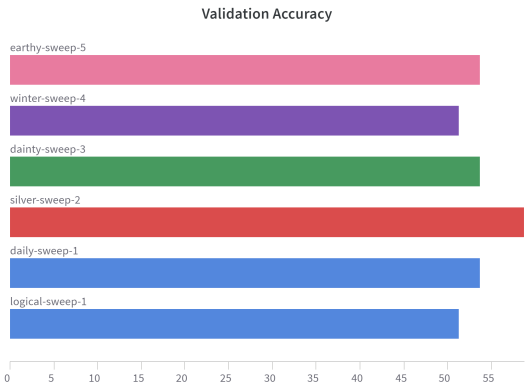

Figure 35: 1D-CNN Model hyperparamter tuning validation accuracy

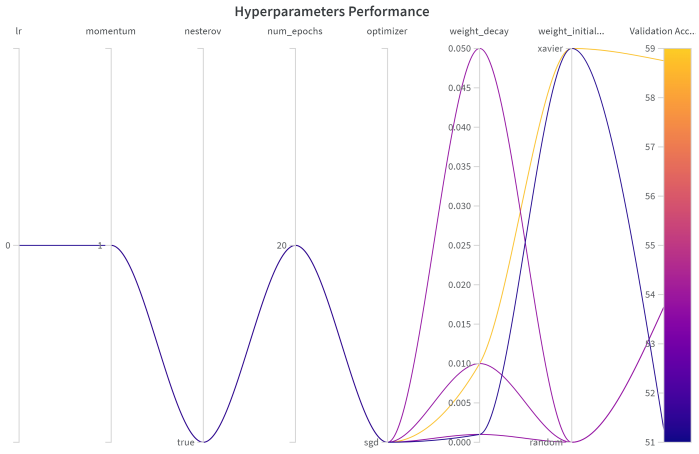

Figure 36: 1D-CNN Model hyperparamter tuning using Grid Seach

Hyperparameters	**Value**
Learning Rate (lr)	0.001
Momentum	0.9
Nesterov	True
Number of Epochs (num_epochs)	20
Optimizer	SGD
Weight Decay	0.01
Weight Initialization	Xavier

Table 6: Best Hyperparameters for the 1D-CNN Model

The higher validation accuracy in the first case suggests that the model generalizes better to unseen data. This could be attributed to the lower weight decay, which provides less regularization, and the Xavier initialization, which is known to provide a good starting point for the weights, leading to faster convergence and better performance.

In contrast, the second case, despite having a higher training accuracy, performed worse on the validation set. This could be due to overfitting, where the model learns the training data too well and performs poorly on unseen data. The higher weight decay in this case might have been too much regularization for the model, preventing it from learning the underlying patterns in the data effectively. Additionally, random weight initialization can lead to slower convergence or getting stuck in less optimal minima, resulting in poorer performance.

Hyperparameters	Run Summary				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.01 weight_init: random	Epoch 19	Train Acc. 95.83%	Train Loss 6.25843	Valid Acc. 53.75%	Valid Loss 6.0109

Table 7: Hyperparameters and Run Summary for the 1D-CNN.

Hyperparameters	Run Summary				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.01 weight_init: xavier	Epoch 20	Train Acc. 92.08%	Train Loss 7.48738	Valid Acc. 58.75%	Valid Loss 5.78984

Table 8: Hyperparameters and Run Summary for 1D-CNN.

1D-CNN + Transformer

The best result was achieved with a weight decay of 0.001 and Xavier weight initialization, yielding a training accuracy of 99.58% and a validation accuracy of 68.75%. The worst result was observed with a weight decay of 0.05 and random weight initialization, resulting in a training accuracy of 99.58% but a lower validation accuracy of 57.5%.

The higher validation accuracy in the first case suggests that the model generalizes better to unseen data. This could be attributed to the lower weight decay, which provides less regularization, and the Xavier initialization, which is known to provide a good starting point for the weights, leading to faster convergence and better performance.

In contrast, the second case, despite having a similar training accuracy, performed worse on the validation set. This could be due to overfitting, where the model learns the training data too well and performs poorly on unseen data. The higher weight decay in this case might have been too much regularization for the model, preventing it from learning the underlying patterns in the data effectively. Additionally, random weight initialization can lead to slower convergence or getting stuck in less optimal minima, resulting in poorer performance.

Figure 37: CNN-Transformer Model hyperparamter tuning train loss

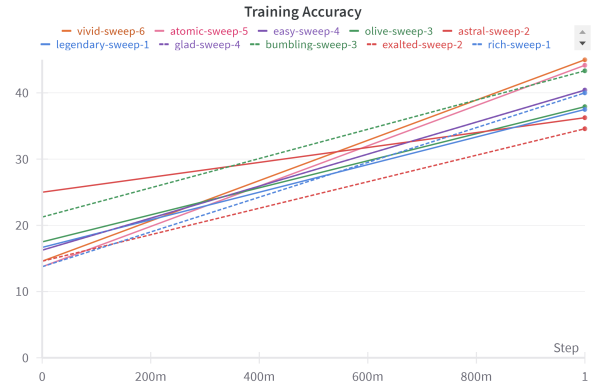

Figure 38: CNN-Transformer hyperparamter tuning train accuracy

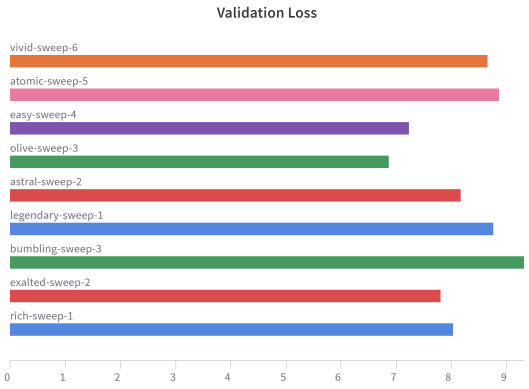

Figure 39: CNN-Transformer Model hyperparamter tuning validation loss

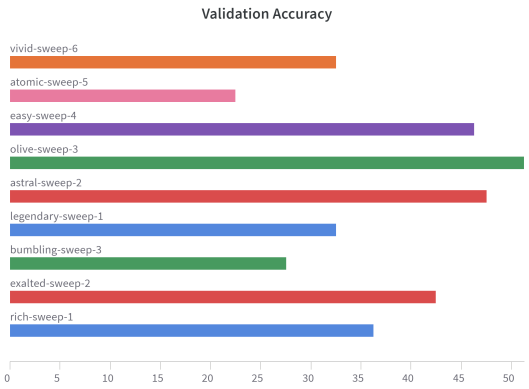

Figure 40: CNN-Transformer Model hyperparamter tuning validation accuracy

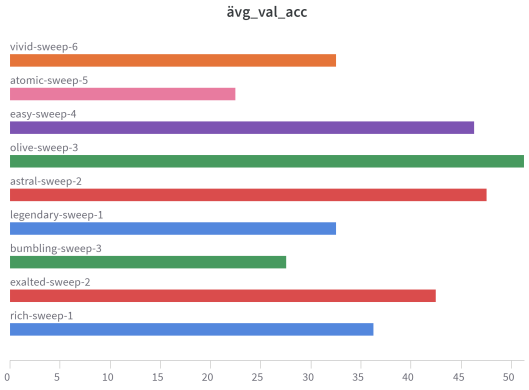

Figure 41: CNN-Transformer Model hyperparamter tuning validation accuracy

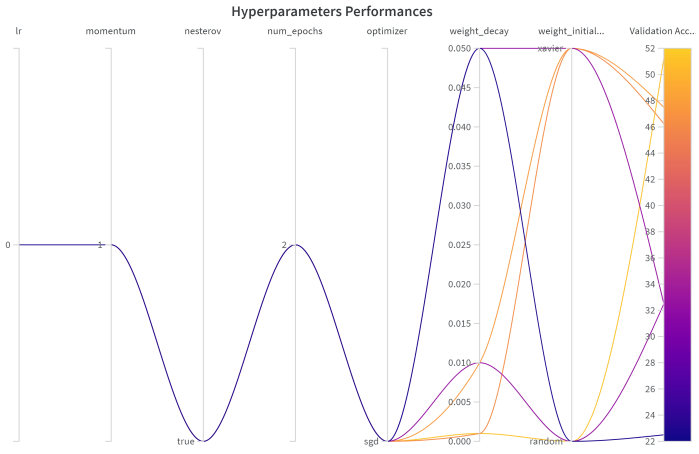

Figure 42: 1D-CNN Model hyperparamter tuning using Grid Seach

Hyperparameters	**Run Summary**				
lr: 0.001	Epoch	Train Acc.	Train Loss	Valid Acc.	Valid Loss
momentum: 0.9	20	92.08%	6.76745	53.75%	5.833
nesterov: True					
num_epochs: 20					
optimizer: sgd					
weight_decay: 0.001					
weight_init: random					

Table 9: Combined Hyperparameters and Run Summary for 1D-CNN .

Hyperparameters	**Run Summary**				
lr: 0.001	Epoch	Train Acc.	Train Loss	Valid Acc.	Valid Loss
momentum: 0.9	20	92.08%	7.48738	58.75%	5.78984
nesterov: True					
num_epochs: 20					
optimizer: sgd					
weight_decay: 0.01					
weight_init: xavier					

Table 10: Hyperparameters and Run Summary for 1D-CNN

Hyperparameters	**Run Summary**				
lr: 0.001	Epoch	Train Acc.	Train Loss	Valid Acc.	Valid Loss
momentum: 0.9	19	93.33%	7.50746	53.75%	6.0401
nesterov: True					
num_epochs: 20					
optimizer: sgd					
weight_decay: 0.05					
weight_init: random					

Table 11: Hyperparameters and Run Summary for the 1D-CNN

Hyperparameters	**Value**
Learning Rate (lr)	0.001
Momentum	0.9
Nesterov	True
Optimizer	SGD (Stochastic Gradient Descent)
Weight Decay	0.001
Weight Initialization	Xavier

Table 12: Best Hyperparameters for the 1D-CNN + Transformer Model.

Hyperparameters	**Run Summary**				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.01 weight_init: random	Epoch 19	Train Acc. 100.0%	Train Loss 0.11643	Valid Acc. 62.5%	Valid Loss 5.87866

Table 13: Hyperparameters and Run Summary for the given configuration. (Transformer-CNN)

Hyperparameters	**Run Summary**				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.01 weight_init: xavier	Epoch 19	Train Acc. 100.0%	Train Loss 0.12191	Valid Acc. 65.0%	Valid Loss 5.31805

Table 14: Hyperparameters and Run Summary for the given configuration.(Transformer-CNN)

Hyperparameters	**Run Summary**				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.001 weight_init: random	Epoch 19	Train Acc. 100.0%	Train Loss 0.13167	Valid Acc. 61.25%	Valid Loss 5.2306

Table 15: Hyperparameters and Run Summary for the given configuration.(Transformer-CNN)

Hyperparameters	**Run Summary**				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.001 weight_init: xavier	Epoch 19	Train Acc. 99.58%	Train Loss 0.26608	Valid Acc. 68.75%	Valid Loss 6.04176

Table 16: Hyperparameters and Run Summary for the given configuration.(Transformer-CNN)

Hyperparameters	**Run Summary**				
lr: 0.001 momentum: 0.9 nesterov: True num_epochs: 20 optimizer: sgd weight_decay: 0.05 weight_init: random	Epoch 19	Train Acc. 99.58%	Train Loss 0.57533	Valid Acc. 57.5%	Valid Loss 5.80589

Table 17: Hyperparameters and Run Summary for the given configuration.(Transformer-CNN)