SmartGloveXfer: Detection and transcription of movements with Transfer learning

REY Quentin MULLER Guillaume
Mines Saint-Etienne, Institut Henri Fayol, Saint-Etienne, France

Abstract

Handwritten letter recognition poses a significant challenge, particularly in scenarios where data availability is limited. While neural networks can achieve respectable accuracy in user-dependent settings with sparse data, their performance often plummets when presented with new handwriting styles (i.e., in user-independent settings) due to inadequate generalization.

This article investigates the application of transfer learning techniques to address this issue and enhance prediction accuracy for novel writing styles. This exploration was conducted making use of a physical wearable device equipped with an Inertial Measurement Unit (IMU) sensor, known as the Smart Glove. This device is capable of capturing acceleration, gyroscopic, and magnetic data corresponding to finger movements during lowercase letter writing.

A CNN and an LSTM were employed with distinct datasets. The LSTM achieved an accuracy rate of 89.19% and 71.02% in user-dependent and user-independent settings, respectively. Meanwhile, the CNN achieved accuracy rates of 82.46% and 69.26% in user-dependent and user-independent settings, respectively. Notably, employing transfer learning enhanced user-independent accuracy by 16% compared to the previous best model, thereby notably enhancing generalization across various writing styles.

Keywords— Machine Learning, Deep learning, Transfer learning, Fine-tuning, Handwriting Recognition, CNN, LSTM

1 Introduction

Nowadays most documents are created digitally through the use of laptops or smartphones, using a keyboard or directly by writing on the screen. However, all these methods make use of a physical surface to be able to write. Eliminating the need for physical devices holds considerable promise in specific contexts. For example, in virtual or augmented reality settings where physical keyboards aren't available, individuals could seamlessly take notes. Such technology could also be used inside smart factories or smart class-room where it could be integrated to devices used to teach the hand movements of beginners [Luo et al., 2024].

This article addresses a specific challenge: recognizing lowercase English alphabet letters drawn in the air using a wearable device powered by a microcontroller and battery, equipped with an IMU sensor. However, due to the limited recorded dataset (ie: target dataset) compared to the variability in handwriting styles, the issue of generalizing to new handwriting naturally emerges. To mitigate this challenge, the paper investigates transfer learning techniques, leveraging similar datasets from the internet (ie: source datasets) to enhance the model's generalization capabilities. The article proceeds by conducting a survey of previous works and related research. Subsequently, it delves into the data utilized and the preprocessing techniques applied. The paper then elaborates on the models employed, including LSTM and CNN networks, and presents the corresponding results. The code for the preprocessing, the training, and the evaluation of the models is accessible on a GitHub repository.

These models will be evaluated in 2 different settings:

- A user dependent setting: 200 randomly selected samples are extracted from the target dataset and constitute the validation dataset. The remaining data and the source dataset constitute the training set. This means that, when making a decision, the neural network likely already saw examples of the same letter drawn by the same person.
- A user independent setting: The test dataset is composed of data of a single user (ie: 100 examples containing 10 letters) that was excluded from the training dataset. The training dataset is composed of all the other users as well as the data from the source dataset.

2 Previous work

Previous works already utilized the same device, developed in a prior project [Gaffarov, 2023]. This device features an IMU sensor positioned at the fingertip [Appendix A]. [Gaffarov, 2023] additionally provided 2 datasets containing lowercase letters from the English alphabet. One contains 1374 examples of 27 points each drawn by a single person. The second one contains 1600 samples drawn by 16 different people. It provides 32 points over a period of 1.2 seconds. Each person drew 10 samples for 10 randomly chosen letters. For this research project, the second dataset (ie: multi-user dataset) was employed.

Additionally, [Gaffarov, 2023] also examined these datasets, employing machine learning techniques such as LSTM and CNN for letter recognition. On the multi-user dataset, these techniques achieved 78% and 80% accuracy respectively for the CNN and the LSTM on a user dependent setting. These results were obtained by making use of neural networks with few parameters to adapt to the small amount of data available.

A second project [Zrira et al., 2024] was conducted on the subject. This project experimented with a Random forest and got an accuracy of 85.6%. Then, to allow the model to exploit the temporal component of the signal, an LSTM was used, which allowed for an accuracy of 93%. Unfortunately, when the same model was trained on a totally new user (ie: user independent setting), this accuracy dropped to 55%, which highlights the need to find a way to better generalize to new handwriting styles.

3 Related work

Recent research has explored the potential of IMU sensors for character detection, revealing two primary approaches.

Some studies aimed at enhancing network performance on existing data, neglecting adaptation to new handwriting styles. Consequently, these networks struggled in this aspect. Examples include prior projects on the same device ([Gaffarov, 2023] and [Zrira et al., 2024]) as well as [Chen et al., 2021], where researchers utilized an IMU sensor positioned at the other end of a pen to classify English alphabet letters. Although good results were achieved for known users using RNN and CNN networks (CNN: 78% acc, LSTM: 86.6% acc), accuracy sharply declined in user-independent settings (CNN: 40.2% acc, LSTM: 53.6% acc).

Other studies endeavored to address both challenges simultaneously. [Dash et al., 2017] exemplify this approach. They employed an IMU sensor to capture data of users drawing numbers in the air. They extracted a 2D representation of the numbers which was then prepossessed and interpolated in order to get clean, evenly spaced points. These points were then used to classify the samples thanks to a combination of 2 GRU and a pre-trained CNN through a fusion model making use of Borda Count. With this method, they achieved an accuracy of 91.7% in a person independent evaluation and a 96.7% accuracy in a person dependent evaluation. However, this method utilized 1270 examples to classify digits into 10 classes, a relatively larger dataset compared to the task complexity. Additionally, they partially utilized transfer learning techniques by fine-tuning a pre-trained model.

This paper also aims to address both challenges simultaneously while managing a larger number of classes and a smaller dataset size. To do so, it will leverage transfer learning methods.

4 Data acquisition

4.1 Data format

The smart glove's IMU sensor captures acceleration, gyroscopic, and magnetic data. To facilitate transfer learning, it is crucial to find data for which there is a substantial amount of classified lowercase letters available. It turns out that very few datasets exist making use of IMU sensors. At the same time, a large amount of data is available for 2D representation of letters. Some datasets consist of images representing letters (offline datasets), as seen in the EMNIST dataset introduced in [Cohen et al., 2017], while others consist of point lists along with the corresponding drawing times (online datasets), exemplified by the DigiLeTs dataset in [Fabi et al., 2023]. Each type of dataset offers its advantages; offline ones can be effectively processed by CNNs, and online ones are particularly useful in RNN architectures.

To leverage these 2D datasets effectively, it's crucial for the source and target datasets to exhibit some degree of similarity. Hence, the preprocessing stage is designed to transform the data from the IMU sensor into points within a 2D space and images.

4.2 Pre-processing

To convert data from the IMU sensor into a 2D representation, the first step involves using a Madgwick Filter to transform the data into Euler angles. By extracting the pitch and the yaw of this data as coordinates in a 2D space, it is possible to obtain a 2D representation of the drawn letter.

4.2.1 Data cleaning

Transfer learning yields better results when the target distribution is as close as possible to the source distribution. However, data from our experiment were acquired very differently than in the source datasets. For instance, the DigiLeTs dataset recorded the data by making use of a tablet, which resulted in a clean sequence of points of varying length. On the contrary, our dataset is composed of a fixed sequence of 32 points that tend to be noisy.

To counteract these differences, the data was pre-processed in the same way, by applying the following steps, inspired by [Dash et al., 2017]:

1. Points are scaled between 0 and 1 while respecting the aspect ratio to avoid distorting the letter. [Fig 1]
2. To filter out noisy points while retaining essential information, a slight smoothing was applied to the points using the following formula:

$$(x^t, y^t) = (\frac{x^{t-1} + x^t + x^{t+1}}{3}, \frac{y^{t-1} + y^t + y^{t+1}}{3})$$

3. If a point is too close to the previous one, it might lure the LSTM in the wrong direction. That's why points that were too close to each other in terms of the euclidean distance were removed. A higher threshold was applied to consecutive points that were closely spaced towards the end (in red). This adjustment aimed to eliminate as many points as possible occurring during the final stages of drawing a letter, where users might be waiting for the recording to finish. [Fig 2]
4. Finally the points were interpolated using a Cubic spline interpolation of degree 3. This compensated for the lack of points of certain recordings, by replacing a list of sequence segments by a round shape. This interpolation also allowed all the data to be the same length (100 samples), which further diminish the difference between data from the source dataset and the target dataset. [Fig 3]

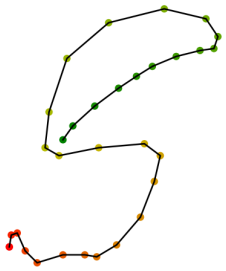

Figure 1: Raw points

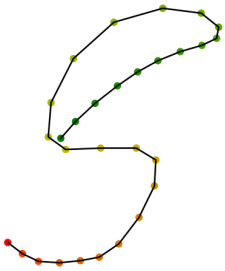

Figure 2: Cleaned and smoothed points

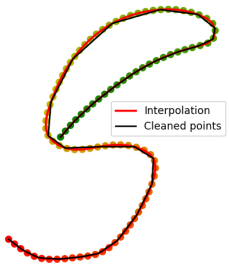

Figure 3: Interpolated points

4.2.2 Conversion to an image

Once a 2D représentation is extracted [Fig 4], it becomes possible to generate an image. Creating a grayscale representation of the letter involves drawing lines between the points, resulting in white pixels on a large canvas [Fig 5]. Subsequently, a Gaussian filter is applied to transform the image into grayscale values. Finally, the image is scaled down to the desired dimensions [Fig 6]. This technique allows the conversion from 2D points to an image while retaining as much information as possible.

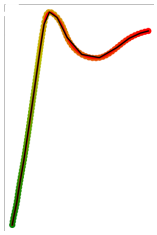

Figure 4: Interpolated letter

Figure 5: Image on a large canvas

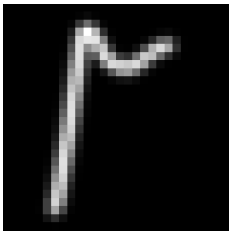

Figure 6: Scaled down image

5 Transfer learning using a CNN

5.1 Considerations

During the analysis of errors with CNN models, they tended to make more mistakes when dealing with letters that required the person to lift the pen. This discrepancy can be attributed to the experimental setup, where participants were not able to lift their pens. Conversely, the EMNIST dataset included data recorded with the option to lift the pen. This dissimilarity between the source and target datasets adversely affected the neural network's performance.

To address this issue, after 23 tries to explore various strategies, the source dataset was composed of 49 791 images from the EMNIST dataset and 10 009 images from the DigiLeTs dataset. Points from the DigiLeTs dataset underwent the same preprocessing as the target data which ensured consistent preprocessing between the target and DigiLeTs datasets and enables the drawing of a line connecting the points where the pen was raised and where it was set down again [Appendix B]. Incorporating the DigiLeTs dataset enriched the networks' ability to accurately recognize letters, particularly in cases where pen lifting occurs.

Moreover, in the absence of utilizing the EMNIST dataset, experiments were conducted with images of both 28×28 and 56×56 dimensions to enrich the information captured within the letters. Nonetheless, it was observed that downsizing to 28×28 following the application of Gaussian blur proved to be effective in retaining the majority of pertinent data. Indeed, the employment of 28×28 images surpassed its 56×56 counterpart in performance. Furthermore, considering embedded machine learning scenarios where computational resources and memory are limited, 28×28 images were deemed more suitable, hence chosen for evaluation purposes.

5.2 Model used

A total of 33 attempts were made to refine the model, experimenting with variations in layer numbers, striding, and dropout rates. Ultimately, the optimized model architecture consisted of two convolutional and max-pooling layers, followed by a flattening layer, a dense layer with 128 units, and a final dense layer comprising 26 units. [Appendix D].

This model is composed of 227,098 trainable parameters, which results in overfitting. Despite its susceptibility to overfitting, any effort to reduce the number of parameters or introduce regularization techniques (ex: normalization, striding, dropout, or layer removal) negatively impacts accuracy, significantly compromising performance on the target dataset.

In addition to the challenge of overfitting to the training data, the model also tend to learn to recognise specifically letters from the exact distribution of the training set. Consequently, a notable discrepancy in accuracy emerges between data from the validation dataset (composed solely of samples from the target dataset) a validation dataset sharing the same distribution as the training dataset (composed predominantly of data from the source dataset). [Appendix C]

To address this issue, a two-step training approach was implemented. At first, the model was trained exclusively on the source dataset. Then, it was fine-tuning on the target dataset. This two-step training strategy significantly enhanced the model's performance.

The experiment setup employs the categorical cross entropy loss function in conjunction with the Adam optimizer. Both the initial training phase with the source dataset and the subsequent fine-tuning phase utilized a learning rate of 0.001 and a batch size of 32. The training phase spanned 500 epochs, while the fine-tuning phase was conducted over 200 epochs. In both scenarios, to address time constraints and prevent overfitting, training ceased after 30 epochs if validation accuracy showed no improvement.

5.3 Results

5.3.1 Person dependent evaluation

In an effort to preserve the already limited available space data, a dataset size of 200 samples was chosen for both the development and testing datasets. However, this limited testing data quantity introduces certain biases into the results. To mitigate this effect, the reported performances are an average of five different splits in the data, with each split trained three times, resulting in a total of 15 unique training instances. This approach aims to provide a more balanced and less biased representation of the results.

Following this methodology on a user dependant setting, the CNN model obtained an accuracy of **82.46%**

The confusion matrix provided in Appendix F illustrates that the overall accuracy is negatively impacted by the dominance of the EMNIST dataset, which notably lacks instances where the pen is lifted. This deficiency is particularly evident in the recognition of letters such as 'i' and 'x', where only 66% of these letters are correctly identified. Furthermore, the matrix highlights the frequent misclassification of letters that bear a resemblance to each other, such as 'a' and 'q'.

5.3.2 Person independent evaluation

When presented with a new writing style, the model's accuracy demonstrates notable variability based on the specific type of letter drawn and the individual's handwriting style. To account for this variability, the reported performances are an average of the results obtained with one training for each of the 16 different individuals in the dataset. This approach provides a more comprehensive and representative evaluation, considering the diverse writing styles and patterns inherent in the dataset.

Following this methodology on a user independent setting, the CNN model obtained on accuracy of **69.26%**.

It is important to qualify the previous findings. As demonstrated in Appendix G, accuracy can vary significantly depending on the user and the specific letter drawn. For example, User 11 achieved a poor accuracy of 42% because, among its ten letters, it included 'i', 'a', and 'q', which are challenging to classify.

Furthermore, the confusion matrix highlights the model's struggle to generalize effectively, particularly for letters with multiple variations in writing styles, such as 't'.

6 Transfer learning using an LSTM

The LSTM model was trained and evaluated in a similar manner than what is explained in section 5.

6.1 Considerations

RNN models necessitate online datasets to be trained. Hence, to assess the model's performance, the source dataset contains 9500 examples from the DigiLeTs dataset. However, this relatively small dataset size negatively impacted the models' performance as certain handwriting styles were underrepresented. To address this issue, experiments were conducted using the BRUSH dataset [Kotani et al., 2020]. Unfortunately, the observed results did not show significant improvement compared to solely utilizing the DigiLeTs dataset. It is believed that the potential benefits of increased variety in writing styles were counteracted by the noise in the data linked to the fact that the letters from the BRUSH dataset are extracted from handwritten sentences. That's why the presented results were achieved using only the DigiLeTs dataset.

6.2 Model used

Following 14 iterations adjusting both the number of units and LSTM layers, the most effective configuration turned out to be a very simple one , composed of a bidirectional LSTM followed by a dense layer with 26 units [Appendix E].

The number of 50 units was carefully chosen to maximise the accuracy on a user independent setting. Its 23 826 parameters ensure the best performances while avoiding overfitting.

Following numerous tries, the experiment setup is as follows. It makes use of the categorical cross entropy loss function paired with the Adam optimizer. In the training phase with the source dataset, the model uses a learning rate of 0.01 and a batch size of 32 over 500 epochs. Fine-tuning occurred with a batch size of 64 and a learning rate of 0.01 over 200 epochs. Similarly to what was done in 5.2, an early stopping was implemented after 30 epochs for both the training and the fine tuning phase.

6.3 Results

6.3.1 Person dependent evaluation

This simple model obtained an average accuracy of **89.19%**.

The confusion matrix provided in Appendix H reveals that while the model achieves a good accuracy, its performance is limited by misclassifications of letters that bear resemblance, such as 'o' and 'i' when the pen is not lifted. This underscores the necessity for additional data to better differentiate between similar letters.

6.3.2 Person independent evaluation

By following the same methodology described in 5.3.2, the LSTM model obtained an average accuracy of **71.02%**.

Similar to the CNN in a user-independent setting, the accuracy of the LSTM varies significantly between users, as evidenced in Appendix I. Additionally, the confusion matrix highlights that the errors made by the LSTM differ from those made by the CNN.

7 Performance evaluation

The primary objective of this article was to improve the model's performance with respect to new writing styles. However, the ultimate goal is to deploy the model on the microcontroller within the smart glove.

To achieve this, the program must utilize less than the 520 kB of RAM. Unfortunately, experiments evaluating RAM usage [Appendix J] revealed that 309 MB were utilized to infer a letter with the LSTM and 364 MB with the CNN.

8 Conclusion

8.1 Model comparison

Model	Person dependant accuracy	Person independent accuracy
LSTM [Sec 6]	89.19%	71.02%
CNN [Sec 5]	82.46%	69.26%
LSTM [Zrira et al., 2024]	93%	55%
LSTM [Gaffarov, 2023]	80%	-
CNN [Gaffarov, 2023]	78%	-
Random forest [Zrira et al., 2024]	85.6%	-

Table 1: Comparison of different model on the person dependant and independent accuracy

As indicated Table 1, the transfer learning approach exhibits promising results, demonstrating significantly better generalization capabilities on new handwriting samples. Indeed, while the LSTM from [Zrira et al., 2024] exhibits slightly better accuracy in a person-dependent setting, the one presented in this paper notably enhances person-independent accuracy by 16%. This substantial improvement outweighs the 4% reduction observed in the person-dependent setting.

A notable observation between the two models discussed in this article is that the LSTM outperforms the CNN by 6.7% in the person-dependent setting and only by 1.76% in the person-independent setting. This discrepancy could be attributed to the limited diversity of data utilized to train the LSTM. Despite the low quality of the data, the CNN demonstrates superior generalization, largely due to the abundance of data. This highlights the potential for further improvement in the LSTM model through the acquisition of additional high-quality online datasets.

8.2 Future work

To mitigate memory requirements, the preprocessing could be rewritten in a low-level programming language, and the model size could be reduced by implementing tinyML techniques such as those presented in [Gaffarov, 2023].

The deficiency in diverse data has also been identified as a significant issue. Recording or leveraging new online datasets, along with experimenting with data augmentation techniques, could potentially enhance the classifier's performance.

Finally, akin to the approach outlined in [Dash et al., 2017], fusion strategies integrating various components such as CNN and LSTM could be employed to achieve significantly improved results.

References

[Chen et al., 2021] Chen, J. K., Xie, W., and He, Y. (2021). Motion-Based Handwriting Recognition. arXiv:2101.06022 [cs].

[Cohen et al., 2017] Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. arXiv:1702.05373 [cs].

[Dash et al., 2017] Dash, A., Sahu, A., Shringi, R., Gamboa, J., Afzal, M. Z., Malik, M. I., Dengel, A., and Ahmed, S. (2017). AirScript - Creating Documents in Air. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 908–913. ISSN: 2379-2140.

[Fabi et al., 2023] Fabi, S., Otte, S., Scholz, F., Wührer, J., Karlbauer, M., and Butz, M. V. (2023). Extending the Omniglot Challenge: Imitating Handwriting Styles on a New Sequential Data Set. *IEEE Transactions on Cognitive and Developmental Systems*, 15(2):896–903. Conference Name: IEEE Transactions on Cognitive and Developmental Systems.

[Gaffarov, 2023] Gaffarov, A. (2023). Design and Implementation of a Smart Glove with Embedded Handwritten Letter Recognition. Technical report, Mines Saint-Etienne, Institut Henri Fayol, Jean Monnet University.

[Kotani et al., 2020] Kotani, A., Tellex, S., and Tompkin, J. (2020). Generating handwriting via decoupled style descriptors. In *European Conference on Computer Vision*, pages 764–780. Springer.

[Luo et al., 2024] Luo, Y., Liu, C., Lee, Y. J., DelPreto, J., Wu, K., Foshey, M., Rus, D., Palacios, T., Li, Y., Torralba, A., and Matusik, W. (2024). Adaptive tactile interaction transfer via digitally embroidered smart gloves. *Nature Communications*, 15(1):868. Publisher: Nature Publishing Group.

[Zrira et al., 2024] Zrira, I., Bellanca, U., and Aharrar Soulali, M. (2024). Technical project report - Handwritten Letter Recognition. Technical report, Mines Saint-Etienne.

Appendices

A Sensor

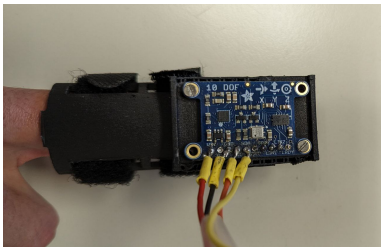

Figure 7: IMU sensor with its box and wires (top view)

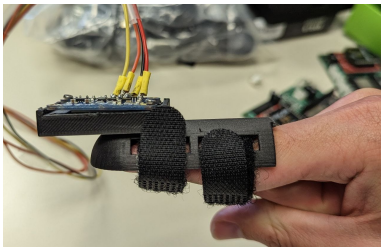

Figure 8: IMU sensor with its box and wires (front view)

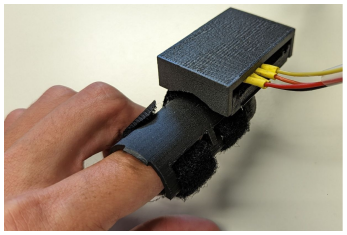

Figure 9: IMU sensor with its box, lid and wires

B Examples illustrating the letter "x" with and without the pen lifted

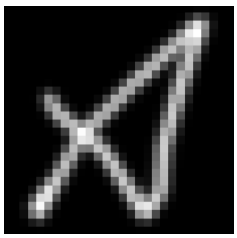

Figure 10: X from the sensor

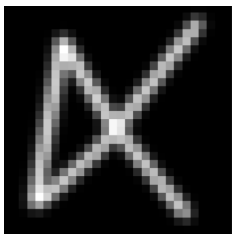

Figure 11: X from DigiLeTs

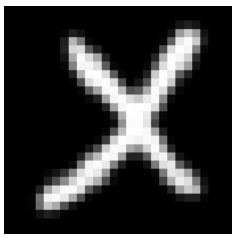

Figure 12: X from EMNIST

C CNN training exemple

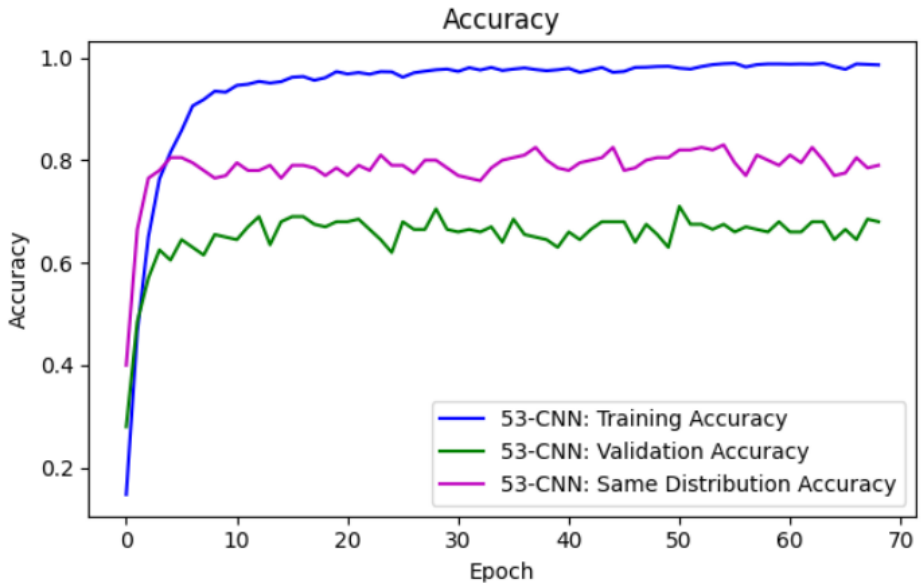

Figure 13: Example of a training of a CNN without utilizing fine-tuning

D Proposed CNN architecture

Layer	Output Shape	Layer Type	Kernel Size	Kernels	Activation
1	$(26, 26)$	Convolutional	$(3, 3)$	32	Relu
2	$(13, 13)$	Max Pooling	$(2, 2)$	32	-
3	$(11, 11)$	Convolutional	$(3, 3)$	64	Relu
4	$(5, 5)$	Max Pooling	$(2, 2)$	64	-
5	(1600)	Flatten	-	-	-
6	(128)	Dense	-	-	Relu
7	(26)	Dense	-	-	Softmax

Table 2: The proposed CNN architecture for classifying 26 lowercase letters of the English alphabet

E Proposed LSTM architecture

Layer	Output Shape	Layer Type	Units	Activation
1	(100)	Bidirectional LSTM	50	Tanh and Sigmoid
2	(26)	Dense	-	Softmax

Table 3: Proposed LSTM architecture for classifying 26 lowercase letters of the English alphabet

F Results CNN - Person dependant

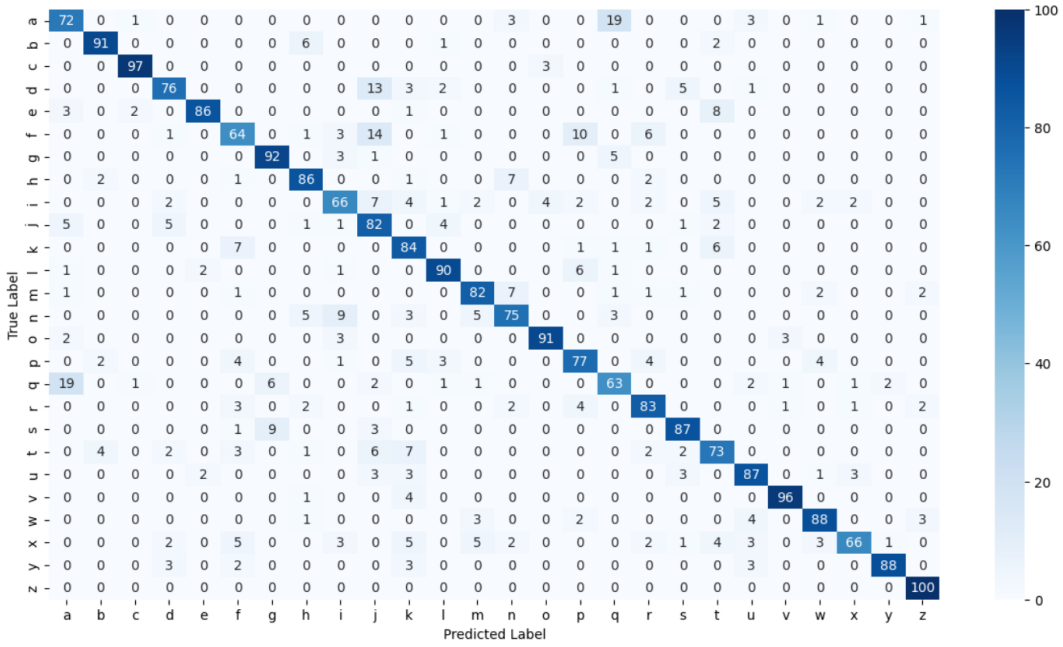

Figure 14: Confusion matrix for the CNN on person dependant setting (Rounded Percentage Values)

G Results CNN - Person independent

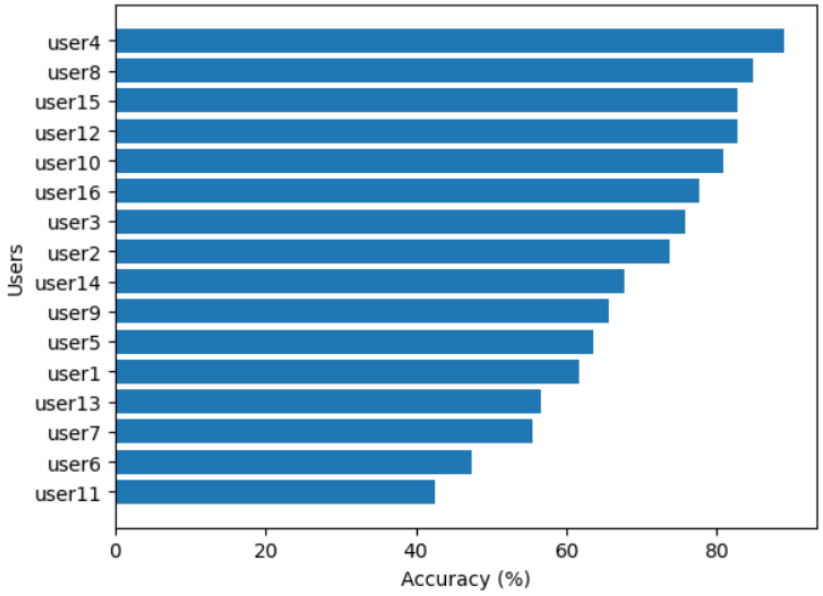

Figure 15: Accuracy per user using the CNN

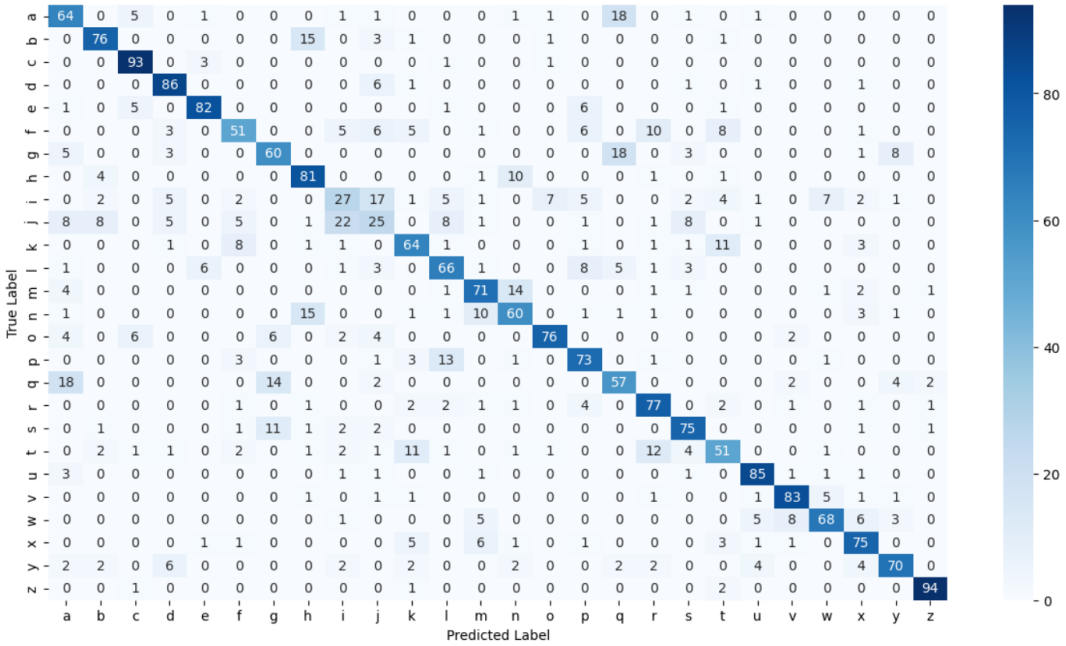

Figure 16: Confusion matrix for the CNN on person independent setting (Rounded Percentage Values)

H Results LSTM - Person dependant

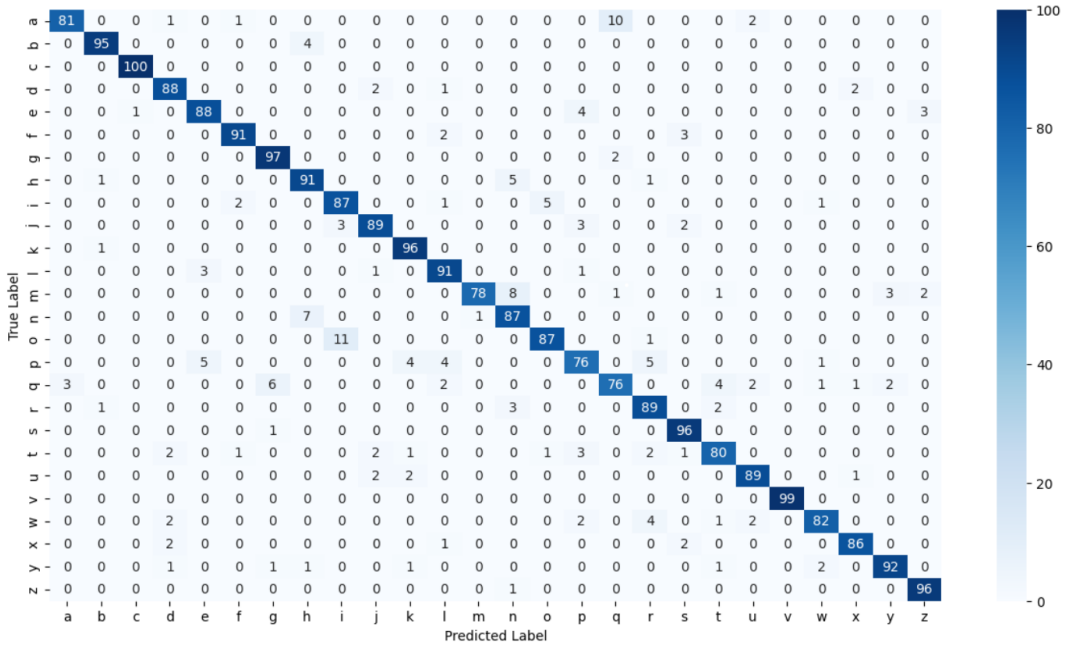

Figure 17: Confusion matrix for the LSTM on person dependant setting (Rounded Percentage Values)

I Results LSTM - Person independent

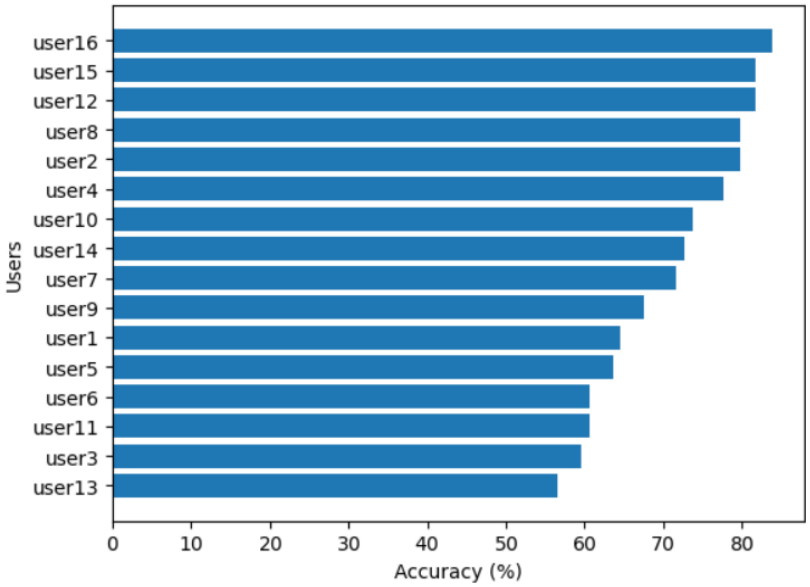

Figure 18: Accuracy per user using the LSTM

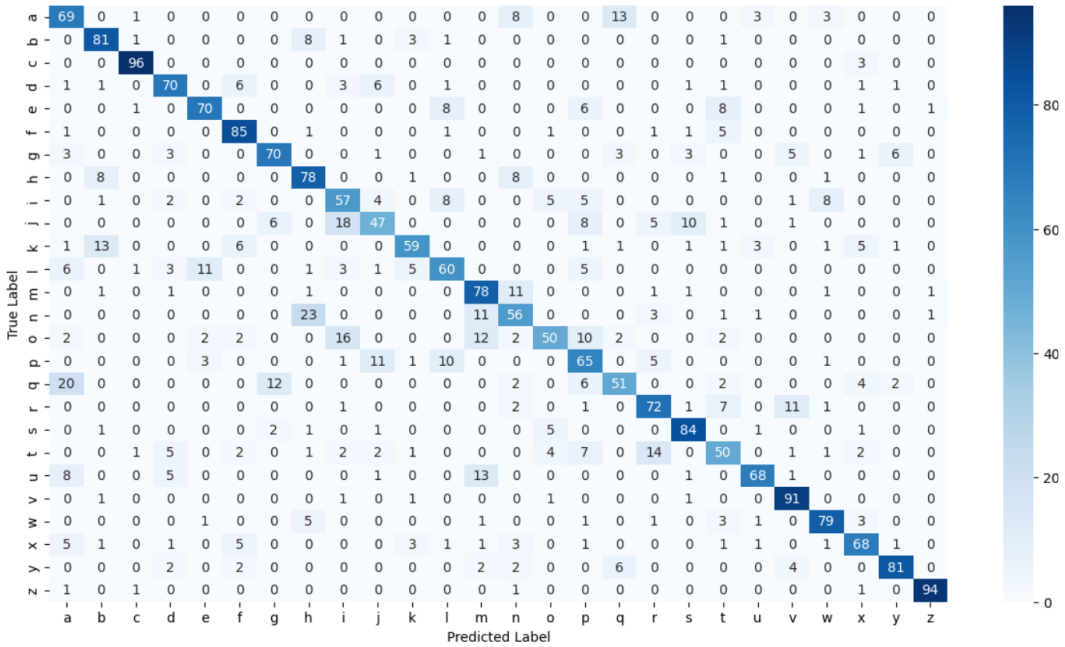

Figure 19: Confusion matrix for the LSTM on person independent setting (Rounded Percentage Values)

J Memory report for the inference of a letter

The full memory report of the LSTM can be found here and the one for the CNN can be found here.

Part of the memory report for the CNN can be found bellow:

```
Mem usage    Increment  Line
===================== ===============
 44.1 MiB    44.1 MiB  @profile
```

```
                    def main():
 55.5 MiB    11.4 MiB    from numpy import argmax,matmul,array,sin,cos,pi
 79.3 MiB    23.8 MiB    from pandas import read_csv
 81.2 MiB     2.0 MiB    from ahrs.filters import Madgwick
 81.2 MiB     0.0 MiB    from ahrs.common.orientation import q2rpy
 81.2 MiB     0.0 MiB    from sys import path
259.1 MiB   177.8 MiB    from keras.models import load_model


                        # go to the correct directory to import utils
259.1 MiB     0.0 MiB    folder_path = "C:/"
259.1 MiB     0.0 MiB    path.insert(1,f'{folder_path}Previous_work/Personal_Experiments/
    Extract_values')
275.2 MiB    16.2 MiB    from utils import convert_points_to_images, remove_close_points,
    smoothen_points,inter,scale_and_center

295.8 MiB     0.0 MiB    def rotation_matrix(theta):
295.8 MiB     0.0 MiB        return array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])

                        # Load the specified try
275.2 MiB     0.0 MiB    def load_try(try_name):
275.2 MiB     0.0 MiB        weight_folder_path = f'{folder_path}Previous_work/
    Personal_Experiments/ModelWeights'

                            # get the name of the model used for the try
275.2 MiB     0.0 MiB        with open(f'{weight_folder_path}/{try_name}/model_type.txt',
    newline='') as csvfile:
275.2 MiB     0.0 MiB            model_name = csvfile.readline()

                            # load the model
294.9 MiB    19.6 MiB        model = load_model(f'{weight_folder_path}/initialization/{
    model_name}.h5')

                            # load weights from the try
294.9 MiB     0.0 MiB        model.load_weights(f'{weight_folder_path}/{try_name}/data.h5')

294.9 MiB     0.0 MiB        print(f'Loading model {model_name} for try {try_name}')

294.9 MiB     0.0 MiB        return model


275.2 MiB     0.0 MiB    model_to_load = "57-CNN-FT-0"#"61-LSTM-FT-1"
275.2 MiB     0.0 MiB    model_LSTM = False

294.9 MiB     0.0 MiB    model = load_try(model_to_load)
295.3 MiB     0.4 MiB    df = read_csv(f'{folder_path}Previous_work/Personal_Experiments/
    Performance_evaluation/example_data.csv')

295.7 MiB     0.3 MiB    acc_data = df[['acc_x', 'acc_y', 'acc_z']].values
295.7 MiB     0.0 MiB    mag_data = df[['mag_x', 'mag_y', 'mag_z']].values
295.7 MiB     0.0 MiB    gyro_data = df[['gyr_x', 'gyr_y', 'gyr_z']].values

295.8 MiB     0.1 MiB    madgwick = Madgwick(acc=acc_data, gyr=gyro_data, mag=mag_data,
    frequency=27.0)
295.8 MiB     0.0 MiB    angles = [q2rpy(i, in_deg = False) for i in madgwick.Q] # x,y,z =
    roll, pitch, yaw
295.8 MiB     0.0 MiB    angles = array(angles)

                        # x= pitch; y= yaw
295.8 MiB     0.0 MiB    xy = angles[:,[2,1]]

                        # put letters upside down
```

```
295.8 MiB    0.0 MiB    xy_shift = array([matmul(rotation_matrix(pi),xyi) for xyi in xy])

                        # scale the values between 0 and 1 while keeping the aspect ratio +
                            center the number
295.8 MiB    0.0 MiB    xy_scaled = scale_and_center(xy_shift)

                        # do the average and remove redundant points
295.8 MiB    0.0 MiB    xy_smoothed = smoothen_points(xy_scaled,3)
295.8 MiB    0.0 MiB    xy_clean,mask = remove_close_points(xy_smoothed)

295.8 MiB    0.1 MiB    interpolation = inter(xy_clean,100,3)


295.8 MiB    0.0 MiB    if model_LSTM:
                            # shape correctly for the LSTM
                            interpolation = interpolation.transpose(1,0).reshape(1,100,2)
                            predicated_value = chr(argmax(model.predict(interpolation))+97)
                        else:
296.7 MiB    0.8 MiB        image = convert_points_to_images(interpolation.T,gaussian_kernel
    =21,final_dimension=28,initial_dimension=128,line_thickness=1.5,border_thickness=3)
296.7 MiB    0.0 MiB        image = image.reshape(1,28,28)
309.1 MiB   12.4 MiB        predicated_value = chr(argmax(model.predict(image))+97)


309.1 MiB    0.0 MiB    print(predicated_value)
```

Part of the memory report for the LSTM can be found bellow:

```
Mem usage   Increment Occurrences  Line Contents
=============================================================
44.0 MiB    44.0 MiB   @profile
                       def main():
55.8 MiB    11.8 MiB      from numpy import argmax,matmul,array,sin,cos,pi
79.4 MiB    23.5 MiB      from pandas import read_csv
81.0 MiB     1.6 MiB      from ahrs.filters import Madgwick
81.0 MiB     0.0 MiB      from ahrs.common.orientation import q2rpy
81.0 MiB     0.0 MiB      from sys import path
259.2 MiB   178.2 MiB      from keras.models import load_model


                                        # go to the correct directory to import utils
259.2 MiB    0.0 MiB      folder_path = "C:/"
259.2 MiB    0.0 MiB      path.insert(1,f'{folder_path}Previous_work/Personal_Experiments/
    Extract_values')
275.8 MiB   16.6 MiB      from utils import convert_points_to_images, remove_close_points,
    smoothen_points,inter,scale_and_center

339.4 MiB    0.0 MiB3      def rotation_matrix(theta):
339.4 MiB    0.0 MiB2          return array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)
    ]])


                                        # Load the specified try
275.8 MiB    0.0 MiB      def load_try(try_name):
275.8 MiB    0.0 MiB          weight_folder_path = f'{folder_path}Previous_work/
    Personal_Experiments/ModelWeights'

                                            # get the name of the model used for the try
275.8 MiB    0.0 MiB          with open(f'{weight_folder_path}/{try_name}/model_type.txt',
    newline='') as csvfile:
275.8 MiB    0.0 MiB              model_name = csvfile.readline()

                                            # load the model
```

```
338.5 MiB    62.7 MiB         model = load_model(f'{weight_folder_path}/initialization/{
    model_name}.h5')

                                      # load weights from the try
338.6 MiB     0.1 MiB         model.load_weights(f'{weight_folder_path}/{try_name}/data.h5')

338.6 MiB     0.0 MiB         print(f'Loading model {model_name} for try {try_name}')

338.6 MiB     0.0 MiB         return model


275.8 MiB     0.0 MiB     model_to_load = "61-LSTM-FT-1"#"57-CNN-FT-0"
275.8 MiB     0.0 MiB     model_LSTM = True

338.6 MiB     0.0 MiB     model = load_try(model_to_load)
339.0 MiB     0.4 MiB     df = read_csv(f'{folder_path}Previous_work/Personal_Experiments/
    Performance_evaluation/example_data.csv')

339.3 MiB     0.3 MiB     acc_data = df[['acc_x', 'acc_y', 'acc_z']].values
339.3 MiB     0.0 MiB     mag_data = df[['mag_x', 'mag_y', 'mag_z']].values
339.3 MiB     0.0 MiB     gyro_data = df[['gyr_x', 'gyr_y', 'gyr_z']].values

339.4 MiB     0.1 MiB     madgwick = Madgwick(acc=acc_data, gyr=gyro_data, mag=mag_data,
    frequency=27.0)
339.4 MiB     0.0 MiB5     angles = [q2rpy(i, in_deg = False) for i in madgwick.Q] # x,y,z =
    roll, pitch, yaw
339.4 MiB     0.0 MiB     angles = array(angles)

                                  # x= pitch; y= yaw
339.4 MiB     0.0 MiB     xy = angles[:,[2,1]]

                                  # put letters upside down
339.4 MiB     0.0 MiB5     xy_shift = array([matmul(rotation_matrix(pi),xyi) for xyi in xy])

                                  # scale the values between 0 and 1 while keeping the
                                       aspect ratio + center the number
339.4 MiB     0.0 MiB     xy_scaled = scale_and_center(xy_shift)

                                  # do the average and remove redundant points
339.4 MiB     0.0 MiB     xy_smoothed = smoothen_points(xy_scaled,3)
339.4 MiB     0.0 MiB     xy_clean,mask = remove_close_points(xy_smoothed)

339.5 MiB     0.1 MiB     interpolation = inter(xy_clean,100,3)


339.5 MiB     0.0 MiB     if model_LSTM:
                                  # shape correctly for the LSTM
339.5 MiB     0.0 MiB         interpolation = interpolation.transpose(1,0).reshape(1,100,2)
364.0 MiB    24.5 MiB         predicated_value = chr(argmax(model.predict(interpolation))+97)
                              else:
                                  image = convert_points_to_images(interpolation.T,
                                      gaussian_kernel=21,final_dimension=28,
                                      initial_dimension=128,line_thickness=1.5,
                                      border_thickness=3)
                                  image = image.reshape(1,28,28)
                                  predicated_value = chr(argmax(model.predict(image
                                      ))+97)


364.0 MiB     0.0 MiB     print(predicated_value)
```