MACHINE LEARNING CHALLENGE 2022

ML Challenge Group 21

Group Members:

Ali Burak Yuksel	*Ela Guven*	*Konrad Hein*	*Edoardo Saldarelli*

PART 1 - Feature Engineering

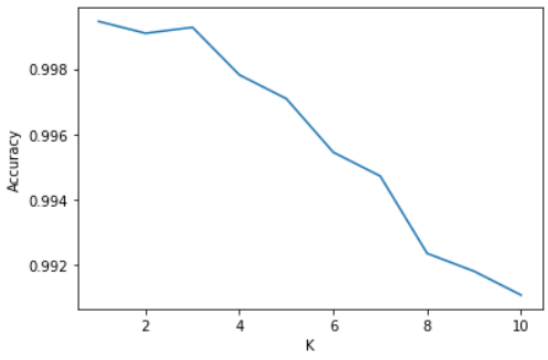
In common agreement among the group members, we decided to split the entire training dataset into training and validation subsets with a proportion of **0.8 and 0.2** respectively. In the training data all classes occurred between **957 and 1294** times, so we assumed balanced classes and, consecutively, we used macro accuracy. Afterwards, the modelling strategy we implemented (with KNN model) has led us to consider first the relevance of engineering some features of the datasets, but we concluded that this step had to be avoided for the following reason:

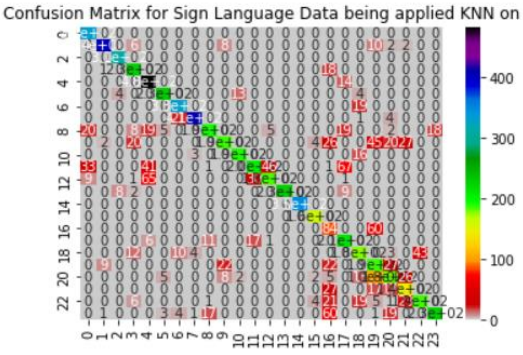
1) For modelling with KNN, our purpose was mainly to look for the distance between pixels, and hence transforming / extracting pixel values did not define our analysis' framework. Moreover, we were concerned about the repercussions of removing noisy images from the training subset, expecting that the variance of the data could drastically change. Therefore, the Feature Engineering step for KNN was not within our project's scope.
2) Conversely, for modelling with CNN we relied on the CNN feature extractor capabilities, which allows for automatic extraction of the features in the training process, rather than implementing it manually. For the CNN, in the preprocessing part, we used *LabelBinarizer* method to transform the labels which were *Pandas* object to *Numpy* arrays.
   After reshaping the images into the size 28x28x1 from 1D to 3D, data was ready as input for CNN model.

PART 2 – Learning Models and Algorithms

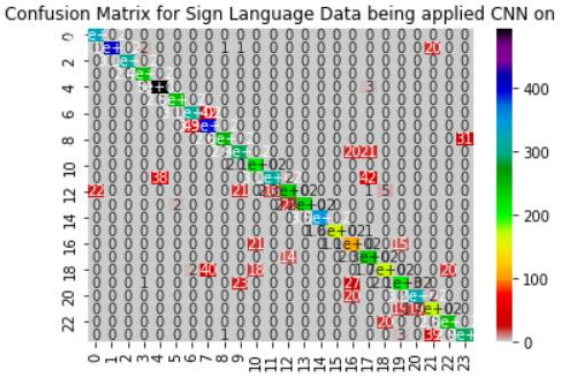
After coding and evaluating of the performance metrics of the models chosen on the training set, we could see that **KNN** and **CNN** models' accuracy scores showed robust performance. Additionally, we trained both final models on the joined training and validation set. Hence, we decided to select these 2 models as part of Task 1. Specifically, our decision was driven by the following factors:

1) KNN was chosen because of its solid performance over the test set. Below is the accuracy plot for the KNN hyperparameter tuning, with out-of-sample-validation of K=1 being the best pick and its related confusion matrix. The accuracy value on the validation set is **0.9995**, while the final accuracy for KNN is **0.8104.** Plus, KNN locates all data points in a multidimensional space, in our case 784-dimensional, and calculates the distance from a new input to every data point. Then, it chooses the best distance based on the majority vote of the k nearest neighbors.





2) Traditionally, NNs can be seen as an advanced machine learning technique because of their high capability of finding important features. Additionally, CNN is a type of deep learning model for processing data with a gridded pattern (such as images) which is inspired by the structure of the brain's visual cortex and designed to learn spatial hierarchies of features automatically and adaptively, from low- to high-level patterns. We, therefore, decided to use a CNN because it is especially designed for images and can leverage the spatial dependencies for predictions. Indeed, the final accuracy for CNN on the test set is **0.9078**.



By comparing the two final confusion matrices, we could also clearly see that both models struggle with classes **16** and **12**.

PART 3 – Parameters Tuning

We decided to tune hyperparameters for our two models in the following way:

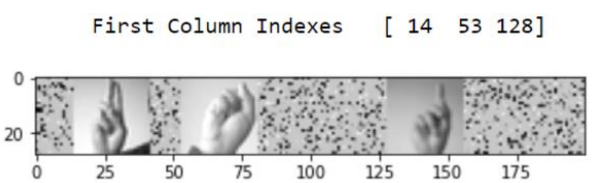
1) For KNN (as shown above) the hyperparameter tuning process was performed by simply selecting the best K-value, which resulted to be K=1 with an accuracy score of **0.8104.**
2) For CNN, we used *Keras* which is one of the Python libraries used to make fast computations and **Grid-Search** with different batch sizes and different numbers of epochs with 1 and 2 layers, as shown in the summary table below. As a result, we recorded the highest test-accuracy score for **1 CNN layer, batch-size=512, number of epochs=5, final score=0.9078.** All combinations reached a validation-accuracy score of 1.0 and, therefore, we decided based on the shortest training time.

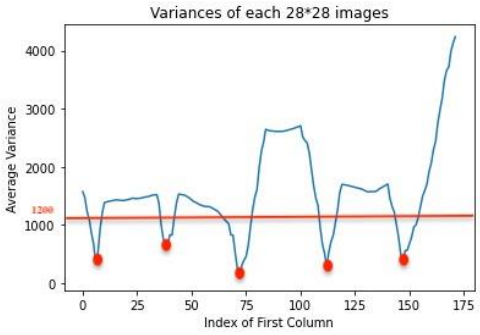
	1 Convolutional Layer			*2 Convolutional Layers*		
# Batch	**5 epochs**	**10 epochs**	**15 epochs**	**5 epochs**	**10 epochs**	**15 epochs**
Batch size 128	Acc: 1.0 Time: 105.72s	Acc: 1.0 Time: 210.38s	Acc: 1.0 Time: 312.54s	Acc: 1.0 Time: 133.25s	Acc: 1.0 Time: 262.02s	Acc: 1.0 Time: 390.54s
Batch size 256	Acc: 1.0 Time: 103.19s	Acc: 1.0 Time: 200.07s	Acc: 1.0 Time: 301.77s	Acc: 1.0 Time: 129.06s	Acc: 1.0 Time: 256.72s	Acc: 1.0 Time: 383.12s
Batch size 512	Acc: 1.0 Time: 101.38s	Acc: 1.0 Time: 203.01s	Acc: 1.0 Time: 306.26	Acc: 1.0 Time: 128.10s	Acc: 1.0 Time: 261.28s	Acc: 1.0 Time: 380.10s

PART 4 – Performance of the Solution

For Task 2 we implemented two different image detection models, because one of them allowed us to extract and predict the indexes of sign images. The other method helped us to get information from one image without considering the overlapping areas. Hence, using these two methods helped us to classify different image specifications. These are the functions we implemented[1]:

1) **find_indexes**: function that locates the left edges of the images and uses a variance through the images to find the indexes for detecting images correctly 95% of the times and detects all the images with a threshold value of 1200 (see plots below):





[1] The prediction functions have a fallback-mechanism for which they add an arbitrary prediction (‘***05000811’; “FAIL” encoded in the targets***) when they do not detect any images. By doing so, we avoided the occurrence of an error when making all predictions that would crash our final file.

2) **im_detect_cnn** and **im_detect_knn**: these two functions get as input a single 28*200 image and run the **find_indexes** function. They then create an array that contains the 28*28 images for all found indices (which flattens all of them to be able to use only the KNN). At the end, they make predictions using the KNN for single-digit classes, therefore the leading 0 is added (for example, 8 becomes “08”).

3) **im_detect_knn**: function that gets as input a single 28*200 image and runs the **find_indexes** function. It then creates an array that contains the 28*28 images for all found indices and it flattens all of them to be able to use KNN model. Finally, it makes predictions using the KNN for single-digit classes, therefore the leading 0 is added (for example, 8 becomes “08”).

4) **proba_pred**: function that classifies each 28x28 area of the sequence which has 172 images per sequence. If the probability of the predictions has a probability above the threshold using CNN, then the function keeps the prediction and skips the next 27 pixels because it assumes that they are in the same image. The function also uses image detection to separate the images contained in an input-image and then predicts each image using the CNN from task 1, but due to the encoding of the target, the labels obtained with argmax do not pay respect to classes 9 and 25.

Overall, we made two predictions with the first image detection, one with CNN and one with KNN, and three predictions with the second method by changing the threshold for accepting predictions. For that purpose, we took the input as a single big 28 * 200 image. Then, we made predictions for every single 28*28 image within the large image. While doing this, we stored the predicted class and its probability. Finally, we looped over these predictions. If the probability matched with a pre-defined threshold (0.99), then that prediction was kept for making the final prediction. In that case, the function initializes a recursive function that calls the same function 28 pixels further to the right, to “skip” the image. Additionally, the predicted probabilities are high and there would still be a difference between 0.99, 0.9999, and 1.0 as thresholds. So, we opted for choosing them high, else we may end up recording too many wrong predictions. Finally, it is important to point out that the two methods we used for Task 2 cover the weaknesses of each other, and that is the reason we decided to use both

PART 5 – Roles of the group members

All members of the group have proposed, debated, tested and implemented several machine learning algorithms by splitting the group into 2 sub-teams for both Task 1 and Task 2, following the below logic:

Task 1:

1) ***Ali Burak Yuksel & Edoardo Saldarelli (Team_1)***: investigation and implementation of KNN, Perceptron, Decision Tree and Logistic Regression models and outputs.
2) ***Ela Guven & Konrad Hein (Team _2)***: investigation and implementation of CNN model and output.

Task 2:

1) ***Ali Burak Yuksel & Konrad Hein (Team_1)***: coding development and commenting for image detection methods and predictions. Before and after the coding implementations, all the members contributed to merge the models and functions together within the same strategic framework, rather than keep them fragmented.
2) ***Ela Guven & Edoardo Saldarelli (Team_2)***: mutual collaboration in building the report and understanding the processes, techniques and concepts developed by Team_1. At the end, the report has been discussed and approved by the whole group.

References:

- https://www.kaggle.com/code/madz2000/cnn-using-keras-100-accuracy
- https://www.tensorflow.org/tutorials/images/cnn
- https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594
- https://analyticsindiamag.com/hands-on-guide-to-sign-language-classification-using-cnn/