SHREC 2021 Gesture Benchmark

1 SHREC Datasets

SHREC dataset is made of 180 sequences of gestures. Each sequence has been planned to include from 3 to 5 gestures padded by semi-random hand movements labeled as non-gesture. The original dictionary is made of 18 gestures divded in static gestures, characterized by a static pose of the hand, and dynamic gestures, characterized by the trajectory of the hand and its joints. The dataset is split in a training set with 108 sequences including originally 24 examples per gesture class and a text sequence including about 16 examples per class. However, we discovered some smal bugs in the annotation provided for the contest and we propose, therefore an update of the dataset (annotation only). In particular, we removed a gesture class (POINTING) that had relevant issues related to potential similarity with subsetes of other gesticulations and was not correctly annotated in some cases. And we checked and revised the annotations of all the other gestures, so that the limits of the gestures have sometimes been changed and a few wrong executions been detected.

The list of gestures in the dictionary and data proportion is presented as follows:

Dataset	Training	Testing				
Sequences	108	72				
List of the gestures						
Static Gestures	one	two	three	four	ok	menu
Ynamic Gestures	left	right	circle	v	cross	
Fine Dynamic Gestures	grab	pinch	tap	deny	knob	expand

Tabela 1 – List of gestures and data proportion

2 Task Description

[1]: Models often overfit training data and sometimes fail to learn effectively. Please use any necessary techniques, such as hyperparameter adjustment, model structure modification, or data augmentation, to improve the accuracy of the model (achieving validation accuracy of > 85)

[2]: Write one page document explaining the difference between offline and online gesture recognition. You can use the results of the current notebook to illustrate your point of view. This notebook includes implementation for both offline and online evaluations.

3 Baseline Testing

As mentioned, the current models suffer from the problem of overfitting the training data. This paper first tests seven baseline models. To ensure experimental parity, we set the hyperparameters of all baselines consistently. The experimental results can be compared with the results of the subsequently optimized models.

Parameters:	max length	batch size	epochs	output dim	learning rate	drop out
	250	64	100	17	0.0001	0.3

During the model training phase, we plotted the train loss and validation loss curves. Using accuracy as the evaluation metric, we also plotted the accuracy curves for each baseline on both the training and validation sets. Additionally, we recorded the learning rate changes of the models on the validation set. The results of all curves for the baselines under consistent parameters can be found in the appendix.

We statistically recorded the evaluation metrics of the models on the validation set. The recorded results represent the best performance achieved in three experimental runs of each model, serving as benchmarks for subsequent model optimizations.

	Accuracy	**Precision**	**Recall**	**F1 Score**
CNN	0.1000	0.0993	0.0267	0.0386
small CNN	0.0593	0.0588	0.0035	0.0067
Inception	0.7519	0.7516	0.7786	0.7471
LSTM RES	0.0889	0.0882	0.0230	0.0267
resNet LSTM	0.2963	0.2957	0.3870	0.2543
resNet	0.8184	0.8136	0.5818	0.8156
RNN att	**0.8444**	**0.8435**	**0.8488**	**0.8416**

Tabela 2 – baseline on real validation dataset

From the results, it is evident that ResNet and RNN with attention perform relatively well under initial parameters and without any data preprocessing. Other models show poorer performance on the validation set, with classification accuracies generally ranging from 60% to 80% on the training set. In addition, we conducted online testing for all baselines, considering the real-time recognition system's requirement for recognition speed. We recorded the prediction speeds and results of all pre-trained baseline models on the validation set, along with confusion matrix results for each class. Details can be found in the appendix. Among these baselines, ResNet stands out for its performance.

	Trainable Params	**Time(s)**	**Prediction**	**True**	**Acc**
CNN	8296259	41	[ ]	[14 11 12 16]	0.0
small CNN	152817	38	[ ]	[14 11 12 16]	0.0
Inception	832561	44	[14 15 11 15 12 16]	[14 11 12 16]	0.5
LSTM RES	7234641	75	[ ]	[14 11 12 16]	0.0
resNet LSTM	14545278	72	[7]	[14 11 12 16]	0.22
resNet	1660241	44	[14 11 16]	[14 11 12 16]	0.75
RNN att	56783633	78	[14 13 15 16 11 12 15 12 15 16]	[14 11 12 16]	-0.5

Tabela 3 – Real-time Prediction Results of the Baseline Model

Table 3 records the trainable parameters of each baseline, the time taken for online prediction processing, and details the prediction results and accuracy. Trainable parameters provide a visual representation of the model's size, while online prediction processing time helps understand the corresponding speed and processing rate of each model for real-time predictions. This assessment delay is crucial for evaluating real-time prediction performance.

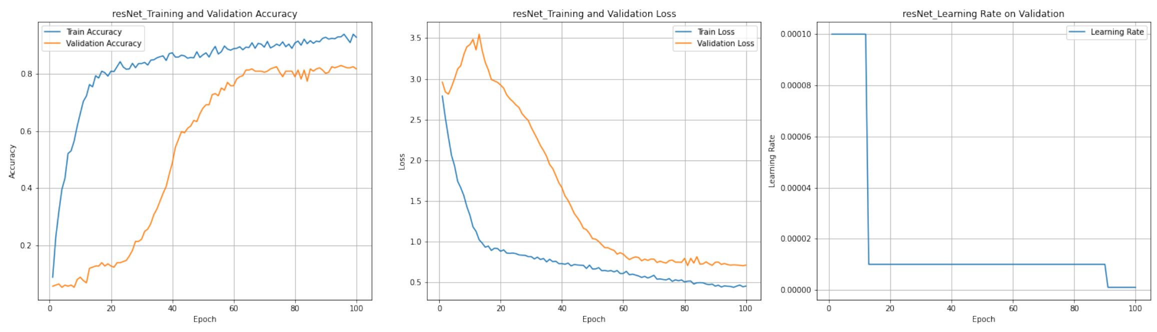


Figura 1 – ResNet performance

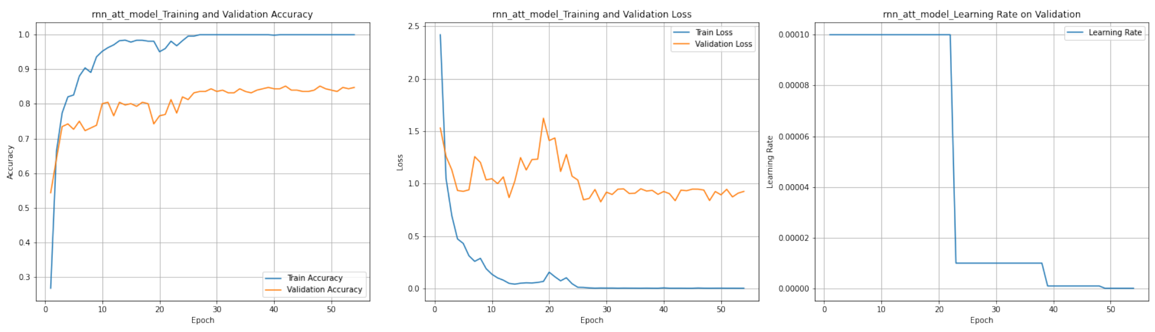
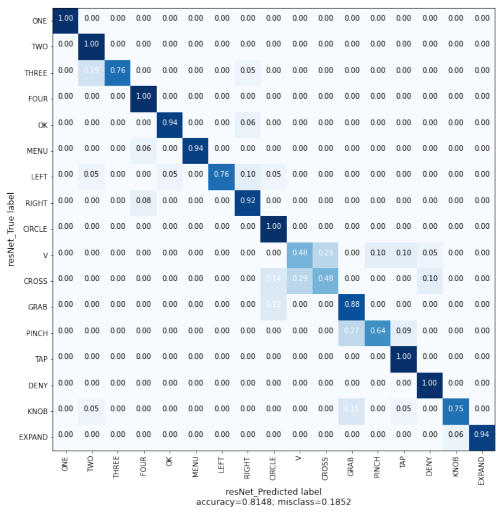
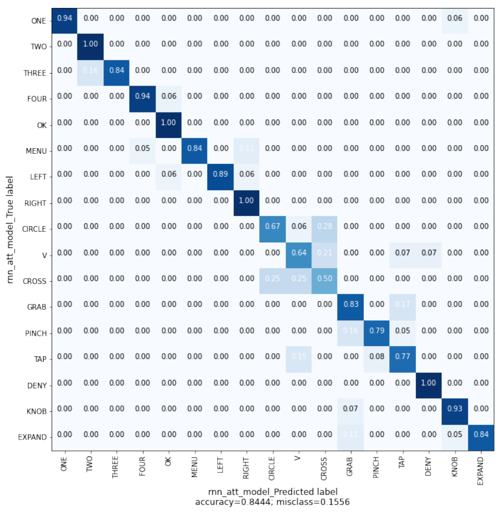


Figura 2 – RNN Att performance

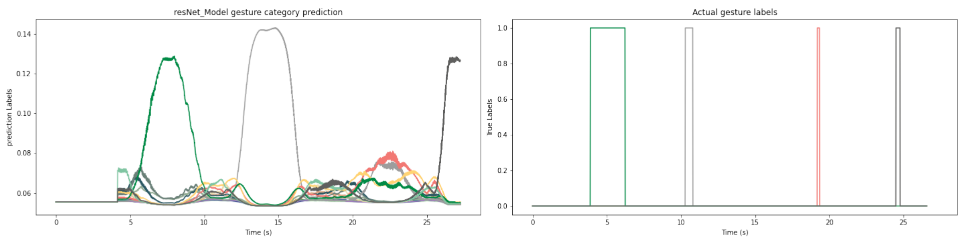


(a) ResNet Confusion Matrix

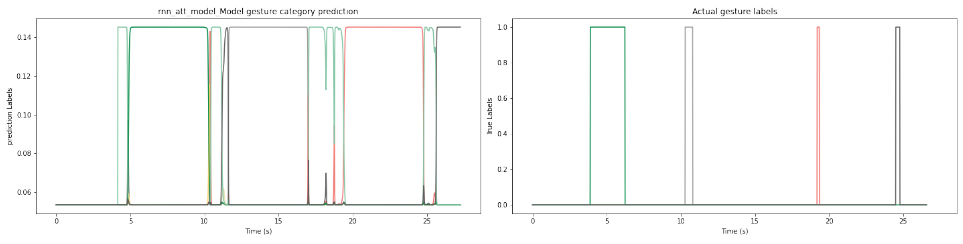


(b) RNN Att Model Confusion Matrix

Figura 3 – RNN Att performance



(a) ResNet Gesture Category Prediction



(b) RNN Att Model Gesture Category Prediction

Figura 4 – RNN Att performance

It is worth noting that the parameters set for the prediction, true value, and accuracy calculations here may not be suitable for the current model. Due to the role of parameters such as threshold in calculating the predicted value and evaluating whether the label is ultimately recorded in the prediction list, it is normal for some models to obtain an empty list of predicted labels under default parameters. The default parameter setting is: $threshold = 0.6, recurthreshold = 20, max = 0.14, min = 0.06$

The corresponding function name is' *evalAccuracyCttd* ', where threshold is used to determine when the predicted gesture label is considered valid, and when the predicted softmax value exceeds the threshold, it is confirmed as a valid prediction. *RecurThreshold* is used to consider the situation of continuous prediction repetition. If the number of predictions for the same gesture exceeds this threshold and the threshold condition is met, these predictions are considered valid results. Max and min are used to set the normalization range of softmax.

4 Optimizations

We will optimize the model from the aspects of hyperparameter optimization, data processing and augmentation, and online evaluation optimization. After testing, we found that the current task is relatively simple, and modifications to the model architecture resulted in only minor performance improvements. In contrast, optimizations from the data perspective and hyperparameter tuning showed significant improvements in evaluation metrics. Therefore, we ultimately decided against modifying the model architecture. The following sections will provide a detailed introduction to the optimization methods and improvements. The effectiveness of these measures will be validated through comparative and ablation experiments.

4.1 Data Processing and Data Enhancement

The overall data processing and augmentation section includes several main steps: data padding, data augmentation, data shuffling, and data type conversion. The following is a detailed explanation of each step:

4.1.1 Padding Data

```python
def pad_data(input_dim, data,max_length):
    data_padded = np.zeros([len(data),max_length,input_dim])
    for i in range(len(data)):
        if len(data[i]) <= max_length:
            data_padded[i,:len(data[i])] = data[i]
        if len(data[i]) > max_length:
            data_padded[i] = data[i][:max_length]
    return data_padded
```

Listing 1 – pad_data function

We first call the ‘pad_data‘ function to pad the training and testing data, ensuring that each sample has a consistent length and checking for the presence of any NaN values.

Next, we perform data augmentation on the training set. First, we define a function ‘augment_data‘, which includes denoising, time transformation, data expansion, and interpolation operations.

```python
def augment_data(train_data, train_label, input_dim, max_x_length, multi=1,
   apply_denoise=True, apply_time_warp=True):
    if apply_denoise:
        train_data = denoise_all(train_data, input_dim)
    if apply_time_warp:
        train_data = [time_warp(stroke) for stroke in train_data]
    train_data, train_label = multiplier2(train_data, train_label, multi=multi)
    train_data = [interpolate(stroke, max_x_length, input_dim) for stroke in
       train_data]
    return train_data, train_label
```

Listing 2 – augment_data function

The implementation code for the denoising part is shown below. Its main function is to denoise the input coordinate data using a Savitzky-Golay filter. The purpose is to smooth the data, reduce noise interference, and thereby improve the model training effect. In the implementation of the denoising part, the denoise_all function is first called to apply the denoise function to all samples in the entire training set. In the denoise function, each dimension of the data for each sample is smoothed separately, and then the denoised coordinate data is concatenated and returned.

```python
def denoise_all(stroke_all, input_dim):
    stroke_new_all = []
    for coords in stroke_all:
        stroke = denoise(coords, input_dim)
        stroke_new_all.append(stroke)
    return stroke_new_all
```

Listing 3 – denoise_all function

```python
def denoise(coords, input_dim):
    stroke = savgol_filter(coords[:, 0], 7, 3, mode='nearest')
    for i in range(1, input_dim):
        x_new = savgol_filter(coords[:, i], 7, 3, mode='nearest')

        stroke = np.hstack([stroke.reshape(len(coords), -1), x_new.reshape(-1, 1)])
    return stroke
```

Listing 4 – denoise function

Next, we apply time warping to the training set. The 'time_warp' function perturbs the time steps and alters the temporal structure of gesture data samples to generate new variant data, thereby achieving data augmentation. In practical applications, different individuals may have variations in the rhythm or speed of their gestures. Time warping can change the original gestures' temporal segments to simulate gesture data from a wider range of individuals, thus improving generalization performance.

```python
def time_warp(stroke, sigma=0.2):
    new_stroke = stroke.copy()
    time_steps = np.arange(stroke.shape[0])
    warped_time_steps = np.sort(np.random.normal(time_steps, sigma))
    for i in range(stroke.shape[1]):
        new_stroke[:, i] = np.interp(time_steps, warped_time_steps, stroke[:, i])
    return new_stroke
```

Listing 5 – time_warp function

The main function of the interpolate function is to perform interpolation on the input gesture data, adjusting its length along the time dimension to max_x_length. This ensures that all samples have the same number of time steps.

```python
def interpolate(stroke, max_x_length, input_dim):
    coords = np.zeros([input_dim, max_x_length], dtype=np.float32)

    if len(stroke) > 3:
        for j in range(input_dim):
            f_x = interp1d(np.arange(len(stroke)), stroke[:, j], kind='linear')
            xx = np.linspace(0, len(stroke) - 1, max_x_length)
            x_new = f_x(xx)
            coords[j, :] = x_new
    coords = np.transpose(coords)
    return coords
```

Listing 6 – interpolate function

The overall data processing and augmentation workflow is shown in the following diagram:

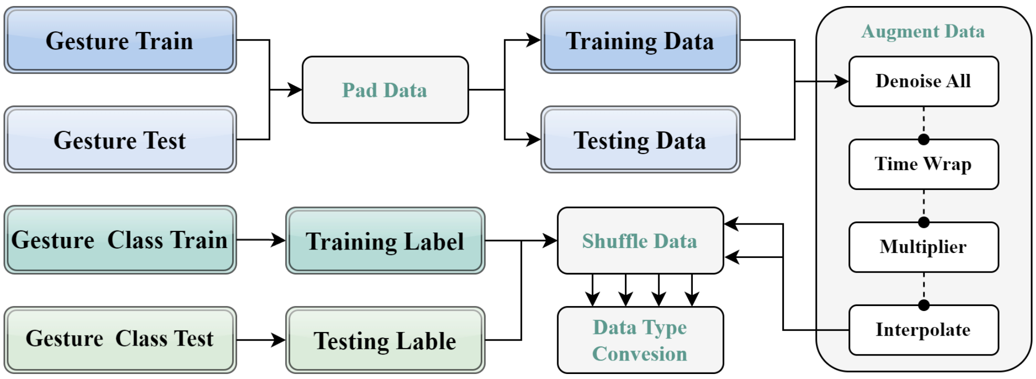


Figura 5 – Data Processing and Augmentation Workflow

4.1.2 Hyperparameter Optimization

The hyperparameter optimization process employed a grid search strategy, mainly adjusting the hyperparameters ‘max_length‘, ‘batch_size‘, ‘learning_rate‘, and ‘drop_out‘. During the hyperparameter optimization, the results of each hyperparameter combination were not recorded, nor were parallel plots drawn to represent them. However, the subsequent experimental results documented the optimal hyperparameter combination, and multiple repeated experiments were conducted to verify the reliability and stability of the results.

4.2 Optimization Results

The specific optimization results are as follows: after data augmentation and hyperparameter optimization, all baselines achieved over 85% on four evaluation metrics on the validation set. Among them, ResNet performed the best, with an accuracy of nearly 96% on the validation set. It also performed well in online real-time prediction. Subsequently, ablation experiments will be conducted around ResNet and ResNet LSTM to verify the reliability of the experimental results.

It is worth noting that Inception, ResNet, and RNNatt provided significant improvements in evaluation metrics compared to the previous baselines. Due to the poor performance of other models on the validation set, their subsequent improvement rates were not calculated.

Model	Accuracy	Acc.%	Precision	Prec.%	Recall	Rec.%	F1 Score	F1.%
CNN*	0.8926		0.8918		0.9046		0.8900	
small CNN*	0.9111		0.9026		0.9079		0.9124	
Inception*	0.8963	19.20%↑	0.8965	11.93%↑	0.9051	16.25%↑	0.8949	19.78%↑
LSTM RES*	0.8630		0.8608		0.8738		0.8593	
resNet LSTM*	0.9444		0.9443		0.9501		0.9443	
resNet*	**0.9593**	17.22%↑	**0.9596**	17.95%↑	**0.9637**	65.64%↑	**0.9589**	17.57%↑
RNN att*	0.8852	4.83%↑	0.8839	4.79%↑	0.9010	6.15%↑	0.8830	4.92%↑

Tabela 7 – Optimization on real validation dataset

Below are the optimal hyperparameter settings for all baselines. The ‘max‘ and ‘min‘ parameters control the normalization range during the prediction process, which are important parameters in online evaluation and directly affect the final classification results.

Parameters:	max length	batch size	epochs	learning rate	drop out	max	min
CNN*	125	32	100	0.001	0.4	0.20	0.09
small CNN*	125	32	100	1e-4	0.25	0.20	0.09
Inception*	125	32	100	2e-4	0.45	0.20	0.09
LSTM RES*	125	32	100	1e-4	0.35	0.20	0.09
resNet LSTM*	125	32	100	0.001	0.4	0.21	0.08
resNet*	125	32	100	0.001	0.4	0.21	0.08
RNN att*	125	32	100	2e-5	0.25	0.20	0.09

Tabela 8 – Optimizing the Hyperparameter Settings of the Model

Below are the plots showing the loss and accuracy of ResNet and ResNet LSTM Model during training, as well as the learning rate changes on the validation set. Additionally, the confusion matrix on the validation set and the real-time prediction curves during online evaluation are provided.

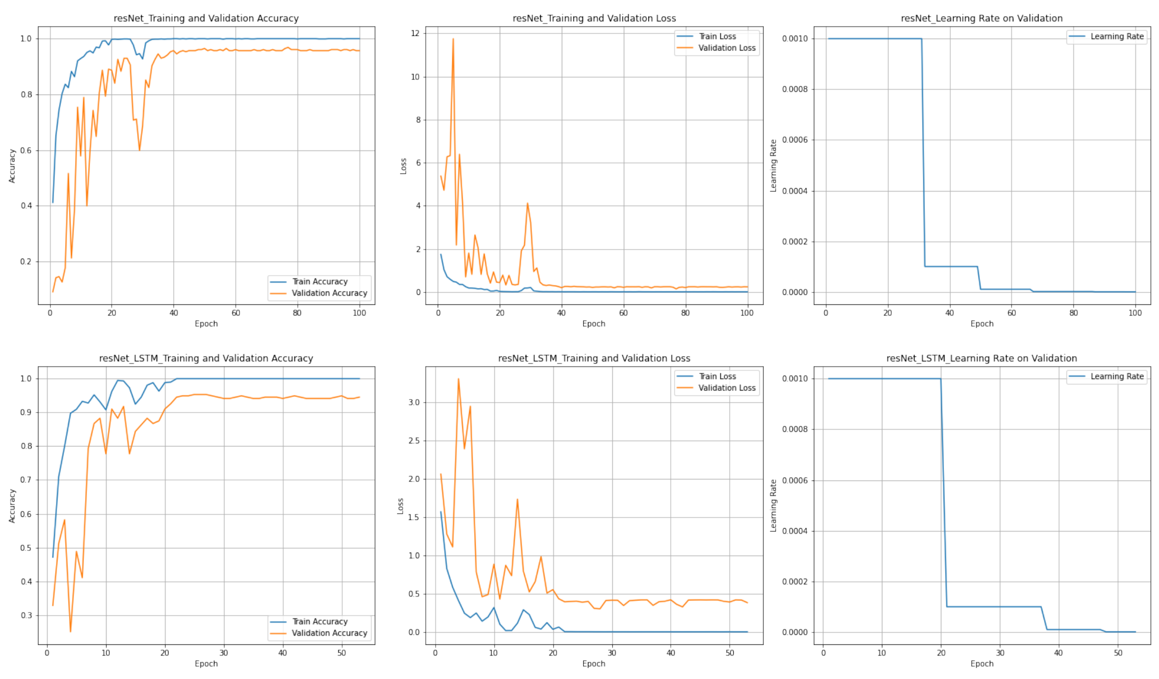
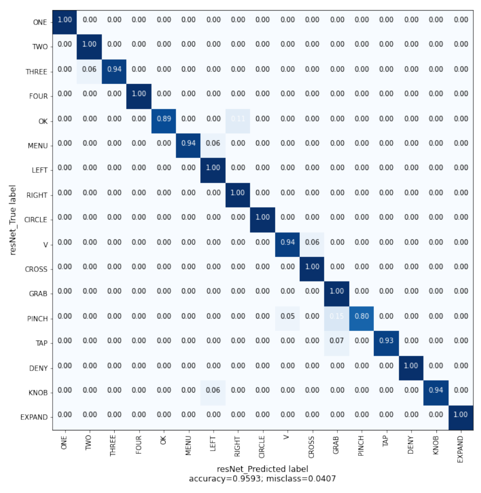
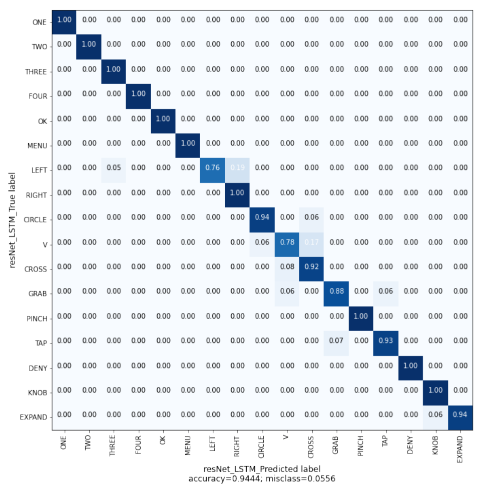


Figura 6 – Performance of the Optimized ResNet and ResNet_LSTM Model



(a) ResNet Confusion Matrix



(b) RNN Att Model Confusion Matrix

Figura 7 – RNN Att performance

4.3 Ablation Study

Next, we conducted ablation experiments using the best-performing ResNet and ResNet LSTM models to verify the effectiveness of the above processing flows.We designed four ablation experiment scenarios, namely setting only the optimal hyperparameter (wp), not setting the optimal hyperparameter (w/op), not performing data augmentation (w/oda), and not performing data shuffling (w/os).The part of the process in Figure 5 where data augmentation is not performed refers to the augment Data section.

Models	Accuracy	Precision	Recall	F1 Score
		ResNet		
ResNet*	$\mathbf{0.9593} \pm 0.008$	$\mathbf{0.9596} \pm 0.007$	$\mathbf{0.9637} \pm 0.005$	$\mathbf{0.9589} \pm 0.007$
$w\ p$	0.2132 ± 0.007	0.2133 ± 0.006	0.2133 ± 0.006	0.1598 ± 0.003
$w/o\ p$	0.9185 ± 0.017	0.9186 ± 0.017	0.9375 ± 0.009	0.9364 ± 0.008
$w/o\ da$	0.9424 ± 0.008	0.9360 ± 0.007	0.9309 ± 0.035	0.9174 ± 0.031
$w/o\ s$	0.9370 ± 0.003	0.9364 ± 0.004	0.9417 ± 0.005	0.9364 ± 0.003
		ResNet_LSTM		
ResNet_LSTM*	$\mathbf{0.9444} \pm 0.012$	$\mathbf{0.9443} \pm 0.010$	$\mathbf{0.9501} \pm 0.012$	$\mathbf{0.9410} \pm 0.014$
$w\ p$	0.8439 ± 0.022	0.8416 ± 0.018	0.8595 ± 0.020	0.8440 ± 0.020
$w/o\ p$	0.9037 ± 0.004	0.9039 ± 0.005	0.9112 ± 0.012	0.9020 ± 0.004
$w/o\ da$	0.9005 ± 0.017	0.8897 ± 0.016	0.9041 ± 0.026	0.8985 ± 0.016
$w/o\ s$	0.8841 ± 0.002	0.8839 ± 0.002	0.8908 ± 0.002	0.8817 ± 0.003

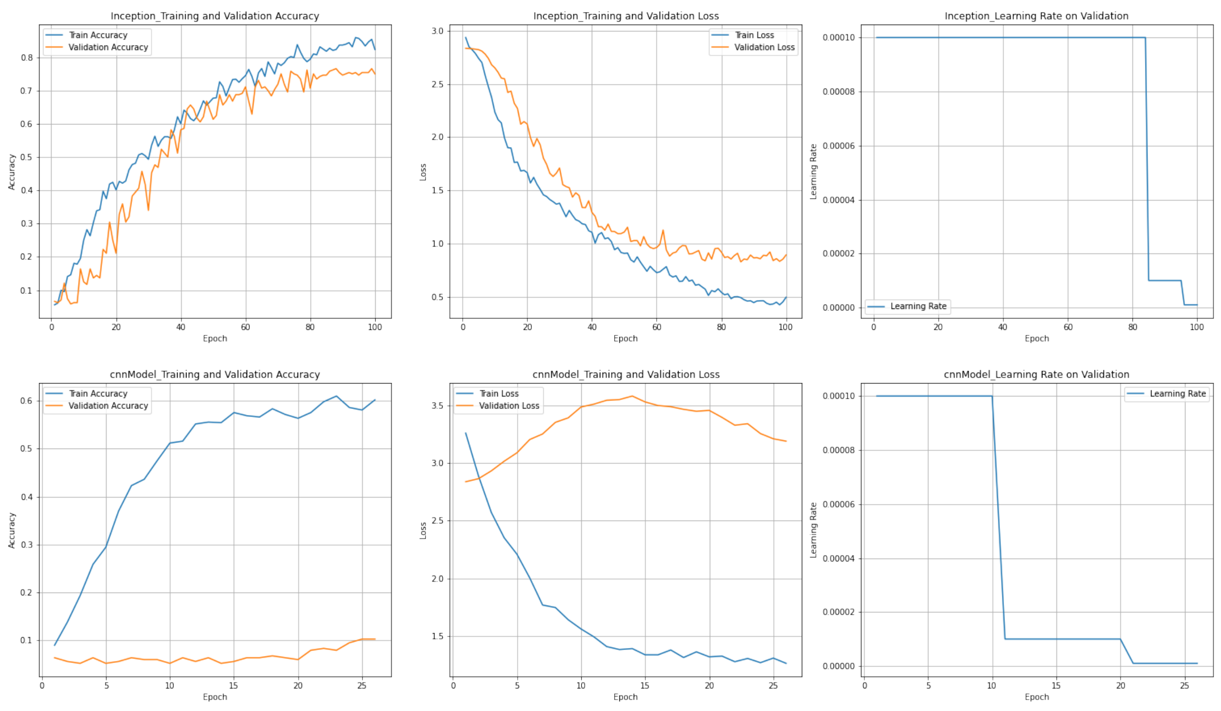
Tabela 10

From the ablation experiment results, it can be concluded that, taking ResNet as an example, optimized hyperparameters alone cannot achieve good results, but by using methods such as data augmentation, ResNet can achieve a recognition accuracy of nearly 96% on the validation set.Similarly, all data augmentation and processing processes have a positive impact on the final results, and can be used alone to achieve a certain degree of improvement on the original baseline.

Therefore, this ablation experiment proves that the current optimization measures are effective and universal.

4.4 Baseline Experiment Results

In this chapter, the experimental results of the baseline model will be presented:



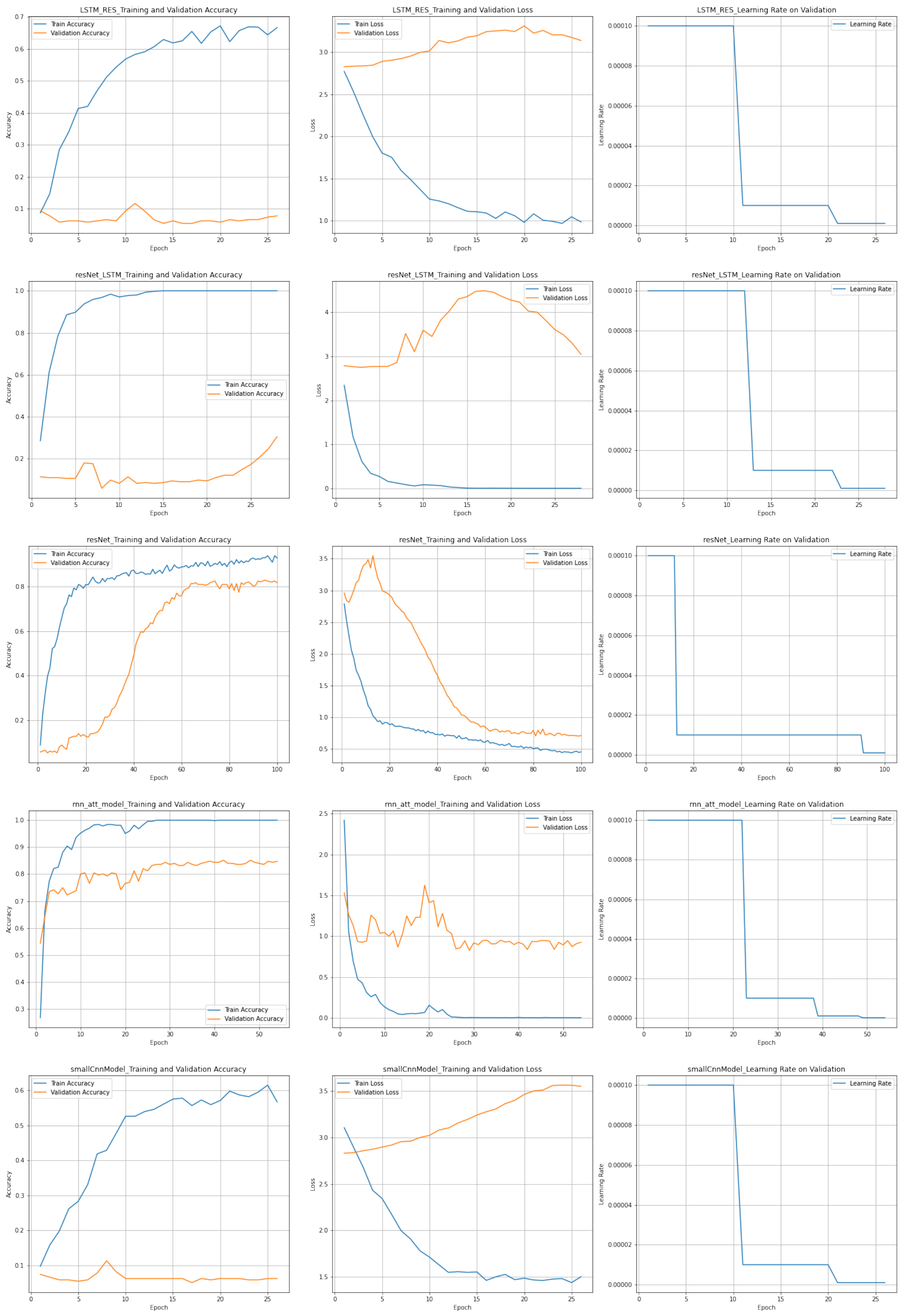
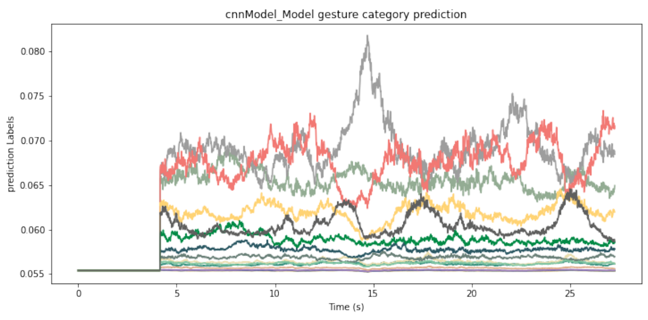
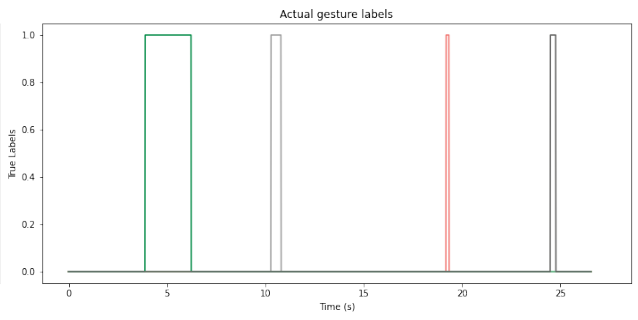
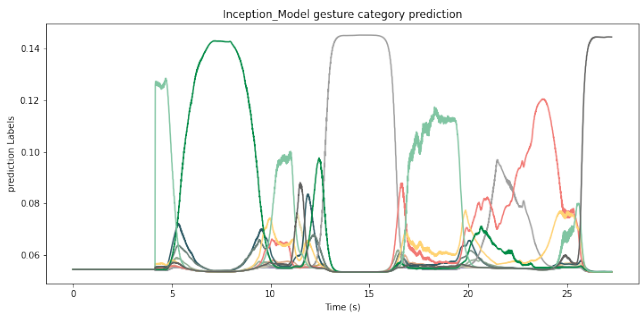
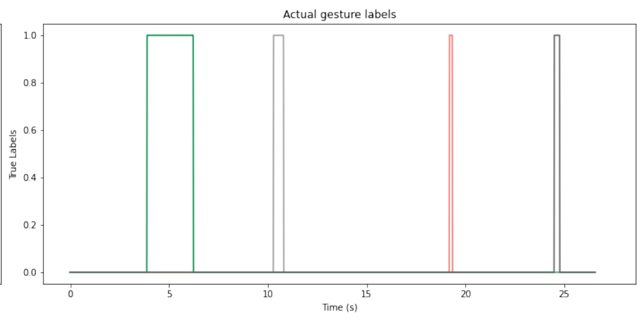


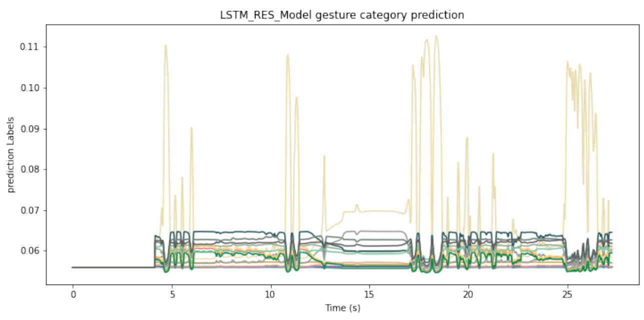
Figura 8 – Model Performance of the Baseline Model

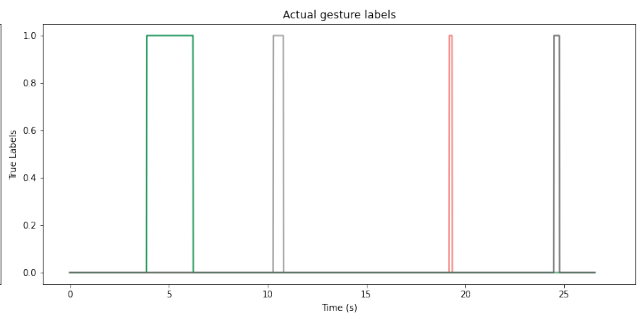

cnnModel_Model gesture category prediction
prediction Labels
Time (s)

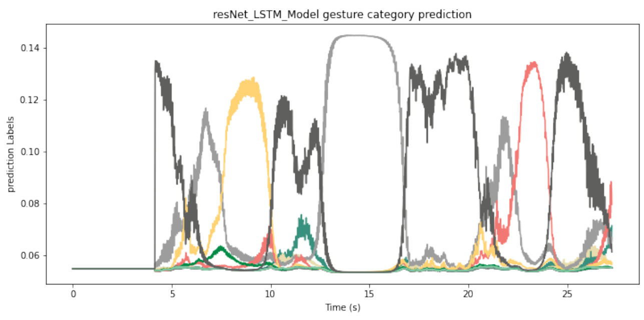

Actual gesture labels
True Labels
Time (s)

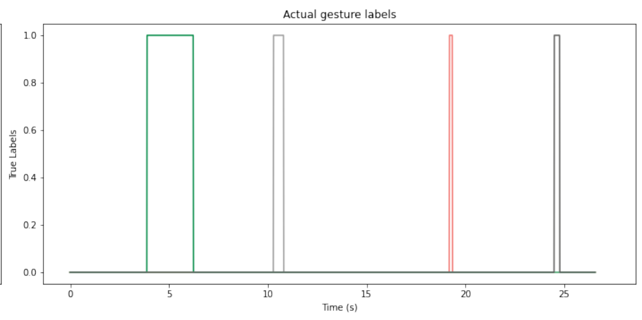

Inception_Model gesture category prediction
prediction Labels
Time (s)

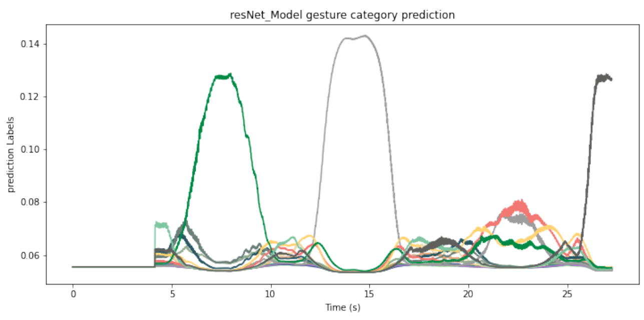

Actual gesture labels
True Labels
Time (s)

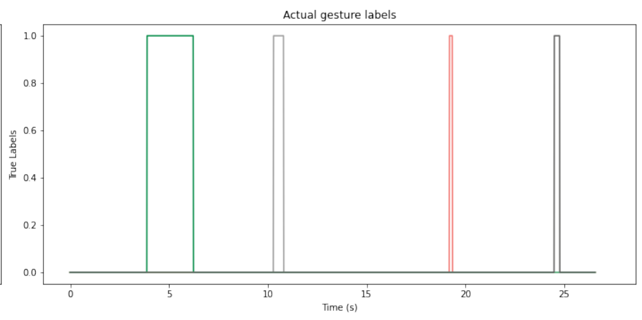

LSTM_RES_Model gesture category prediction
prediction Labels
Time (s)

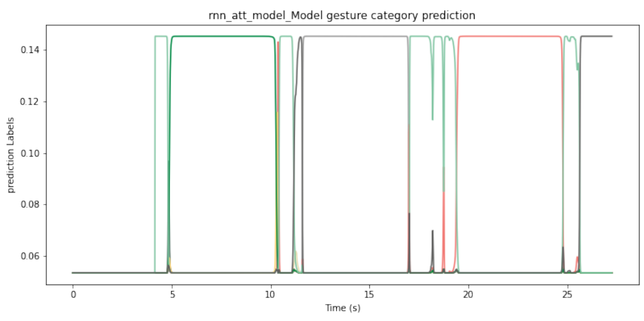

Actual gesture labels
True Labels
Time (s)

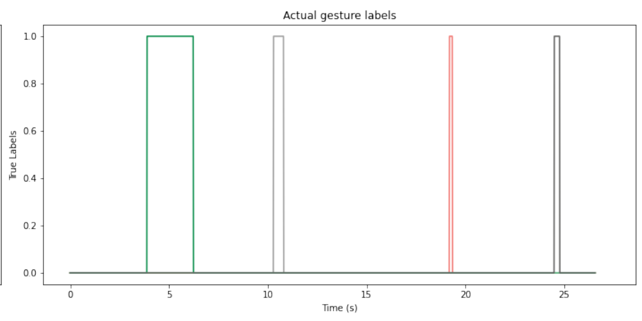

resNet_LSTM_Model gesture category prediction
prediction Labels
Time (s)


Actual gesture labels
True Labels
Time (s)


resNet_Model gesture category prediction
prediction Labels
Time (s)


Actual gesture labels
True Labels
Time (s)


rnn_att_model_Model gesture category prediction
prediction Labels
Time (s)


Actual gesture labels
True Labels
Time (s)

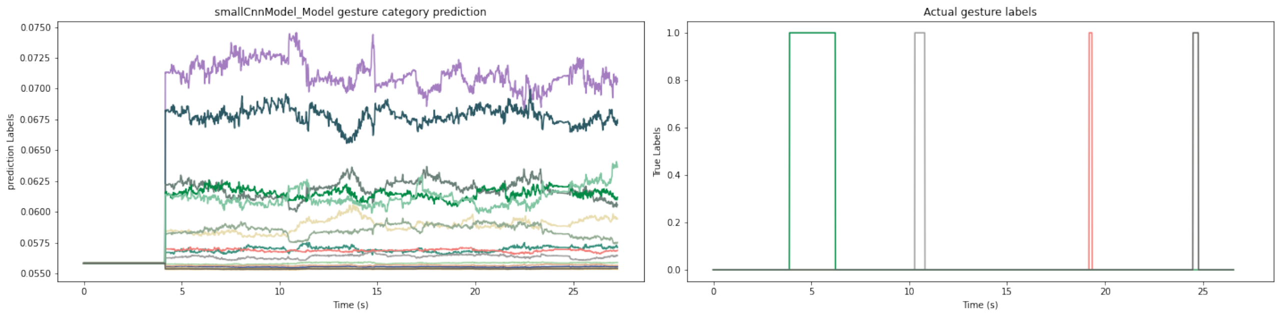
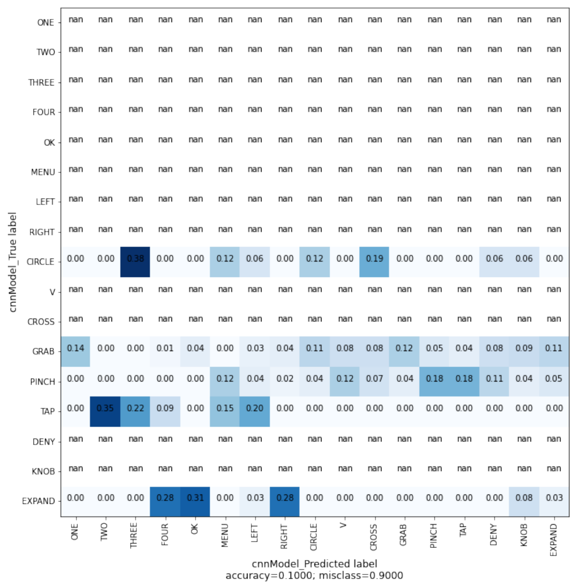
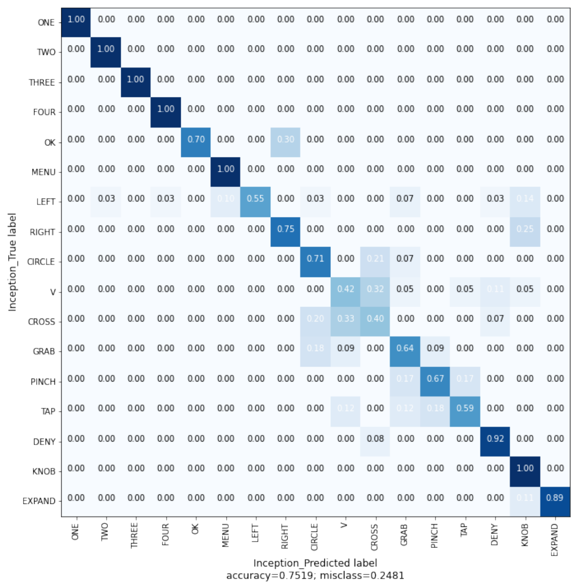
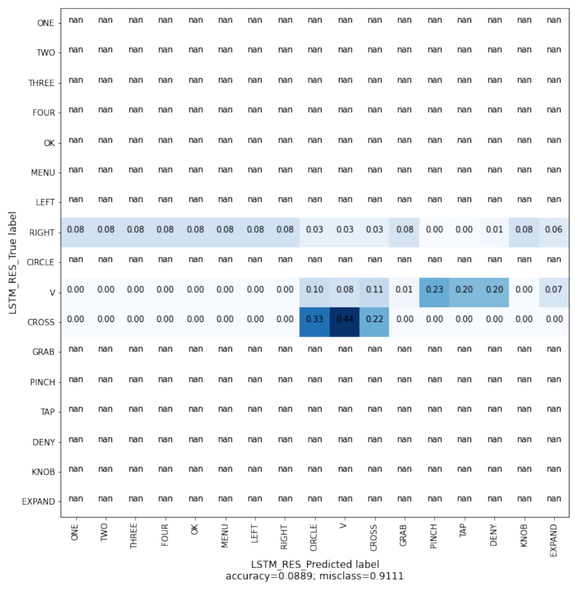
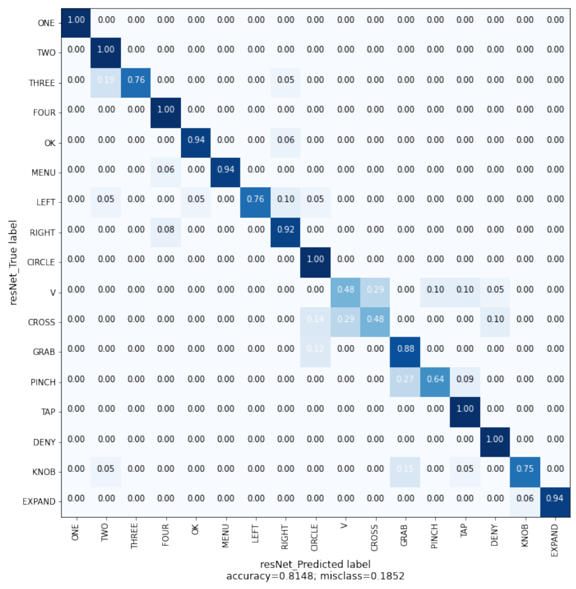


Figura 9 – Gesture Category Prediction of the Baseline Model









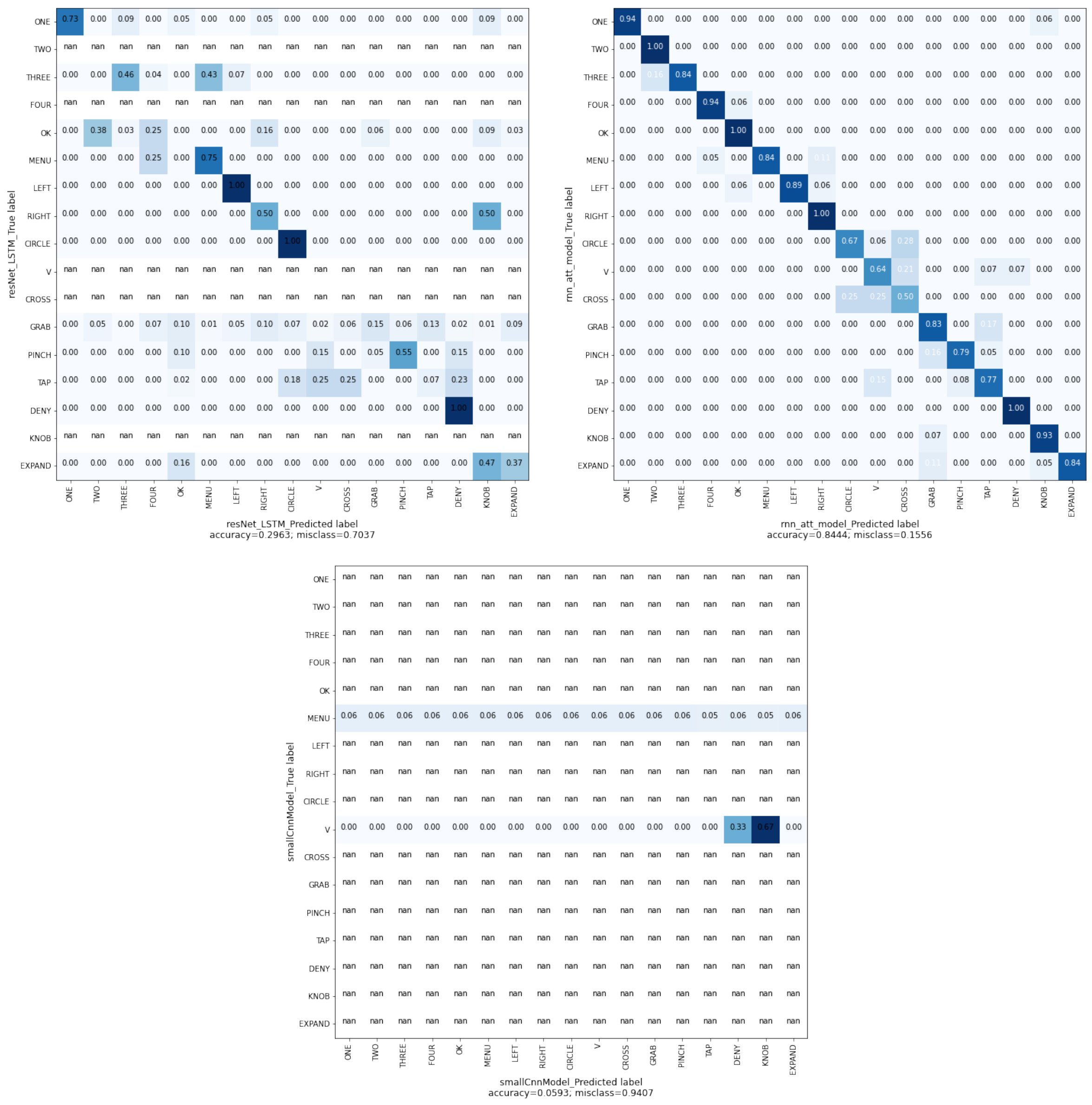
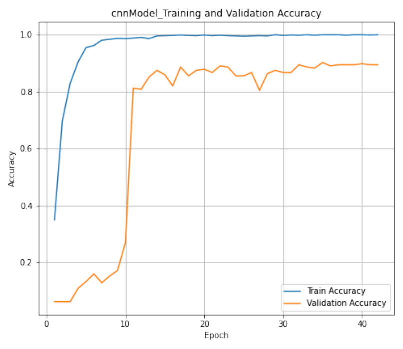
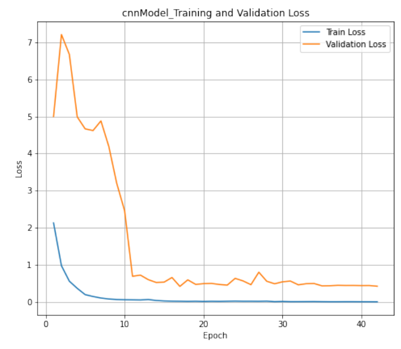
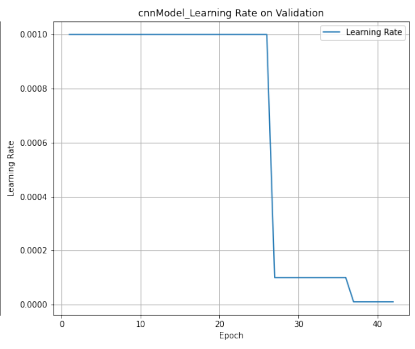


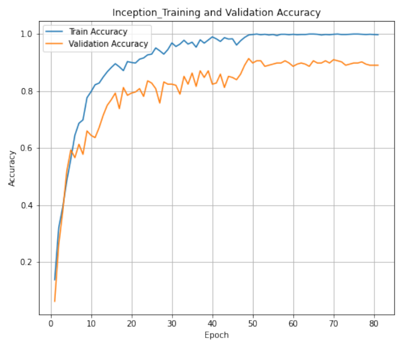
Figura 10 – Confusion Matrix of the Baseline Model

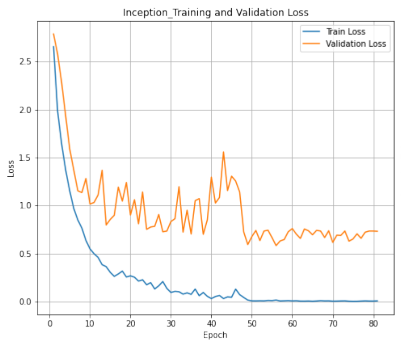
4.5 Experimental results of Optimized Model

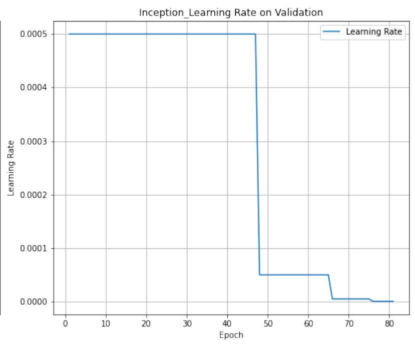

cnnModel_Training and Validation Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy

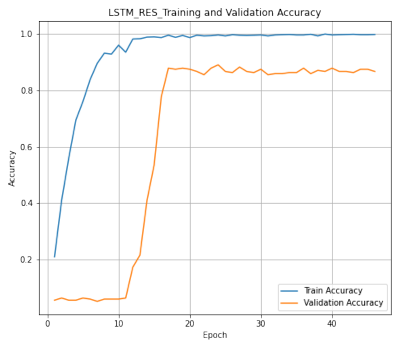

cnnModel_Training and Validation Loss
Loss
Epoch
Train Loss
Validation Loss

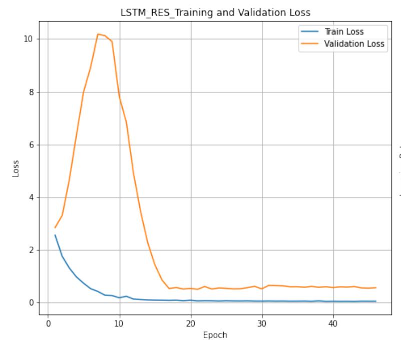

cnnModel_Learning Rate on Validation
Learning Rate
Epoch
Learning Rate

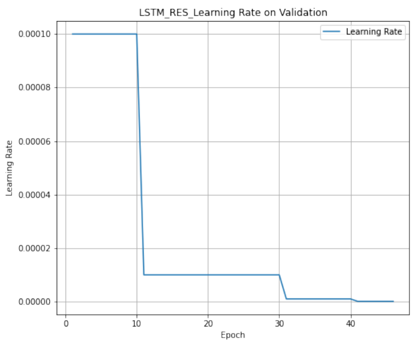

Inception_Training and Validation Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy

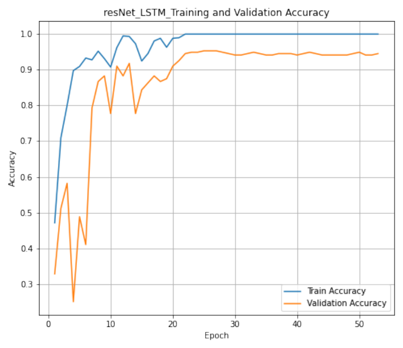

Inception_Training and Validation Loss
Loss
Epoch
Train Loss
Validation Loss

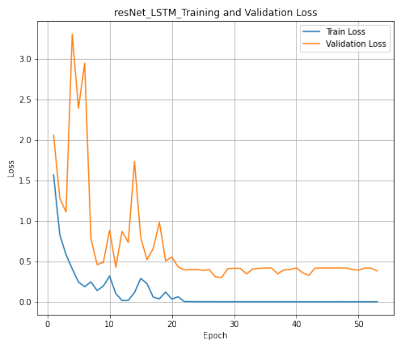

Inception_Learning Rate on Validation
Learning Rate
Epoch
Learning Rate

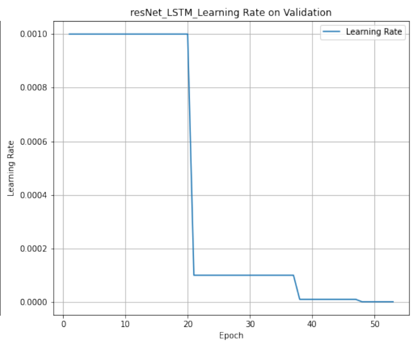

LSTM_RES_Training and Validation Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy

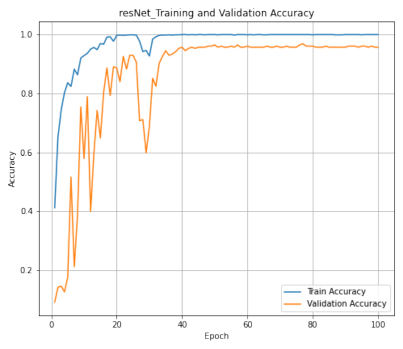

LSTM_RES_Training and Validation Loss
Loss
Epoch
Train Loss
Validation Loss

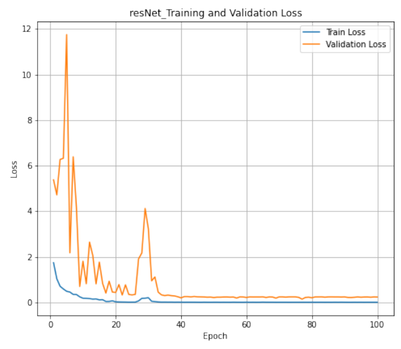

LSTM_RES_Learning Rate on Validation
Learning Rate
Epoch
Learning Rate

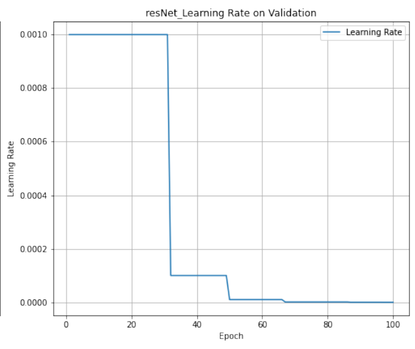

resNet_LSTM_Training and Validation Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


resNet_LSTM_Training and Validation Loss
Loss
Epoch
Train Loss
Validation Loss


resNet_LSTM_Learning Rate on Validation
Learning Rate
Epoch
Learning Rate


resNet_Training and Validation Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy


resNet_Training and Validation Loss
Loss
Epoch
Train Loss
Validation Loss


resNet_Learning Rate on Validation
Learning Rate
Epoch
Learning Rate

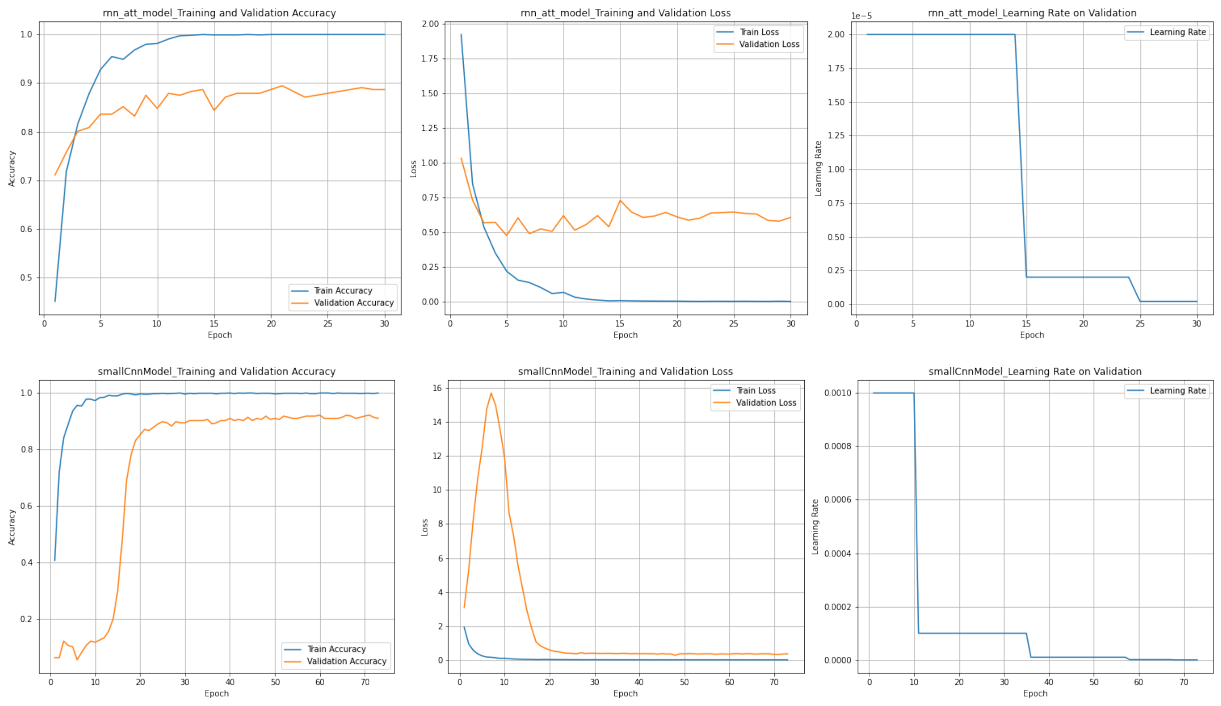
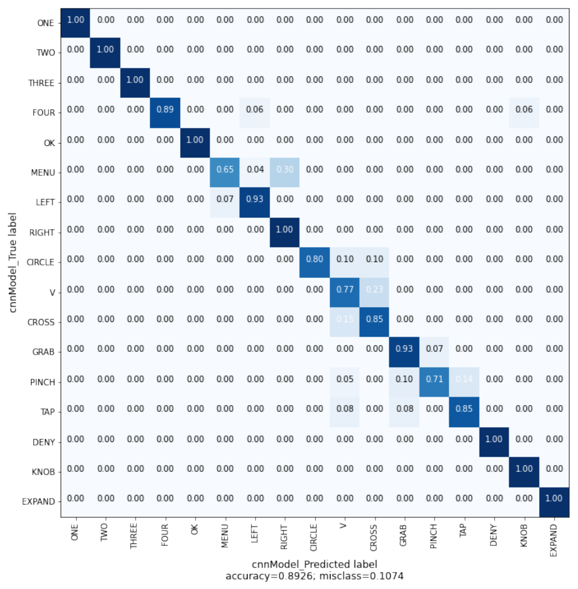
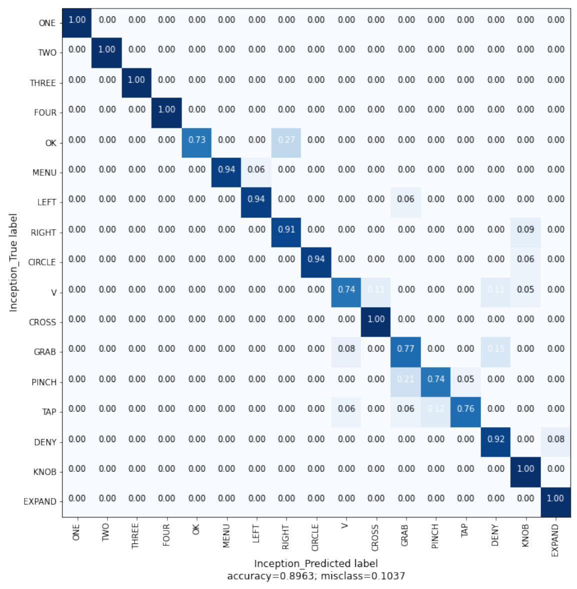
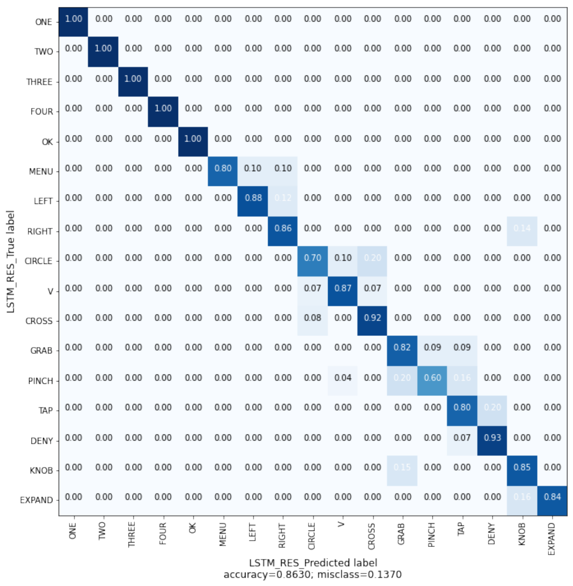
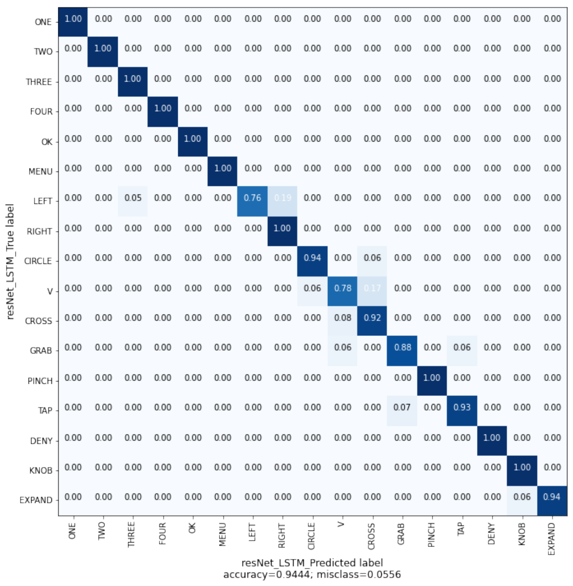


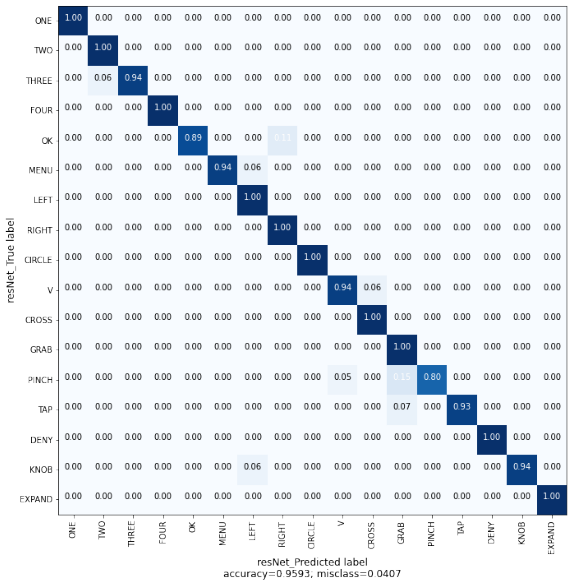
Figura 11 – Model Performance of the Optimized Model

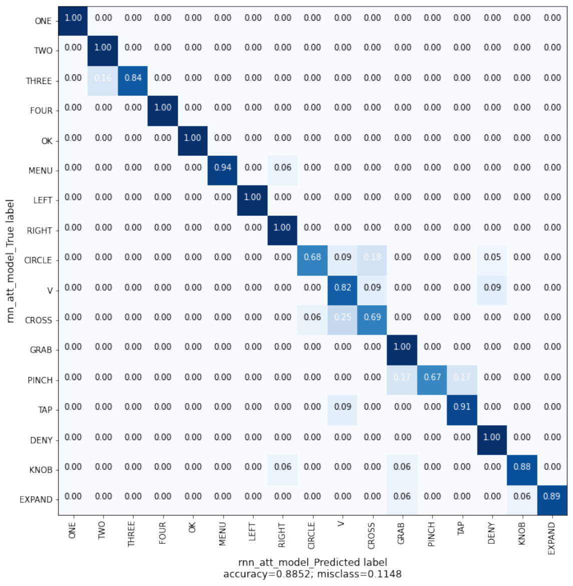












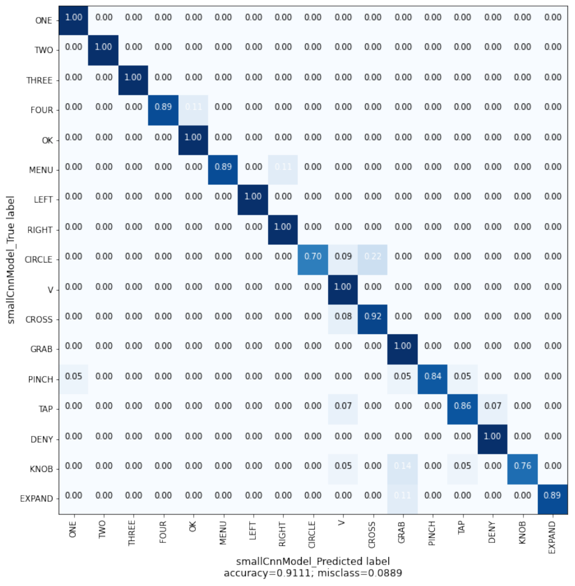


Figura 12 – Confusion Matrix of the Optimized Model

5 Offline and Online Gesture Recognition

We will introduce the similarities and differences between offline gesture recognition and online gesture recognition from the perspectives of tasks, algorithms, and processing approaches. Finally, we will propose feasible optimization measures for online prediction.

5.1 Task level differentiation

Online gesture recognition requires data processing and real-time prediction to begin as the user performs the gesture. This task demands high real-time performance and algorithmic efficiency. In contrast, offline recognition involves processing and analyzing data after complete collection. Since there are no real-time requirements, more complex algorithms can be employed to improve the accuracy of task recognition.

5.2 Algorithm and processing approach

In the original code, online prediction is performed using a sliding window for partial prediction, gradually processing small data segments through the sliding window. This process simulates the system's behavior when receiving real-time gesture data, where a short waiting period occurs until the data reaches the sliding window size, after which prediction is made. This approach is faster compared to offline prediction. However, because only partial data is used, the prediction accuracy is relatively lower. Additionally, during the prediction process, the results are likely to change frequently, as the features of the partial data may still exist in other gesture categories. This reduces the differences between gesture categories, making them harder to distinguish.

Offline prediction can wait for a complete gesture to be performed before making an overall prediction, resulting in higher accuracy. However, it is important to determine the interval between two gestures. In the original code, the online prediction process involves concatenating multiple gesture data segments and making predictions using a sliding window. Therefore, there is no distinct boundary between different gesture data segments, simulating the process of continuous gesture prediction.

5.3 Code level and function implementation

```python
def online_rec(model,max_length, window_step, input_test):
    window_width = max_length
    outputs_list = []
    y_pred_list = []
    y_pred_value_list = []

    for window in tqdm.tqdm(np.arange(0, int(len(input_test) - window_width) + 1,
        window_step)):
        window = int(window)

        input_slice = np.expand_dims(input_test[window:window + window_width], axis
            =0)

        Data = tf.data.Dataset.from_tensor_slices(
            (input_slice)
        )
        Data = Data.batch(1).prefetch(buffer_size=1)

        outputs = model.predict(Data, verbose=0)
        outputs_softmax = tf.nn.softmax(outputs)
        y_pred = np.argmax(outputs_softmax, axis=1)
        y_pred_value = np.max(outputs_softmax)
```

```

            outputs_list.append(outputs_softmax)
            y_pred_list.append(int(y_pred))
            y_pred_value_list.append(float(y_pred_value))

    return y_pred_list, outputs_list, y_pred_value_list
```

Listing 7 – online_rec function

In the online prediction, the online_rec function is invoked, which returns y_pred_list, outputs_list, and y_pred_value_list, representing the predicted labels, predicted outputs, and prediction confidence information, respectively. This function is based on a sliding window approach for online prediction of the input gesture recognition data. The invocation requires setting the step size, maximum length of the sliding window, and the test data. The prediction is performed by calling the trained model to obtain the outputs, which are then converted to a probability distribution using softmax.

Subsequently, the eval_accuracy_nttd function utilizes the outputs returned by online_rec. By setting confidence intervals, it filters the prediction results to obtain a list of predictions within the specified thresholds and threshold counts. This can be understood as an offline global prediction screening based on the results of the online prediction. Therefore, this does not fall under the category of online prediction; it merely evaluates the accuracy of the online prediction results.

The eval_accuracy_NTtD function reflects the prediction latency characteristics. It calculates the prediction frame time by taking element values using frame_idx and is used to assess the model's recognition accuracy and the positional accuracy of the predicted labels. Similarly, it also serves as an evaluation method for the online prediction.

5.4 online prediction performance of the optimized model

In online prediction, in addition to the performance results on the test set, a temporal prediction curve is also generated, reflecting the changes in the predicted values during the real-time prediction process. The model's real-time prediction performance is evaluated by calling evaluate_nttd and evaluate_NTtD.

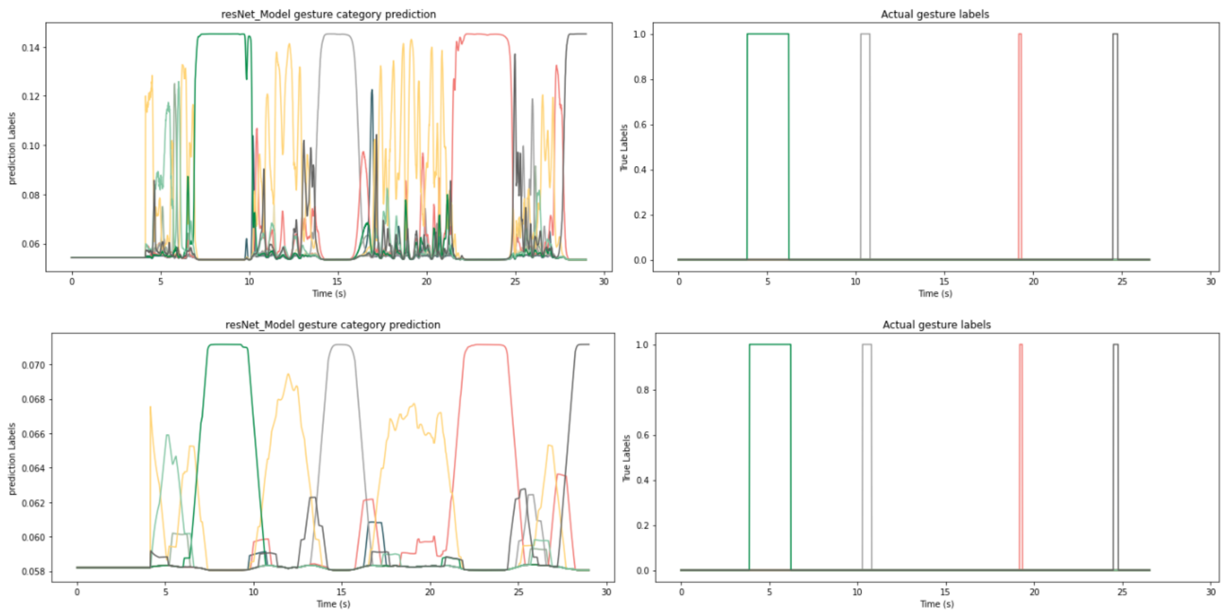
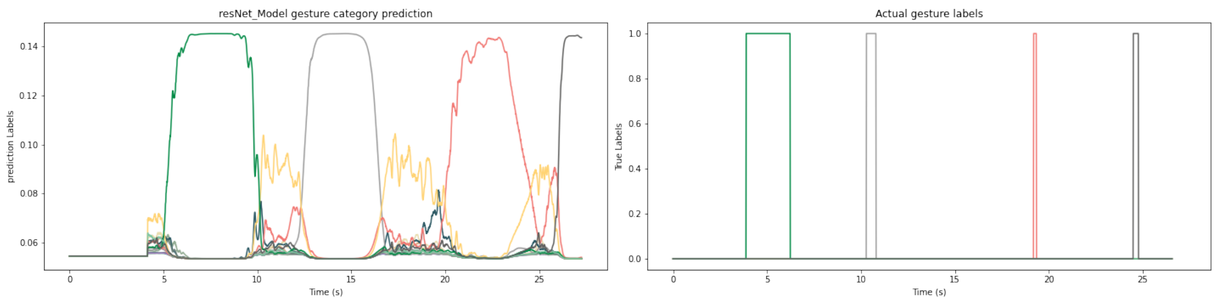


Figura 13 – Optimized ResNet Gesture category prediction

From the figure, taking the optimized ResNet as an example, it can be seen that in the real-time prediction curve, when using the sliding window for prediction, the model shows certain prediction probabilities for multiple categories in some local regions. This is a normal phenomenon, as the local features of some gestures may appear in multiple gestures, leading to higher recognition rates for multiple categories during prediction. In terms of prediction real-time performance, it can be observed that the predicted interval differs from the actual gesture action interval by about 3 seconds.

From the figure, it can be seen that the prediction values for incorrect categories are also relatively high. If we wish to reduce the model's capture of finer features or the common features between different gestures, we can appropriately lower the learning rate. Although this might reduce the model's recognition accuracy, it can help minimize the occurrence of blind predictions during real-time prediction.

The second prediction graph includes a smoothing improvement on the original prediction algorithm, which helps filter out noise-induced prediction fluctuations and smooths out drastic changes.



Of course, by sacrificing some accuracy, the model can be made less prone to overfitting, and maintaining insensitivity to local features can reduce the frequency of predictions and the number of incorrect predictions during the real-time prediction process mentioned above.

5.5 Analysis of Optimizing Model Real-Time Prediction Latency

```python
def evaluate_NTtD_3(classes, frame_idx, frame_sequence, y_true, y_pred):
    frame_pred = (np.array(frame_idx)+100 )[::2]
    frame_true = frame_sequence

    print("frame_idx:", frame_idx)
    print("frame_pred:", frame_pred)
    print("frame_true:", frame_true)

    for j, b in enumerate(y_true):
        print("Processing true label:", b)
        for i, a in enumerate(y_pred):
            start = frame_true[j * 2]
            end = frame_true[j * 2 + 1]
            print("Checking predicted label:", a, "with true label:", b)
            print("Predicted frame:", frame_pred[i], "True frame range:", start, "-", end)

            if a == b and start < frame_pred[i] < end:
                NTtD = (frame_pred[i] - start + 1) / (end - start + 1)
                print('{} is recognized correctly at {} seconds, with NTtD {}'.format(
                    classes[b].upper(), frame_pred[i] / 72, NTtD))
```

Listing 8 – evaluate_NTtD

This function calculates the NTtD value by comparing the start and end frames in frame_true corresponding to the actual labels with the predicted start and end frames of the predicted labels. The calculation formula for NTtD is as follows:

$$NTtD = \frac{frame\_pred[i] - start + 1}{end - strat + 1}$$

Using the formula, the NTtD values for the four prediction results of the optimized ResNet model are calculated as follows:

Frame_idx	243,244,729,730,1297,1298,1717,1718
Frame_pred	343,829,1397,1817
Frame_true	280,449,741,778,1382,1393,1764,1784

The NTtD value reflects the distance between the predicted frame and the end frame of the actual label. When the predicted frame coincides with the end frame of the actual label, this value is 1. If the value is greater than 1, it indicates that the model's prediction is delayed, and the extent of the delay can be reflected by the NTtD value.

True Lable	**Predicted Label**	**True Frame Range**	**Predicted Frame**	**NTtD**
14	14	[280,449]	343	0.376
11	11	[741-778]	829	2.225
12	12	[1382-1393]	1392	0.917
16	16	[1764-1784]	1817	2.571

Tabela 23

From the final results, the optimized ResNet model performs relatively well in real-time prediction. The correctly predicted frames are close to the actual end frames, demonstrating low latency and high recognition rate in real-time prediction.