EMG Signal for Gesture Recognition

A.V. Kuznetsov

Introduction

Electromyographic (EMG) signals are biomedical data that record the electrical activity of muscles. They play a key role in medical research and applications such as the diagnosis and monitoring of diseases, as well as in the management of various technologies, including prostheses and robotic systems. In this project, methods for classifying EMG signals for recognizing hand gestures are being studied. Various approaches are considered, including traditional machine learning methods and more modern neural networks. Particular attention is paid to the choice of the model architecture, since it determines how accurately and effectively the model will be able to classify signals. The project selected an architecture based on convolutional and recurrent neural networks, which allows to improve the performance of the model by extracting spatiotemporal features from complex biomedical data.

1. Analysis of Existing Solutions

During the research, three key solutions were considered:

1. **Classification of EMG-signals Using AutoGluon**:
   - **Pros**: This approach uses AutoGluon, an AutoML tool that automates the process of selecting models and hyperparameters. AutoGluon can significantly speed up the model selection process, which is especially useful at the prototyping stage when it is necessary to quickly find a working solution.
   - **Cons**: Despite the speed and convenience, AutoML models often have limitations in flexibility and depth of customization. For high-precision tasks such as classification of EMG signals, more detailed tuning and optimization may be required than an automated approach can offer.
2. **Gesture Classification EMG**:
   - **Pros**: This approach specializes in the task of gesture classification. Presumably, it includes advanced classification techniques such as neural networks, which can be more effective when working with EMG signals due to their ability to capture complex nonlinear dependencies in data.
   - **Cons**: Although neural networks are a powerful tool for solving such problems, their performance largely depends on the quality of the data, the architecture of the model, and the learning process. Additional optimization of the algorithms may be required.
3. **Denoisenet**:
   - **Pros**: The main focus of this approach is on reducing data noise, which is extremely important for working with EMG signals. Noise reduction can significantly improve the accuracy of the classification model, as it improves the quality of the input data.
   - **Cons**: Despite the importance of noise elimination, in order to achieve high classification accuracy, it is also important to ensure an optimal network architecture and advanced learning algorithms.

Based on the analysis of several existing solutions, the Gesture Classification EMG algorithm was chosen as the most promising solution for further work. This approach includes using several classical machine learning algorithms, such as KNN, SVM, Random Forest, and Gradient Boosting, which allows not only the evaluation of the performance of each of them flexibly and the choice of the optimal classifier for working with EMG signals. In addition, the code efficiently processes data, extracts key features, and provides a convenient platform for comparative analysis of models based on accuracy and training time metrics.

However, despite the effectiveness of classical methods, EMG signals contain complex nonlinear dependencies, noise, and artifacts in the raw data (in this project, the data has already been cleared), which limits the potential of these models. Classical machine learning algorithms tend to be less able to handle such complexities without significant refinement. In this regard, it was decided to rewrite the architecture using neural networks to achieve greater flexibility and improve the model's accuracy. Neural networks have significant potential when working with EMG signals, as they can effectively model complex dependencies and consider non-linearities, which is especially important for biomedical data. Introducing convolutional and recurrent neural networks (CNN and LSTM) will improve the extraction of spatiotemporal features and improve the model's overall performance.

1.1 Analysis of the Gesture Classification EMG Code

When analyzing the source code and applying various classical machine learning models to classify EMG signal data, it was noticed that, despite reasonably good results in accuracy and F1 metrics, the models do not always show stable and high results on different subsamples of data. This is because classical algorithms such as SVM, KNN, and Decision Trees work well on data where features have clearly defined boundaries. However, EMG signals are time sequences that contain hidden dependencies and features that are implicitly represented in the source data.

Classical methods such as logistic regression and decision trees have demonstrated promising results when using primary data preprocessing approaches such as feature standardization and extraction of some statistical characteristics (e.g., RMS, zero crossing, and other functions). However, despite these efforts to improve classification quality through data preprocessing, the use of more complex models, such as random forests and gradient boosting, still did not significantly improve the accuracy and F1 metrics. Moreover, the models showed a deterioration in performance when trying to increase their complexity, which indicates difficulties in processing temporary data using classical methods.

After analyzing the possibilities for further improvement of classical models through optimization of hyperparameters and architectures, it became evident that the capabilities of these models for EMG classification are limited. In this regard, revising the architecture and teaching methods became necessary. The decision to rewrite the model based on neural networks in PyTorch was the most logical step. Neural networks, especially architectures such as CNN-LSTM, have the advantage that they can capture time dependencies in sequential data, as well as identify hidden patterns and features of signals. The main area of improvement was the change of architecture from classical methods to a more modern and adaptive architecture of neural networks. CNN efficiently process spatial dependencies in data. LSTM copes well with temporal dependencies, which is especially important for data received from EMG signals. At the same time, the adjustment of hyperparameters, such as the number of layers, the size of the hidden state, and the use of dropout, made it possible to achieve a significant increase in the accuracy of the model.

The use of CNN-LSTM architecture also made it possible to simplify data preprocessing. Unlike classical methods, which require extracting many manual features, neural networks can independently find necessary dependencies and structures in data, making them more effective when working with time sequences such as EMG signals. The final solution improved performance on the main metrics and became more flexible and adaptable in more complex scenarios.

As a result, using a neural network approach instead of classical methods has made it possible to achieve significant improvements in results, which is confirmed by an increase in the accuracy of classification and F1 metrics. This once again underlines that neural networks are a preferable tool for

tasks related to sequences and time series compared to traditional machine learning methods.

Why Neural Networks with PyTorch?

1. **Automatic Feature Extraction**: Unlike manual feature engineering, neural networks automatically extract and optimize features, simplifying data manipulation and improving results, especially on complex biomedical data such as EMG signals.
2. **Flexibility in working with time series**: CNN and LSTM better handle temporal and spatial dependencies, which is especially important when analyzing signals. You can easily combine these architectures in PyTorch to get more accurate results.
3. **Scalability and performance**: PyTorch can work with large amounts of data and effectively train models on the GPU, which significantly speeds up the learning process compared to classical algorithms. Neural networks, especially deep ones, can also work with more complex and large amounts of data, extracting hidden patterns that are difficult to capture using classical methods.
4. **Processing noisy data**: EMG signals are subject to noise, and neural networks (e.g., through architectures with autoencoders or DenoiseNet) can effectively filter noise, which improves the quality of input data for classification.
5. **Adaptability and improvement through complex architectures**: Neural networks can be easily improved by adding new layers, configuring hyperparameters, and using various regularizers (such as Dropout), making them flexible and adaptive. With such capabilities, PyTorch provides more tools for creating customized and robust solutions.

2. CNN-LSTM Hybrid Architecture for Classification of EMG Signals

This project implements an approach to classifying EMG signals using a hybrid architecture based on CNN and LSTM. This approach was chosen because it is necessary to effectively model both spatial and temporal dependencies in EMG data, which contain complex nonlinearities and noise.

2.1 Model Architecture: CNN + LSTM

The architecture used in this project combines convolutional layers to extract spatial features and recurrent layers to account for temporal dependencies in the data. CNNs are used to extract significant features from multi-channel signals automatically. In EMG data, channels represent measurements from different electrodes, and convolutional filters can identify local spatial dependencies between signals, improving their processing. LSTMs allow you to model important time dependencies for sequential data, such as EMG signals. LSTMs effectively cope with tasks where it is necessary to consider previous states, which is especially important for time-dependent signals.

2.2 Data Preprocessing

Using `StandardScaler` to normalize the data is critically important, as it helps improve neural network convergence by bringing the features to a standard normal distribution. Normalization ensures that all features are on the same scale, allowing for faster learning and preventing situations where some parameters dominate others due to scale differences. This is particularly crucial for models that use gradient optimization methods like Adam, as normalized data leads to smoother and faster convergence.

In this experiment, the data was already prepared for model building and did not require additional cleaning or preprocessing. It was normally distributed, which allowed us to focus on configuring the model

and improving its architecture. The data was divided into training, validation, and test sets using the `train_test_split` function from the `scikit-learn` library. This ensured that the data was randomly selected for each set, preventing overfitting to any single dataset.

2.3 Model Architecture

The code uses an architecture that includes two convolutional layers and LSTM layers, which allow efficient processing of sequential data from EMG signals.

The LSTM component in the model was chosen in part to address the vanishing gradient problem, making it more effective when working with long temporal sequences compared to standard RNNs. In this architecture, three LSTM layers are used, selected based on experiments to improve the model's ability to capture temporal dependencies in EMG data. This choice of layers helps balance model complexity and classification quality. It is also worth noting that the last hidden LSTM vector (`out[:, -1, :]`) is used for classification, which allows the model to account for important temporal patterns and improves accuracy on data with temporal dependencies.

The first convolutional layer is implemented using the `nn.Conv1d(in_channels=1, out_channels=64, kernel_size=3, padding=1)` function. This layer has 64 filters and a core size of 3, corresponding to the model's description. ReLU activation adds non-linearity, and the MaxPooling operation (with a core size of 2) reduces the data size, preserving the most important features. The second convolutional layer has 128 filters and uses the same core parameters (`nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, padding=1)`), which allows deeper feature processing. The MaxPooling operation is also applied to this layer to reduce the data's dimensionality further.

LSTM layers process the temporal data structure, allowing the model to account for the relationship between time steps in successive signals. In the code, LSTM has three layers, corresponding to the parameter `num_layers=3`, which coincides with the architecture description. The size of the hidden states is set to 128, which is set by the `hidden_size=128` parameter, and this choice is confirmed in the model description. LSTM returns the last hidden state, which is also correctly described and implemented in the code by selecting the previous output hidden vector `out[:, -1, :]`.

Fully connected layers follow the LSTM layers. The first fully connected layer contains 128 neurons (`nn.Linear(hidden_size, 128)`) and uses ReLU activation. The second fully connected layer is the output layer and has the number of neurons corresponding to the number of classes in the classification problem, represented in the code as `nn.Linear(128, output_size)`. Applying a Dropout with a probability of 0.4 (`nn.Dropout(dropout_rate=0.4)`) prevents overfitting, especially when working with small data.

Thus, this CNN architecture uses convolutional layers with cores of size 3 and several filters 64 and 128. These parameters were chosen for processing complex and noisy EMG signals since small cores make it possible to identify local features in the signals effectively, and increasing the number of filters with each layer improves the ability of the model to extract deeper spatial features. The first convolutional layer (with 64 filters) highlights basic patterns, such as frequency components of signals. The second (with 128 filters) deals with a more detailed analysis and highlights key features for further classification using LSTM.

2.4 Hyperparameters and Model Training

The following hyperparameters were used in the model training code, corresponding to the described settings: the size of the hidden states in the LSTM is 128, the number of LSTM layers is 3, as specified by the parameter `num_layers=3`, and the Dropout probability is 0.4. The learning rate is set to 0.0005 (`learning_rate=0.0005`), which allows the model to converge smoothly during training. During training, the model was optimized using the Adam algorithm (`optim.Adam`), which, due to its adaptive nature, effectively updates weights, especially when working with tasks that require large networks and sequential data. The loss function `nn.CrossEntropyLoss` was used to solve the multi-class classification problem.

In this case, the number of epochs of 20 was chosen based on observations of how the quality metrics of the model change. During the training, the graphs of the loss and accuracy functions show that the error in training gradually decreases, and the accuracy in validation stabilizes, especially towards the last epochs. The error did not show significant fluctuations, and the model was not prone to overfitting, as can be seen from the steady increase in accuracy on the validation data. The early stop was not applied. However, it could be added for more precise control over the learning process.

The learning rate of 0.0005 was selected manually based on previous experience and experiments with EMG signal classification tasks. It was not optimized by hyperparameter search methods but showed a good balance between convergence and model quality. Figure 2 show a consistent decrease in the loss function and an increase in accuracy on validation data, which indicates that there are no signs of over-training or under-training. The model was steadily trained and improved its predictions without sharp deviations.

A batch size of 64 was chosen based on the balance between the model's convergence speed and its quality, evaluated through metrics such as accuracy and F1-Score. In the context of working with EMG signals, a batch size of 64 is optimal for maintaining a stable training process and providing sufficient data for gradient updates at each iteration.

Smaller batch sizes could speed up training due to more frequent weight updates; however, this could also lead to greater variability in gradients, affecting the stability of the model's convergence. On the other hand, larger batch sizes could result in slower training, as weight updates would occur less frequently, and each iteration would take longer due to the increased amount of data being processed at once. In this code, the choice of a batch size of 64 was driven by the following factors:

1. **Training speed**: A batch size of this magnitude allows for sufficiently fast iterations, enabling the model to train in a reasonable amount of time without significant delays.
2. **Convergence quality**: Results for accuracy and F1-score demonstrated that a batch size of 64 supports smooth convergence, reduces loss function, and gradually improves validation metrics. This is confirmed by the graphs, which show how the model steadily improves its predictions with each epoch.
3. **Resource efficiency**: A batch size of 64 strikes a compromise between GPU resource utilization and data processing time. Larger batch sizes could require more memory, potentially leading to inefficiencies in resource usage.

Experiments with other batch sizes (e.g., 32, 128) showed that a batch size of 64 offers the optimal balance between speed and accuracy, allowing the model to converge faster and achieve higher accuracy and F1-Score on the validation set.

The model was trained for 20 epochs. During each epoch, metrics such as loss, accuracy, F1-score, training, and testing time were tracked. Validation was performed on the validation set to monitor potential overfitting and adjust the training process accordingly. The model was saved when the highest accuracy on the validation set was achieved, allowing the selection of the most optimal network version for subsequent testing stages.

3. Results of solving the classification problem

3.1 Model Analysis Based on Classical ML Algorithms

The first model, implemented through classical machine learning algorithms, uses several popular classifiers, such as KNN, SVM, DecisionTreeClassifier, and RandomForestClassifier. This model works with the extracted features of EMG signals, which are the basis for its classification. The code uses simple machine learning methods, such as `min`, `max`, `rms`, and `zero_crossing`, to efficiently extract signal characteristics.

The main advantages of the model are as follows: It is easy to implement and interpret. Algorithms such as KNN and SVM are well suited for classification tasks, especially when fast implementation is required. The time for calculating features and learning is much shorter than that of neural networks. This makes it suitable for tasks where computational efficiency is important. The main disadvantages of this model are the following: Classical algorithms can be limited when working with high-dimensional and complex data, as is often the case with biomedical signals. Such models cannot always capture all the nonlinear dependencies. Significant manual feature adjustment is required, which complicates the expansion of the model to new data or tasks. The average accuracy for classifiers ranged from 76% to 90%, where KNN and SVM showed the highest performance. However, despite the high accuracy results, the training and testing time for models such as Random Forest and Gradient Boosting was higher. The figure 1 shows the following results:

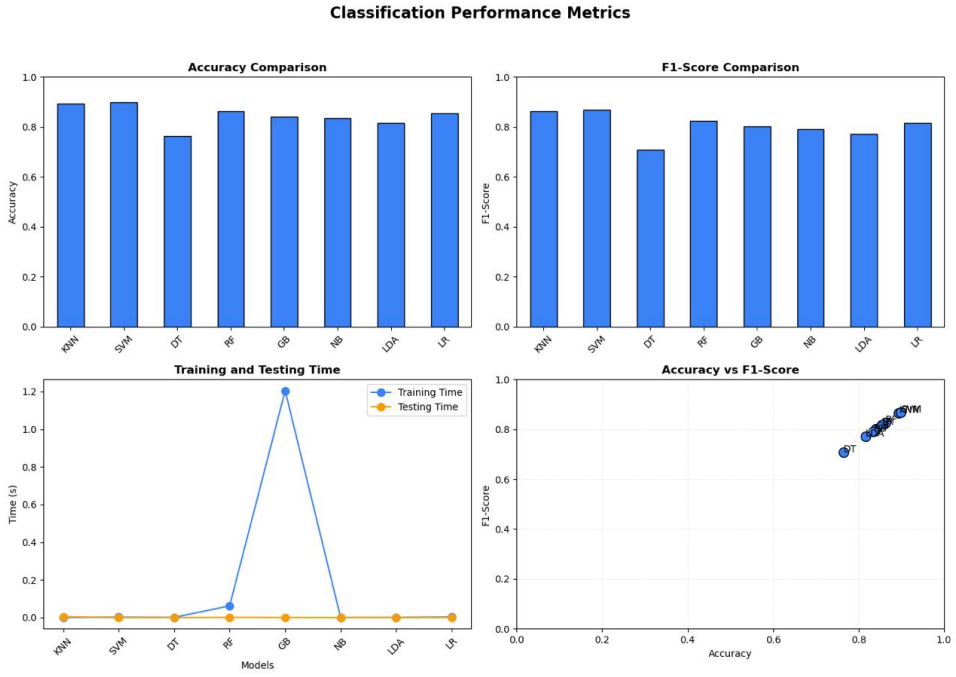

Figure 1: Classification Performance Metrics on Classical ML Algorithms.

1. **Accuracy Comparison**: The first panel shows the accuracy of various machine learning models. SVM and KNN demonstrate the highest accuracy rates of about 89-90%, while Decision Tree has a significantly lower accuracy of about 76%. This indicates that SVM and KNN are better at classifying EMG signals.

2. **F1-Score Comparison**: The second panel shows the F1-measure for all models. Again, SVM and KNN show the best results, indicating balanced performance in accuracy and completeness. Decision Tree and Random Forest are slightly inferior in this indicator.

3. **Training and Testing Time**: The third panel shows significant training time for Random Forest and Gradient Boosting compared to other models. However, the testing time for all models remains low and stable, which indicates their rapid predictive ability.

4. **Comparison of Accuracy and F1-measure**: The fourth panel demonstrates the dependence of accuracy on the F1-measure for all models. It can be seen that models with high accuracy also have high F1-measure values, which confirms their ability to cope well with the classification task.

3.2 Neural Network Model Analysis (CNN-LSTM)

During the development of the model for EMG signal classification, a hybrid CNN-LSTM architecture was used, which includes convolutional neural networks (CNN) for extracting spatial features and LSTM

for modeling temporal dependencies. This architecture was chosen due to its ability to simultaneously process both spatial and temporal dependencies in the data, which is essential when working with sequential signals like EMG.

The model architecture consists of two convolutional layers. The first layer is implemented using the function `nn.Conv1d(in_channels=1, out_channels=64, kernel_size=3, padding=1)`, which uses 64 filters and a kernel size of 3 to extract spatial features from the input data. This layer is supplemented by the ReLU activation to introduce non-linearity, as well as the MaxPooling operation, which reduces the dimensionality of the data and highlights the most important features. The second convolutional layer has 128 filters and uses similar kernel parameters `(nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, padding=1))`, allowing the model to further deepen feature processing. MaxPooling is applied after each convolutional layer to reduce computational complexity and help focus on the key features. After processing the data through the convolutional layers, LSTM layers are used to handle the temporal structure of the data. In the code, the LSTM is represented by a three-layer structure `(num_layers=3)` with the hidden state size set to 128 `(hidden_size=128)`. The LSTM layer in the code returns the last hidden state, which is used for further classification. This corresponds to the architectural description where it was mentioned that using three LSTM layers improves the model's ability to capture temporal dependencies without significantly increasing computational complexity.

The LSTM output is fed into two fully connected layers. The first layer contains 128 neurons `(nn.Linear(hidden_size, 128))`, and the second layer is the output layer with the number of neurons corresponding to the number of classes in the classification task `(nn.Linear(128, output_size))`. Dropout with a probability of 0.4 `(nn.Dropout(dropout_rate=0.4))` was used to prevent overfitting, which is especially important when training on small datasets. Regarding hyperparameters and the training process, the hidden layer size of the LSTM was set to 128, and the number of layers was set to three. Dropout with a probability of 0.4 was used to prevent overfitting, and the learning rate was fixed at 0.0005. The model was optimized using the Adam optimizer, which, due to its adaptiveness, effectively adjusts the model's weights during training. The CrossEntropyLoss function was used to evaluate the model's performance, which was suitable for solving the multi-class classification task.

The model was trained for 20 epochs, and each epoch was accompanied by evaluation on the validation set. The logging of metrics included tracking the loss on the training and validation sets, training time, accuracy, and F1-score. Throughout the experiments, gradual improvement in the model's performance was observed as the number of epochs increased, indicating the correctness of the architecture and the effectiveness of the data processing methods used. The results are presented in Figure 2.

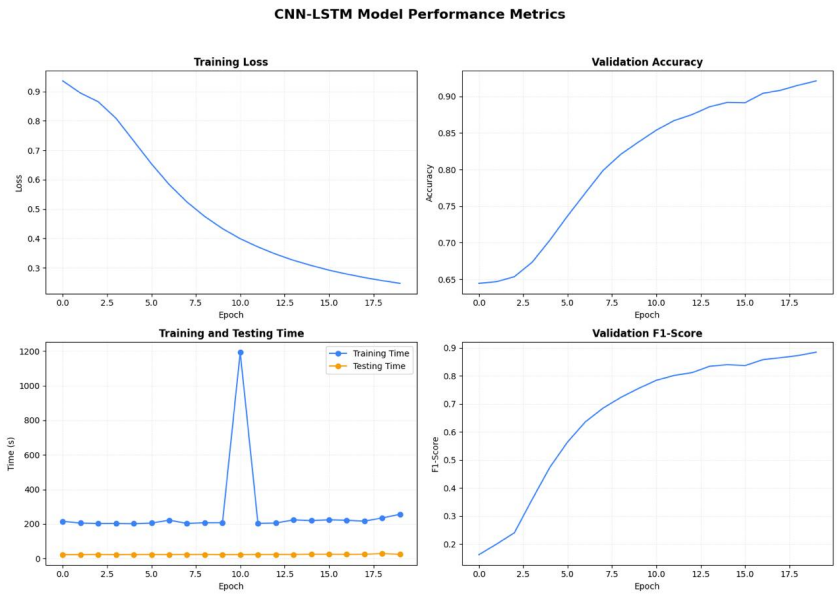

Figure 2: Classification Performance Metrics on Neural Network Algorithm.

The first graph showing the dynamics of the loss function in the training sample demonstrates a steady decrease in the error value as the number of epochs increases. At the initial stages of training (the first and second epochs), the loss function is about 0.9, but by the end of training, it decreases to values below 0.3. This indicates that the model is gradually learning and adapting to the data, improving its predictions. A smooth decline without significant jumps indicates a stable learning process without signs of retraining or problems with divergence. The high quality of the results is presented below

1. **Training Loss**: The graph shows a steady decrease in the loss function with an increase in epochs, indicating good model training. This means that the model gradually improves its predictions on the training data.
2. **Validation Accuracy**: It can be seen that with each epoch, the accuracy on the validation data gradually increases and stabilizes, reaching 92% in recent epochs.
3. **Training and Testing Time**: The learning time remains stable in most epochs, except for one anomaly around the 10th, while the testing time remains low and stable. The indicators of learning and validation in different epochs can also be seen in Table 1 (Appendix).
4. **Validation F1-Score**: The F1 measure based on validation data also shows growth, which indicates balanced accuracy and completeness of the model.

In addition to evaluating the model on the entire test dataset, an additional check was performed using a confusion matrix on a small subset of the data.

1. **Class 0** (the most frequent class) shows the highest number of correct predictions: 15,426 records were classified correctly. However, there is some confusion with neighboring classes (for example, classes 1 and 4), where the model misclassified 134 and 163 cases, respectively. This indicates that while the model performs well in recognizing class 0, it confuses it with other classes, which could be due to the fact that the features of these classes have similar temporal or spatial characteristics.
2. **Classes 2 and 3** show good results: the majority of the records for these classes were classified correctly (1255 and 1295 records, respectively), with minimal confusion with other classes. This suggests that the model has learned to recognize the characteristic features of these gestures well.
3. **Class 1** shows a higher level of confusion compared to other classes. Out of 1436 records belonging to class 1, the model correctly classified 981 records, but 451 records were mistakenly assigned to class 0. This may indicate that the signals of classes 0 and 1 share similar characteristics, making it difficult to distinguish between them.
4. **Classes 5 and 6** also show good results: the model correctly classified 1272 and 1282 records, respectively. However, for class 6, the model often confuses it with class 0 (132 misclassifications), which may indicate the presence of similar patterns between these classes.
5. **Class 7** has the fewest records (only 82), which makes it difficult for the model to learn from this class. Although the model correctly classified 68 records, 9 records were mistakenly assigned to class 0. This could be due to the model being undertrained on the small amount of data for this class.

The model shows high accuracy in classifying most classes, especially those with a large amount of data (e.g., class 0). The most frequent errors occur between classes with similar characteristics, especially between classes 0 and 1, indicating the need for further work to improve the distinguishability of these classes, possibly through additional data processing or model architecture modification. The level of accuracy can be improved by increasing the amount of data for less-represented classes, such as class 7, and optimizing hyperparameters to reduce confusion between similar classes.

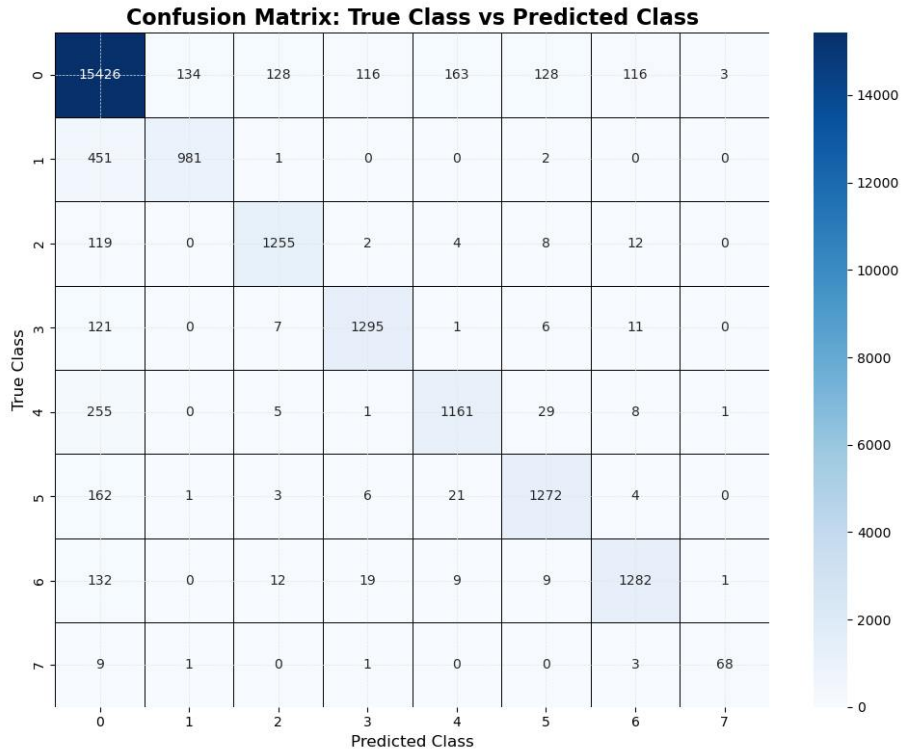

Figure 3: Confusion Matrix showcases the performance of the classification model, highlighting the distribution of true and predicted classes across multiple categories.

3.3 Comparison of results

The neural network has demonstrated excellent results compared to classical machine learning algorithms, achieving 92% accuracy and an F1 measure of 88% on test data. This is due to the fact that CNN-LSTM can model complex dependencies present in EMG signal data, which makes it more suitable for this task.

1. **Accuracy**: The neural network (CNN-LSTM) demonstrated better results in terms of accuracy (92%) and F1 measures (88%) compared to classical algorithms, which had an average accuracy of less than 90%.
2. **Training time**: Although the training time of the neural network was significantly higher, it was justified by improved accuracy and the generalizing ability of the model. Classical algorithms learned faster, but could not achieve the same depth of data analysis and interpretation.
3. **Using features**: Classical models require manual feature extraction, which may limit their ability to adapt to new tasks or data. Neural networks, on the contrary, can automatically extract complex features, which makes them more flexible for use in a wide range of tasks.
4. **Implementation complexity**: Neural networks are more difficult to implement and require more data and time for training. However, in the case of high-dimensional and complex data, as in our project with EMG signals, they give significantly better results.

Conclusion

The CNN-LSTM-based model has performed better than classical machine learning algorithms. Despite the more complex architecture and increased training time, this model copes better with classifying EMG signals, considering the data's complex temporal and spatial dependencies. The choice of neural

networks for such tasks is justified, especially if the goal is to achieve high accuracy and reliability of predictions.

In conclusion, while the CNN-LSTM architecture has shown superior results compared to classical methods, areas remain for improvement. One avenue for further development is experimenting with different neural network architectures, such as trying variations of recurrent layers (e.g., GRU) or incorporating attention mechanisms to better capture long-term dependencies in the data. Additionally, increasing the volume and diversity of the training data could further improve the model's generalizability, especially for less-represented classes, as observed in the confusion matrix.

Moreover, enhancing the preprocessing of EMG signals, such as applying more advanced feature extraction techniques or filtering methods, may help reduce noise and improve the clarity of signal patterns. Future work could also explore more in-depth hyperparameter optimization, such as fine-tuning the number of layers, the size of hidden states, or the learning rate, to further enhance the model's performance.

Files

1. **Report**: A comprehensive report detailing the process of gesture recognition, including model evaluation and results.
2. **Code for classical ML model**: The corrected code for a classical machine learning model from Kaggle used for hand gesture recognition.
3. **Code for neural network model**: A code implementation using a neural network on PyTorch for hand gesture recognition.
4. **Data file**: The EMG dataset used for training and testing both the classical ML and neural network models.

Appendix

Table 1: Training and Validation Metrics Across Epochs

Epoch	Loss	Val Loss	Val Accuracy	Val F1
1	0.9357	0.8977	0.6444	0.1618
2	0.8945	0.8798	0.6467	0.1998
3	0.8648	0.8388	0.6535	0.2400
4	0.8087	0.7703	0.6733	0.3590
5	0.7313	0.6940	0.7035	0.4738
6	0.6536	0.6257	0.7366	0.5643
7	0.5831	0.5675	0.7679	0.6360
8	0.5239	0.5099	0.7988	0.6851
9	0.4746	0.4669	0.8209	0.7232
10	0.4334	0.4389	0.8377	0.7554
11	0.3990	0.4071	0.8538	0.7842
12	0.3710	0.3827	0.8669	0.8016
13	0.3466	0.3680	0.8749	0.8114
14	0.3255	0.3465	0.8857	0.8342
15	0.3080	0.3365	0.8917	0.8399
16	0.2920	0.3380	0.8913	0.8369
17	0.2788	0.3108	0.9040	0.8578
18	0.2668	0.3008	0.9083	0.8649
19	0.2565	0.2853	0.9152	0.8728
20	0.2470	0.2735	0.9211	0.8844
Test	-	-	**0.9205**	**0.8834**
Test Recall	-	-	**0.8668**	-
Test Precision	-	-	**0.9022**	-

1. Loss: Both training and validation losses (*Loss*, *Val Loss*) show a steady decrease. This indicates that the model is learning with each training cycle. Initially, the losses are quite high, but by epoch 10 they reduce by almost half. By epoch 20, the validation loss decreases to 0.2735, indicating good model trainability.

2. Accuracy: Validation accuracy (*Val Accuracy*) improves from 0.64 in the first epoch to 0.92 by the 20th epoch, which is a strong indicator of good classification performance. Accuracy demonstrates how well the model classifies the data as it learns.

3. F1-Score: The F1-Score (*Val F1*) also shows gradual improvement with each epoch. It increases from 0.16 to 0.88, reflecting the model's balanced performance in terms of both precision and recall.

4. Final Test Results: On the test set, the model achieved an accuracy of 0.9205 and an F1-Score of 0.8834. This confirms that the model successfully generalized the data without overfitting. The recall and precision values are also quite high at 0.8668 and 0.9022, respectively.

5. Test Recall (0.8668): This means the model correctly identified 86.68% of all true positive cases (all target classes in the test data), an important metric when aiming to minimize missed cases.

6. Test Precision (0.9022): The model accurately classified 90.22% of all positive predictions. This indicates that most of the positive classes predicted by the model were indeed correct.