Project Report Microneurography Multi-Task Spike-Sorting

People Involved: Bhanu Koppolu, Seyedehmotahare Mousavi, Kevin Bluhm, Alina Troglio, Ekaterina Kutafina, Mayra Elwes

Methods

Data

This study utilizes five microneurography datasets, namely A2, A3, A4, A12, and A21. Each one of the dataset spike waveform recordings has the following specifications:

The label has a different semantic for each dataset.

Dataset Specifications:

- A2 - 2 Classes
    - Total number of Samples: 536
        - Class 1 - Track 3 - 268 Samples
        - Class 2 - Track 4 - 268 Samples
- A3 - 2 Classes
    - Total number of Samples: 346
        - Class 1 - Track 1 - 173 Samples
        - Class 2 - Track 2 - 173 Samples
- A4 - 3 Classes
    - Total number of Samples: 1966
        - Class 1 - Track 1 - 647 Samples
        - Class 2 - Track 2 - 632 Samples
        - Class 3 - Track 4 - 687 Samples
- A12 - 6 Classes
    - Total number of Samples: 1932
        - Class 1 - Track 3 - 321 Samples
        - Class 2 - Track 7 - 324 Samples
        - Class 3 - Track 8 - 324 Samples
        - Class 4 - Track 11 - 323 Samples
        - Class 5 - Track 13 - 318 Samples
        - Class 6 - Track 15 - 322 Samples
- A21 - 2 Classes
    - Total number of Samples: 526
        - Class 1 - Track 3 - 218 Samples
        - Class 2 - Track 4 - 308 Samples

These five datasets, each containing 30-dimensional feature vectors comprising 30 voltage samples. However, each dataset has a balanced class distribution. A12, in particular, with its six classes, is the most complex of the lot.

Dataset Preprocessing:

The study uses multiple approaches to preprocessing before training the models.

1. The Classic FFN, LSTM, and CNN make use of the `make_five_splits` function, which splits each dataset into five equal splits.
2. The Feature-based MTL FFN, LSTM, and CNN make use of the `get_joined_train_test_folds` function, which takes the splits generated by the `make_five_splits` and uses the `split_index` for test dataset creation and the rest of the indexes through concatenation for the training dataset creation.
3. The Classic SVM and Autoencoder use a Stratified K-Fold approach to distribute into five folds.
4. The Classic SVM Optuna uses `stratify=y` in the train-test split from the sklearn Python library.

There are two datasets used throughout the experiments:

1. Raw Dataset: This is the original dataset.
2. Normalized Dataset: This is the dataset created through row-wise max normalized operation performed on the Raw Dataset.

The split ratios for all experiments are 80% Training and 20% Testing.

A table to summarize the Dataset Preprocessing for all the experiments conducted in order.

Experiment ID	Notebook	Method	Train-Test Split %
E1	Classic-FFN-PyTorch.ipynb	`make_five_splits` function with concat based on `split_index`	80% Train, 20% Test
E2	Classic-FFN-PyTorch LSTM.ipynb		
E3	Classic-FFN-PyTorch CNN.ipynb		
E4	Feature-Based-MTL.ipynb	`get_joined_train_test_folds` function based on `split_index` with one-hot encodings for tracks	
E5	Feature-Based-MTL CNN.ipynb		
E6	Feature-Based-MTL LSTM.ipynb		
E7	Classic-SVM-Optuna.ipynb	Stratified holdout split with Optuna optimization	

E8	Classic-SVM-Optuna.ipynb with CV	Stratified 5-Fold with Normalized Dataset	
E9	Autoencoder - SVM - Seperate.ipynb	Stratified train_test_split with Normalized Data	
E10	Autoencoder - SVM - Joint.ipynb	Stratified 3-Fold with Normalized Data	

Table 1. Data Preprocessing Summary

Single-Task Methods

There are a total of 9 Experiments, out of which 5 are Single-Task methods. Let's take a close look at the model architecture for each of the Single-Task methods.

Classic-FFN-PyTorch.ipynb

1. **Framework:** PyTorch
2. **Architecture:** Linear Layers (2 - 5 Hidden Layers, varied according to the search space) and Dropout layers.
3. **Hyperparameter Tuning:** Ray Tune + Optuna
4. **Number of Trials and Sweeps:** 32 Trials * 5 Datasets = 160
5. **Data Preprocessing:** Uses the Raw dataset and uses the `make_five_split` function and the concat method.
6. **Search Space:**
    a. Activation: relu, tanh, sigmoid
    b. Learning Rate: 1e-4 - 1e-2
    c. Batch Size: 32, 64, 128
    d. Hidden Layers: varied hidden layer with options
    e. Epochs: 10, 20, 30, 40, 50
    f. Dropout rate: 0.2 - 0.5

Classic-FFN-PyTorch LSTM.ipynb

1. **Framework:** PyTorch
2. **Architecture:** Similar to FFN with the addition of shared LSTM Layers.
3. **Hyperparameter Tuning:** Ray Tune + Optuna
4. **Number of Trials and Sweeps:** 32 Trials * 5 Datasets = 160
5. **Data Preprocessing:** Same as FFN
6. **Search Space:** Same as FFN, but with shared LSTM layers.
    a. LSTM Layers: [64, 128, 256]

Classic-FFN-PyTorch CNN.ipynb

1. **Framework:** PyTorch
2. **Architecture:** Similar to FFN with the addition of shared Convolution Layers and Pooling layers.
3. **Hyperparameter Tuning:** Ray Tune + Optuna
4. **Number of Trials and Sweeps:** 32 Trials * 5 Datasets = 160

5. **Data Preprocessing:** Same as FFN
6. **Search Space:** Same as FFN, but with shared CNN layers
    a. Convolution Channels: [16, 32, 64]
    b. Kernel Sizes: [3, 5, 7]
    c. Pool Size: [2, 3]

Classic-SVM-Optuna.ipynb

1. **Framework:** Sklearn
2. **Architecture:** SVM
3. **Hyperparameter Tuning:** Optuna
4. **Number of Trials and Sweeps:** A2 - 150, A3 - 150, A4 - 200, A12 - 250, A21 - 150 is the Number of Trials Optuna performed for each dataset.
5. **Data Preprocessing:** Stratified train-test split with Normalized Dataset.
6. **Search Space:**
    a. C: 1e-3 - 1e3
    b. Kernel: rbf, sigmoid, poly, linear
    c. Gamma: based on Kernel
        i. Auto
        ii. Scale
        iii. Float: 1e-4 - 10
    d. Class Weight - Special Handling for A12: None, Balanced

Classic-SVM-Optuna.ipynb with CV

1. **Framework:** Sklearn
2. **Architecture:** SVM
3. **Hyperparameter Tuning:** Optuna
4. **Number of Trials and Sweeps:** A2, A3, A4, A21 - 120 Optuna trials, A12 - 150 Optuna trials, 10 Repetitions * 5 Outer Folds
5. **Data Preprocessing:** Stratified 5-Fold with Normalized Dataset.
6. **Search Space:**
    a. C: 1e-3 - 1e3
    b. Kernel: rbf, sigmoid, poly, linear
    c. Gamme: based on Kernel
        i. Auto
        ii. Scale
        iii. Float: 1e-5 - 1e1

Multi-Task Methods

The remaining 4 are the Multi-Task methods. Let's take a close look at the model architecture for each of the Multi-Task methods.

Feature-Based-MTL.ipynb

1. **Framework:** PyTorch
2. **Objective:** Multi-task classification with shared representation learning
    a. Loss Function: Cross-entropy loss per task, averaged across tasks

    b. Training Strategy: Simultaneous training on all five datasets with task-specific batching
3. **Architecture:** Multi-task neural network with shared hidden layers and task-specific output heads
    a. Shared Component: 1-5 fully connected layers (10-50 nodes each)
    b. Task-Specific Components: Dataset-specific output branches (0-4 additional hidden layers per task)
4. **Hyperparameter Tuning:** Ray Tune + Optuna
5. **Number of Trials and Sweeps:** 32 Ray Trials * 6 Layer Configurations: 0 - 5 Shared Layers
6. **Data Preprocessing:** `get_joined_train_test_folds` function, which returns the Train and Test Dataset based on split_index, where the test is one-hot encoded tracks.
7. **Search Space:**
    a. Activation: relu, tanh, sigmoid
    b. Learning Rate: 1e-4 - 1e-2
    c. Batch Size: 32, 64, 128
    d. Shared Hidden Layers: varied hidden layer with options
    e. Task-specific layers: 0-4 layers per dataset
    f. Epochs: 10, 20, 30, 40, 50
    g. Dropout Rate: 0.2 - 0.5

Feature-Based-MTL CNN.ipynb

1. **Framework:** PyTorch
2. **Objective:** Multi-task classification with shared representation learning
    a. Loss Function: Cross-entropy loss per task, averaged across tasks
    b. Training Strategy: Simultaneous training on all five datasets with task-specific batching
3. **Architecture:** Multi-task neural network with shared hidden layers and task-specific output heads
    a. Shared Component:
        i. Shared Convolution Block: Conv1D and Flatten Layers
        ii. Shared Linear Block: 1-5 fully connected layers (10-50 nodes each)
    b. Task-Specific Components: Dataset-specific output branches (0-4 additional hidden layers per task)
4. **Hyperparameter Tuning:** Ray Tune + Optuna
5. **Number of Trials and Sweeps:** 32 Ray Trials * 6 Layer Configurations: 0 - 5 Shared Layers
6. **Data Preprocessing:** `get_joined_train_test_folds` function, which returns the Train and Test Dataset based on split_index, where the test is one-hot encoded tracks.
7. **Search Space:** Same as MTL FFN, but with additional shared CNN Layers:
    a. Conv1D layers: [32, 64]

Feature-Based-MTL LSTM.ipynb

1. **Framework:** PyTorch

2. **Objective:** Multi-task classification with shared representation learning
    a. Loss Function: Cross-entropy loss per task, averaged across tasks
    b. Training Strategy: Simultaneous training on all five datasets with task-specific batching
3. **Architecture:** Multi-task neural network with shared hidden layers and task-specific output heads
    a. Shared Component:
        i. Shared LSTM Block: LSTM layers
        ii. Shared Linear Block: 1-5 fully connected layers (10-50 nodes each)
    b. Task-Specific Components: Dataset-specific output branches (0-4 additional hidden layers per task)
4. **Hyperparameter Tuning:** Ray Tune + Optuna
5. **Number of Trials and Sweeps:** 32 Ray Trials * 6 Layer Configurations: 0 - 5 Shared Layers
6. **Data Preprocessing:** `get_joined_train_test_folds` function, which returns the Train and Test Dataset based on split_index, where the test is one-hot encoded tracks.
7. Search Space: Same as MTL FFN, but with additional shared LSTM Layers:
    a. LSTM layers: [32, 64]

Autoencoder - SVM - Seperate.ipynb

1. **Framework:** PyTorch and Sklearn
2. **Architecture:** Autoencoder - Hidden Layer + Bottleneck, SVM
3. **Hyperparameter Tuning:** Optuna
4. **Number of Trials and Sweeps:**
    a. Autoencoder: A2 - 50, A3 - 50, A4 - 75, A12 - 100, A21 - 50
    b. SVM: A2 - 150, A3 - 150, A4 - 200, A12 - 250, A21 - 150
5. **Data Preprocessing:** Stratified train-test split with Normalized Dataset.
6. **Search Space:**
    a. Autoencoder:
        i. Hidden Layer Dimension: 10 - 30
        ii. Bottleneck Dimension: 2 - 29
        iii. Learning Rate: 1e-4 - 1e-2
        iv. Batch Size: 32, 64, 128
        v. Dropout Rate: 0.0 - 0.3
        vi. Weight Decay: 1e-6 - 1e-3
        vii. Epochs: 30 - 80
    b. SVM: Same as in Classic SVM Optuna and Classic SVM Optuna with CV.

Autoencoder - SVM - Joint.ipynb

1. **Framework:** PyTorch and Sklearn
2. **Architecture:** Autoencoder - Hidden Layer + Bottleneck, SVM
3. **Hyperparameter Tuning:** Optuna
4. **Tuning Objective:** Weighted combination of Reconstruction Loss + Classification Error.
5. **Number of Trials and Sweeps:** Joint Optimization - A2, A3, A21 - 200 Optuna Trials, A4 - 250 Optuna Trials, A12 - 400 Optuna Trials

6. **Data Preprocessing:** Stratified 3-Fold with Normalized Dataset
7. **Search Space:**
    a. Autoencoder: Same as Autoencoder - SVM Seperate with additional:
        i. Autoencoder Weight: 0.15 - 0.35
    b. SVM: Same as in Classic SVM Optuna and Classic SVM Optuna with CV.

Results

The following tables summarize the performance across all five datasets (A2, A3, A4, A12, A21) in all experiments, using the accuracy, F1-Micro, F1-Macro, and MCC (Matthews Correlation Coefficient) metrics.

Classic FFN Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.892 ± 0.062	0.892 ± 0.062	0.892 ± 0.062	0.785 ± 0.124
A3	2	0.870 ± 0.016	0.870 ± 0.016	0.869 ± 0.016	0.746 ± 0.037
A4	3	0.703 ± 0.034	0.703 ± 0.034	0.698 ± 0.034	0.563 ± 0.045
A12	6	0.309 ± 0.011	0.309 ± 0.011	0.299 ± 0.006	0.173 ± 0.013
A21	2	0.705 ± 0.037	0.705 ± 0.037	0.673 ± 0.040	0.381 ± 0.072

Table 2. Classic FFN Results

Classic FFN LSTM Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.946 ± 0.023	0.946 ± 0.023	0.946 ± 0.023	0.893 ± 0.046
A3	2	0.919 ± 0.015	0.919 ± 0.015	0.919 ± 0.015	0.839 ± 0.030
A4	3	0.568 ± 0.068	0.568 ± 0.068	0.508 ± 0.092	0.395 ± 0.075
A12	6	0.238 ± 0.040	0.238 ± 0.040	0.162 ± 0.045	0.103 ± 0.062
A21	2	0.681 ± 0.081	0.681 ± 0.081	0.627 ± 0.108	0.340 ± 0.144

Table 3. Classic FFN LSTM Results

Classic FFN CNN Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.946 ± 0.019	0.946 ± 0.019	0.946 ± 0.019	0.893 ± 0.038
A3	2	0.913 ± 0.020	0.913 ± 0.020	0.913 ± 0.020	0.828 ± 0.041
A4	3	0.708 ± 0.036	0.708 ± 0.036	0.706 ± 0.036	0.565 ± 0.051
A12	6	0.316 ± 0.017	0.316 ± 0.017	0.308 ± 0.018	0.182 ± 0.021
A21	2	0.760 ± 0.026	0.760 ± 0.026	0.737 ± 0.023	0.496 ± 0.043

Table 4. Classic FFN CNN Results

Feature-Based MTL FFN Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.759 ± 0.044	0.759 ± 0.044	0.758 ± 0.043	0.523 ± 0.088
A3	2	0.734 ± 0.047	0.734 ± 0.047	0.732 ± 0.047	0.474 ± 0.093
A4	3	0.496 ± 0.025	0.496 ± 0.025	0.462 ± 0.014	0.265 ± 0.034
A12	6	0.216 ± 0.025	0.216 ± 0.025	0.197 ± 0.015	0.062 ± 0.032
A21	2	0.599 ± 0.071	0.599 ± 0.071	0.405 ± 0.051	0.064 ± 0.142

Table 5. Feature-Based MTL FFN Results

Feature-Based MTL CNN Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.946 ± 0.025	0.946 ± 0.025	0.946 ± 0.025	0.892 ± 0.049
A3	2	0.922 ± 0.020	0.922 ± 0.020	0.922 ± 0.020	0.845 ± 0.040
A4	3	0.668 ± 0.042	0.668 ± 0.042	0.663 ± 0.041	0.511 ± 0.055
A12	6	0.291 ± 0.026	0.291 ± 0.026	0.247 ± 0.025	0.157 ± 0.033
A21	2	0.717 ± 0.023	0.717 ± 0.023	0.700 ± 0.020	0.422 ± 0.043

Table 6. Feature-Based MTL CNN Results

Feature-Based MTL LSTM Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.871 ± 0.021	0.871 ± 0.021	0.871 ± 0.021	0.749 ± 0.039
A3	2	0.864 ± 0.034	0.864 ± 0.034	0.864 ± 0.034	0.731 ± 0.066
A4	3	0.478 ± 0.053	0.478 ± 0.053	0.400 ± 0.049	0.279 ± 0.057
A12	6	0.218 ± 0.023	0.218 ± 0.023	0.186 ± 0.029	0.066 ± 0.029
A21	2	0.688 ± 0.037	0.688 ± 0.037	0.635 ± 0.044	0.341 ± 0.053

Table 7. Feature-Based MTL LSTM Results

Classic SVM Optuna Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.838 ± 0.027	0.838 ± 0.027	0.837 ± 0.027	0.680 ± 0.057
A3	2	0.780 ± 0.049	0.780 ± 0.049	0.779 ± 0.050	0.566 ± 0.096
A4	3	0.680 ± 0.012	0.680 ± 0.012	0.680 ± 0.012	0.520 ± 0.019
A12	6	0.307 ± 0.023	0.307 ± 0.023	0.300 ± 0.023	0.169 ± 0.028
A21	2	0.648 ± 0.061	0.648 ± 0.061	0.638 ± 0.063	0.277 ± 0.127

Table 8. Classic SVM Optuna Results

Classic SVM Optuna with CV Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.847 ± 0.027	0.847 ± 0.027	0.847 ± 0.027	0.696 ± 0.055
A3	2	0.832 ± 0.033	0.832 ± 0.033	0.832 ± 0.033	0.671 ± 0.061
A4	3	0.663 ± 0.022	0.663 ± 0.022	0.661 ± 0.020	0.497 ± 0.034
A12	6	0.308 ± 0.032	0.308 ± 0.032	0.296 ± 0.032	0.170 ± 0.038
A21	2	0.705 ± 0.032	0.705 ± 0.032	0.665 ± 0.045	0.381 ± 0.075

Table 9. Classic SVM Optuna with CV Results

Autoencoder SVM Seperate Results

Dataset	Classes	Accuracy	F1-Micro	F1-Macro	MCC
A2	2	0.843 ± 0.014	0.843 ± 0.014	0.843 ± 0.014	0.689 ± 0.027
A3	2	0.835 ± 0.045	0.835 ± 0.045	0.835 ± 0.046	0.678 ± 0.088
A4	3	0.680 ± 0.009	0.680 ± 0.009	0.680 ± 0.009	0.520 ± 0.014
A12	6	0.297 ± 0.014	0.297 ± 0.014	0.291 ± 0.014	0.157 ± 0.017
A21	2	0.700 ± 0.056	0.700 ± 0.056	0.674 ± 0.061	0.367 ± 0.122

Table 10. Autoencoder SVM Seperate Results

Autoencoder SVM Joint Results

Dataset	Classes	Accuracy	Std Dev (CV)
A2	2	0.858	0.043
A3	2	0.841	0.037
A4	3	0.691	0.025
A12	6	0.306	0.010
A21	2	0.694	0.031

Table 11. Autoencoder SVM Joint Results

Discussion

In this discussion section, let’s take a close look at the patterns in performance and comparison across the methods or experiments conducted.

From the above Tables 2 - 11. We see apparent performance differences across methods:

1. Best Performance
   a. Classic CNN PyTorch achieves the highest accuracy across most of the datasets (A2 - 94.6%, A3 - 91.3%, A21 - 76.0%).
   b. Classic LSTM PyTorch is not very behind and achieves the highest accuracy in datasets (A2 - 94.6%, A3 - 91.9%).
   c. Feature-Based MTL CNN also achieved strong performance with Multi-Task Learning (A2 - 94.6%, A3 - 92.2%).
2. Classic Methods Performance

    a. Classic CNN achieves the best performance across the three methods (FFN, CNN, and LSTM).
    b. LSTM significantly underperforms compared to FFN in the A4 and A12 datasets.
3. Feature-Based MTL Methods Performance
    a. Here we have two methods, FFN and CNN, with CNN performing the best and FFN significantly underperforming across all datasets.
4. SVM-based Methods Performance
    a. Here, we have two methods: Classic SVM and Classic SVM with CV, both of which are hyperparameter-optimized with Optuna.
    b. Both perform very similarly across all datasets, with a deviation of about ± 5%.
5. Autoencoder-SVM Methods Performance
    a. Here, we have two methods: Autoencoder-SVM Separate and Autoencoder-SVM Joint methods.
    b. There isn't any significant performance difference between these two methods.

Single-Task vs Multi-Task Methods

Out of the nine experiments conducted, five are single-task and four are multi-task. The best performing architectures in both the single-task and multi-task are with CNN. A slight improvement from a single-task Classic CNN to a multi-task Feature-Based MTL CNN is observed only for the A3 dataset; the A2 dataset has the same results in both cases. The remaining datasets showed a slight decrease in performance with the multi-task approach compared to the classic method. The FFN approach experienced a significant decline in performance when moving from the single-task to the multi-task approach. The MTL FFN had the poorest results in all the experiments conducted.

Overall, the best-performing experiment out of the nine conducted is the Classic FFN CNN, which has the highest accuracy scores, with one exception being A3, where the MTL CNN achieved a slight improvement in accuracy. The shared CNN layers are more effective for capturing temporal patterns than just using shared linear layers.

Performance Changes with Transition to Multi-Task Learning from Single Task

The Classic FFN is a single-task model, where we switched to a Feature-based MTL FFN, but we observed a significant performance decline. As a result, the FFN did not show notable improvement; instead, there was a substantial drop in performance with the transition. The Classic SVM is a single-task model, whereas we switched to Autoencoder-SVM, which is a multi-task learning model. We observed no performance changes. The transition to the multi-task model showed no gains or drops in performance.

There are two other experiments, which are Classic CNN and Classic LSTM, which are also experimented with to transition to multi-task learning. The following is observed:

1. For the transition from single task to multi-task for the CNN, the datasets A2 and A3 had no drops in performance, but the datasets A4, A12, and A21 experienced a slight drop in performance, as a difference is observed in Tables 4 and 6.

2. For the same transition for the LSTM, all the datasets' performance drastically reduced when compared to the Classic LSTM performance.

Differences between Datasets

The datasets A2 and A3, which are binary datasets, are the ones that perform the best across all the experiments. Dataset A4, being a three-class dataset, performs weaker despite having more samples per class than the other datasets. Dataset A12, which has six classes, performs the worst across all the experiments; the MCC was close to zero, suggesting that the model does not truly understand the patterns and is no better than random guessing. The most surprising result emerged with the A21 dataset; despite it being a binary dataset, the higher standard deviation indicates a problem with the dataset and the lack of significant pattern differences between the classes, leading to lower performance compared to datasets A2 and A3, which are the best-performing binary datasets.

Let's explore the differences in performance across various datasets:

1. A2 - 2 Classes
    a. Performance across all the experiments is consistent and above 80% except for the MTL FFN experiment.
    b. The CNN approaches provided the best performance, at 94.6%, which was consistent across the MTL CNN and Classic CNN experiments.
    c. The metrics have a lower standard deviation, which means they can provide stable and reliable predictions.
2. A3 - 2 Classes
    a. Performance is similar to the A2 but slightly lower.
    b. The CNN approaches provided the best performance, at 91.3% for the Classic CNN and 92.2% for the MTL CNN.
    c. The slight reduction in performance could be due to fewer samples per class compared to the A2 dataset.
3. A4 - 3 Classes
    a. We notice significant performance drops when compared to the binary datasets (A2 and A3).
    b. The Classic CNN approach provides the best performance at 70.8%.
    c. Despite having significantly more samples per class compared to other datasets, it still underperforms in correctly classifying.
4. A12 - 6 Classes
    a. A12 is the most challenging dataset out of all five.
    b. The Classic CNN approach provides the best performance at 31.6%, but it still has poor performance.
    c. The number of samples per class is higher compared to A2 and A3, but the model performance remains poor.
    d. The Classic LSTM performed poorly, achieving just 23.8%.
    e. The MCC scores for all models are very close to 0, indicating the model is no better than random guessing.
5. A21 - 2 Classes

a. The A21 dataset has a similar number of samples per class as the A2 dataset and fewer samples per class than the A3 dataset; however, the model performs lower than the other binary datasets (A2, A3).
b. The Classic CNN approach achieves the best performance at 76.0%, significantly outperforming the A2 and A3 datasets, which are also binary classification datasets.
c. The metrics reveal a higher standard deviation across all experiments, suggesting an inherent issue with the data itself.
d. Despite experimentation with CNN and LSTM, the patterns in the A21 dataset are not thoroughly adapted to the model, despite the dataset itself being a Binary dataset.

Standard Deviation Methodology

The 5-Fold or five splits approaches were used consistently across all the experiments. Subsequently, the `np.std` function was applied to the five obtained results. This procedure enables the calculation of the standard deviation for the metrics of accuracy, F1-micro, F1-macro, and MCC scores. The standard deviation serves as an indicator of the stability and reliability of the model’s predictions.