June 27, 2020

Machine Learning

Machine Learning Assignment



Contents

[General Information 2](#_Toc43322171)

[Introduction 3](#_Toc43322172)

[Task 1 3](#_Toc43322173)

[Features & Preprocessing 3](#_Toc43322174)

[Learning Model, Algorithm and Method used 3](#_Toc43322175)

[Parameter Tuning 4](#_Toc43322176)

[Discussion of the results 4](#_Toc43322177)

[Task 2 6](#_Toc43322178)

[Features used 6](#_Toc43322179)

[Parameter tuning 6](#_Toc43322180)

[description of the method / system 6](#_Toc43322181)

[discussion of the experiments run and performance of the solution (confusion matrix, accuracy etc.) 6](#_Toc43322182)

[References 7](#_Toc43322183)

[Appendices 8](#_Toc43322184)

# General Information

**Group 40**

**Name of all group members:**

Dovile Kliusovaite Anr: 402518 Snr: 2046983

Meret Schädler Anr.: 2050924 Snr: 748225

**Tasks allocation:**

Dovile Kliusovaite Task 2

Meret Schädler Task 1

# Introduction

The purpose of this challenge is to classify and predict the handwritten letters of a subset of the EMNIST dataset which is an extension of the widely known MNIST dataset.

It is split into two tasks:

Task 1: “Train a model to recognize handwritten English letter”

Task 2: “Use the classifier trained in the first task to identify a series of 5 letters”

The dataset contains images of handwritten letter (upper and lowercases) of the whole English alphabet and the corresponding label. For example, if the image shows a “W”, the label would be 23 and if we see a “K” it would be 11 (see Appendix figure 1). Since the dataset covers the whole alphabet, we have 26 classes (A to Z). Thus, this is a multiclass prediction problem which can be solved with different linear and non-linear algorithms such as logistic regression, random forests, SGD classifier, multi-layer perceptron classifier or with a neural network. To calculate the accuracy of the different models and compare these scores, we use two packages “sklearn” or “keras” from Python.

The dataset used for this task is not the original EMNIST dataset. It only contains 124’000 images and labels equally distributed over the 26 letters (see Appendix figure 2). Moreover, the shape of the input data has already been adjusted in the dataset we used. Originally, each picture had to be converted to a 28x28 size image in a gray-scale of 0 – 255.

Our goal is to find the best predicting model to recognize the handwritten digits. A lot of models have been built to predict digits from the MNIST dataset which show an accuracy rate of close to 100%. However, the EMNIST dataset has not been explored to this extend yet. A good reference is the paper from Cohen, Afshar, Tapson and van Schaik (2017) which states an accuracy score of 55.78% for the letters of the EMNIST dataset when using a linear classifier and 85.15% +/- 0.12% for the OPIUM classifier (neural network).

# Task 1

## Features & Preprocessing

As mentioned in the introduction, the size of each images has been normalized to 28 x 28 what equals 784 pixels. These 784 pixels represent the features of the digits. In figure 1 (see appendix) it can clearly be seen, that the grey-scale ranges from 0 (black) to 255 (white). To use this input features in the models, we normalized the range so that each pixel can only have a value between 0 and 1.

Before we could start training the model, we had to change the labels into binary classes by applying the “LabelBinarizer” from “sklearn”. What this does is changing the 26 categories to a binary matrix. This is needed if we want to train a multi-layer perceptron in “keras”.

## Learning Model, Algorithm and Method used

For all models, we firstly split the 124’000 images into a training and a test set based on the commonly used splitting rule of 80% (train) versus 20% (test).

We trained the data on the following linear and non-linear models by using the “sklear” and “keras” library.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Model** | **Library** | **Parameters1** |
| Linear | Logistic Regression | sklearn | multi\_class = “Multinominal”  C = 10 |
| SGD Classifier | sklearn | loss='log' |
| Non-linear | Random Forest Classifier | sklearn |  |
| Multi-Layer Perceptron | sklearn | hidden\_layer\_sizes=(128,) |
| MLP Base | keras | See text below |
| Convolutional Neural Network | keras |  |

1 Only mentioned if it deviates from the default

For the multi-layer perceptron model, we firstly built a base model in “keras”. Based on this model we did some parameter tuning what will be describe in the next chapter. We decided to use a sequential model with two hidden layers. The input dimension for the first layer is 784 (features), for the activation we used “relu” and the output dimension is 256, for the second layer, we only changed the output dimension to 128 and the last layer has an output dimension of 26 (number of letters) and uses the “softmax” activation function which is used for multiclass classification problems. To compile the model, we used the “adam” optimizer with a learning rate of 0.001 and the loss “categorical\_crossentropy”. To start with, we ran the model with a batch size of 128 for 5 epochs.

Moreover, for the two multi-layer perceptron algorithms we did a 5-fold cross validation what helps us to avoid overfitting and finding a good set of parameters.

## Parameter Tuning

After we ran all these algorithms and compared the test accuracy, we decided to only tune the parameters of the MLP models.

For the Multi-Layer Perceptron models, we adjusted XX, while for the MLP Base model built in “keras” we made the following adjustments:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Output layer** | **Activation** | **Optimizer** | **Batch size** | **Epochs** |
| First | 256 | tanh, relu |  |  |  |
| Second | 128 |  |  |  |
| Output | 26 | softmax | adam, sgd |  |  |
| compile |  |  |  | 32, 64, 128 | 1, 5, 10 |

The results show, that the model with the highest test accuracy score of 88.78% has the following parameters: “relu”, “adam”, 10 epochs, batch size of 128 and the lowest loss is 0.369 of the model with the following parameters: “tanh”, “adam”, 10 epochs, batch size of 128.

## Discussion of the results

To evaluate the results, we used the accuracy score metric. Since the classes are balanced as mentioned in the introduction, we can use the accuracy score to compare and evaluate the models.

Firstly, we want to compare our results with those for the paper written by Cohen, Afshar, Tapson and van Schaik in 2017 to get an idea if our results are in the same range. The accuracy scores in the table below show higher scores for both the linear classifier as well as the non-linear classifiers.

|  |  |  |
| --- | --- | --- |
| **Model** | **Train accuracy** | **Val accuracy** |
| Logistic Regression | 70.99% | N/A |
| SGD Classifier | 61.91% | N/A |
| Random Forest Classifier | 80.50% | N/A |
| Multi-Layer Perceptron | 86.2% | TBC |
| MLP Base (epoche = 4) | Epoch = 2: 85.19% | 85.66% |
| * Variation 1 |  |  |
|  |  |  |
|  |  |  |
| Convolutional Neural Network |  |  |

Accuracy per class

Looking at the Confusion Matrix (see Appendix figure 4) we can identify a few problematic letters such as XX

# Task 2

In the second task, best performing classifier from task 1 is used on a test dataset containing images of 5 letter sequences with size 30x168. As in the first task, dataset images contain handwritten letters (upper and lowercases) of the whole English alphabet. However, there are no corresponding labels and images in the new dataset are noisy. Some of the images contain overlapping letters, Figure 3. The aim of task 2, is to generate 5 label predictions per each image. Labels should correspond to 26 classes (A to Z), concatenated together, e.g. if the letters in the image are ‘abcde’ the true label is = 0102030405.

## Packages used

We used OpenCv (Open Source Computer Vision Library) library to work with the images. OpenCV library is designed to solve computer vision problems, such as noise removal, boundary detection, intersection of images and my more (OpenCv team, 2020).

## Noise

First, we imported image in BGR scale, then we transformed original image to gray scale using cv2.cvtColor function and COLOR\_BGR2GRAY color space (TECHTUTORIALSX, 2018). Gray scaled images are easier to recognize the contours within an image.

Next step is to apply thresholding to our images, to separate values into two categories 0 (white) and to 255 (black) (Image Thresholding, 2020). To choose the optimal threshold we use Otsu’s Binarization algorithm (Nelli, 2017).

Thirdly, we use Morphological Operations to remove left noise and imperfections in the image. Morphological Operations rely on the ordering of pixel values, a small matrix (structured element) is fitted to the image at all locations and calculates the corresponding neighborhood of pixels. We used two Morphological Operations:

* Dilation – adds a layer of pixels to the inner and outer layer of boundary. Thus, value of the output pixel is maximum value of pixels in the neighborhood.
* Erosion – minuses a layer of pixels to the intersections. Is opposite of dilution and act as local minimum (Joram, 2020).

As the final step of noise removal, we inverted the image, so black becomes white and white becomes black. Color inversion is done by subtracting RGB color value form the maximum possible value (255). We decided to invert images to be able better recognize image boundaries (Loch, 2011).

## Boundary estimation

After images have been pre-processed, we extracted boundaries. Firstly, small edges are discounted, then ?????

## Intersection of letters

## Fitting the model

In order to fit the model created in task 1, we needed to resize the images. We started by extracting separate letters from the original image with a 2-pixel margin around the edges, thus from original 30x168 image, we have 5 different size images. Then we reshaped separated letter images into 28x28 size image.

# Conclusion

# References

<https://www.machinecurve.com/index.php/2020/02/18/how-to-use-k-fold-cross-validation-with-keras/>

<https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/>

<https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/>

<https://scikit-learn.org/stable/modules/cross_validation.html>

Hands-On Machine Learning Book

<https://www.kaggle.com/darkside92/simple-best-digit-recognizer-with-cnn-top-5/comments#497794>

<https://medium.com/@mjbhobe/mnist-digits-classification-with-keras-ed6c2374bd0e>

<https://www.codementor.io/@mgalarny/making-your-first-machine-learning-classifier-in-scikit-learn-python-db7d7iqdh>

<https://thedatafrog.com/en/articles/handwritten-digit-recognition-scikit-learn/>

<https://hub.packtpub.com/how-to-classify-digits-using-keras-and-tensorflow/>

<https://www.kaggle.com/raahat98/digit-classification-with-keras-99-accuracy>

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik, EMNIST: an extension of MNIST to handwritten letter in: International Joint Conference on Neural Networks (IJCNN), 2017

Alsaafin, A. and Elnagar, A. (2017) A Minimal Subset of Features Using Feature Selection for Handwritten Digit Recognition. Journal of Intelligent Learning Systems and Applications, 9, 55-68.

# Bibliography

OpenCv team. (2020, 06 16). *About*. Retrieved from opencv.org: https://opencv.org/about/

Singhal, G. (2020, February 11). *Importing Image Data into NumPy Arrays*. Retrieved from Pluralsight: https://www.pluralsight.com/guides/importing-image-data-into-numpy-arrays

TECHTUTORIALSX. (2018, June 2). *Python OpenCV: Converting an image to gray scale*. Retrieved from techtutorialsx.com: https://techtutorialsx.com/2018/06/02/python-opencv-converting-an-image-to-gray-scale/

Nelli, F. (2017, April 13). *OpenCV & Python – The Otsu’s Binarization for thresholding*. Retrieved from https://www.meccanismocomplesso.org/: https://www.meccanismocomplesso.org/en/opencv-python-the-otsus-binarization-for-thresholding/

Joram, N. (2020, January 1). *Morphological Operations in Image Processing*. Retrieved from https://medium.com/: https://medium.com/@himnickson/morphological-operations-in-image-processing-cb8045b98fcc

*Image Thresholding*. (2020, June 17). Retrieved from https://opencv-python-tutroals.readthedocs.io: https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_thresholding/py\_thresholding.html

Loch, F. G. (2011). *IMAGE PROCESSING ALGORITHMS PART 7: COLOUR INVERSION AND SOLARISATION*. Retrieved from https://www.dfstudios.co.uk: https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-7-colour-inversion-solarisation/

# Appendices

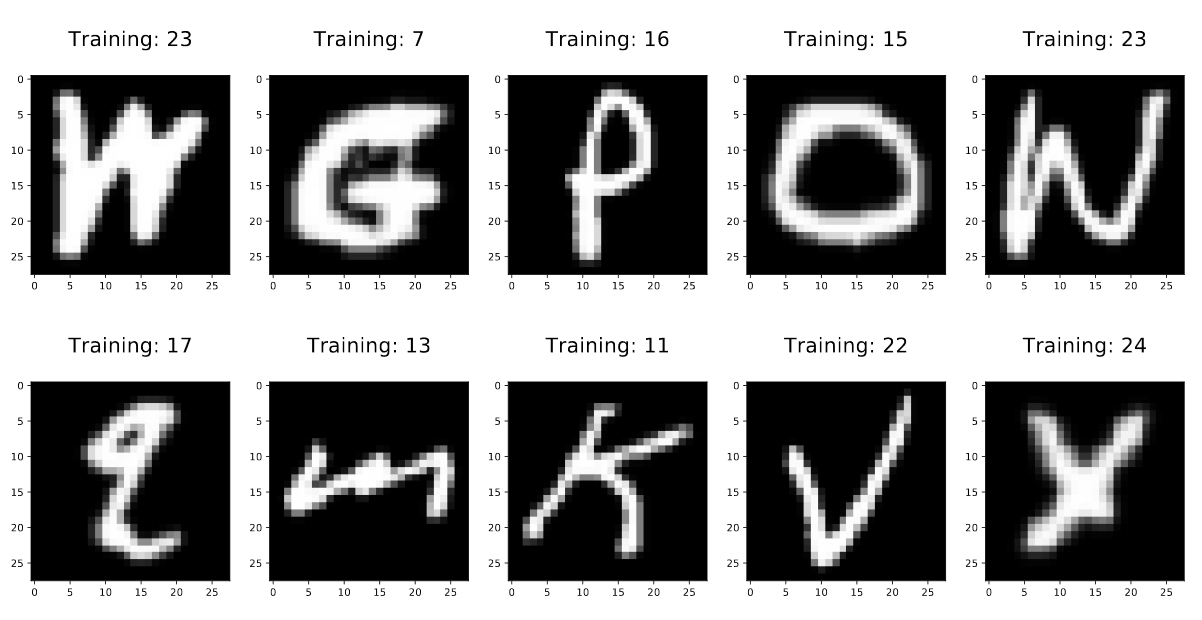


Figure 1 XX

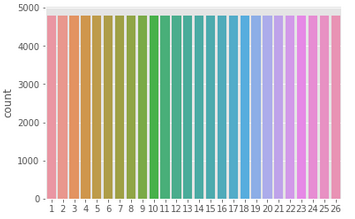


Figure 2 XX

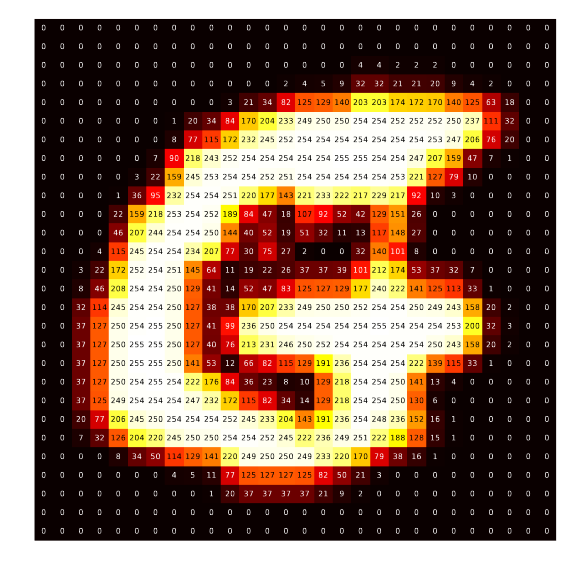


Figure 3 XX

A picture containing monitor, clock, meter

Description automatically generated