Classifier for handwritten letters & digits recognition

Key take-aways

1. A Convolution Neural Network classifier of test accuracy 95.3% has been trained
2. The vast majority of errors made by the classifier on unseen data is made during classification of characters of almost-identical shape:
    - Letter “i” (capital or without the tittle) vs digit “1” (without the diagonal verge)
    - Letter “O” vs digit “0”
    - Other
3. The following technologies and techniques have been used during the development & training
    - Python + sklearn, TensorFlow and Keras libraries
    - Multilayer perceptron classifier (sklearn)
    - Convolutional neural network (TensorFlow GPU + Keras)
    - Data pre-processing
    - Hyperparameters optimization (grid and random)
    - Regularization
    - Cross validation
    - Gradient Descent optimization (adam and rmsprop)
    - Class balancing
    - Loss function with class weights
    - Image augmentation (rotation and shift)
    - Dropout
    - Learning rate reduction

The task: basing on the input data of 30134 labelled images of letters and digits to train a classifier that will recognize and classify handwritten characters.

Data pre-processing

The initial step has been to load the data sample and examine it for completeness and quality. Visual representation of the data is displayed by plotting the reshaped vectors on figure 1.

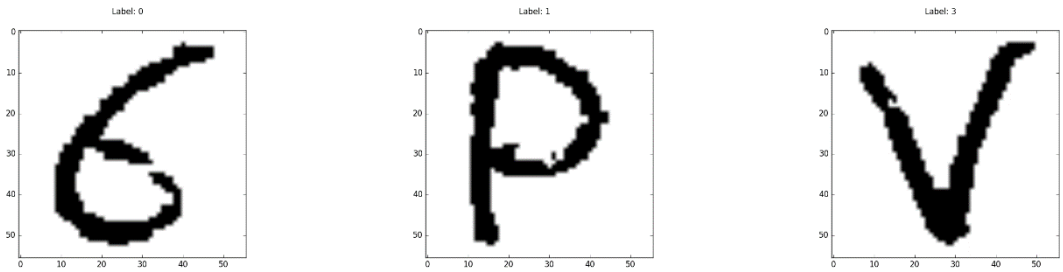

Figure 1 Exemplary handwritten characters from the dataset

The images are neat with little noise. The characters are of similar size, aligned to the center and with little rotation around the central point. The labels are integers from 1 to 36 (10 digits and 26 letters). The order of the labels is not maintained (letters and digits are mixed). This is not an issue for the classifiers, only a matter of aesthetics.

Next, the distribution of classes and their balance has been examined (see Figure 2.)

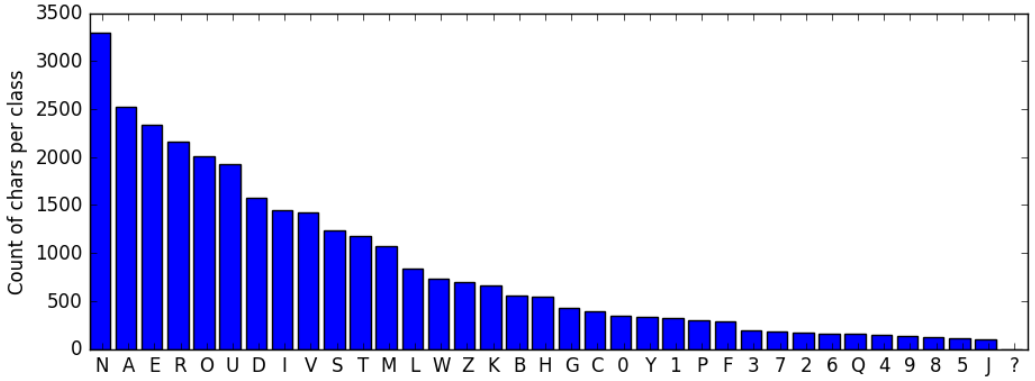

Figure 2 Distribution of data points per class

Two main conclusions have arised:

1. The classes are strongly imbalanced – class for letter ‘N’ consists of over 3000 examples, while classes like ‘8’, ‘5’, ‘J’ consist of less than 200 examples each. The training process might require adjustments for class balancing.
2. One of the classes (labelled here ‘?’) consists of only one entry. This one entry represents letter ‘N’, which already has its own class. The class for letter ‘X’ is missing though. The erroneous example has been dropped.

The data did not need normalization – all pixel values were binary, so already close to 0 and variance less than 1.

Finally, the data has been split into training and test sets (80:20 ratio) and saved for benchmarking of the models. This way, the estimation of generalization accuracy of all trained models could be assessed on the same training dataset.

The next sections focus on training two types classifiers. First, a simpler solution of a multilayer perceptron classifier (MLP). Then, a more complex one: convolutional neural network (CNN).

Multilayer Perceptron

The ‘basic’ neural network, a multilayer perceptron classifier, takes a vector as an input and outputs the predicted class’ label/number (or the probabilities for each class). This means that the information about proximity of pixels on the image is dropped and the input 2D image is flattened to a 1D vector.

The architecture of the classifier (number of hidden layers and number of neurons in each of them) as well as parameters such as regularization term, adam optimizer’s parameters β_1, β_2, ϵ, the learning rate were the questions to be answered by the hyperparameter optimization. In this exercise, the grid search approach has been adapted for the following space of parameters:

- $m \in \{1,2,3,4,5\}$ – the number of hidden layers
- $M \in \{100, 200, 300, 400\}$ – the number of units per hidden layer (equal between the layers)
- $\lambda \in \{10^{-2}, 10^{-1}, 1, 10\}$ – the regularization term
- $l_r = 10^{-3}$ – the initial learning rate
- $\beta_1 = 0.9;\ \beta_2 = 0.999, \epsilon = 10^{-8}$ – adam optimizer parameters
- Activation function: $\tanh(x)$
- No class balancing (upsampling and downsampling did not yield satisfactory gain)

Some of the parameters were fixed to arbitrary values for the sake of simplicity and minimizing computation time. The more comprehensive approach will be done in the CNN section.

4-fold cross validation has been applied with multithreading on 4 cores of CPU. The configuration with the least mean validation error was the following:

- $m = 3$
- $M = 200$
- $\lambda = 1$

The MLP classifier has been saved and evaluated on the test sample left-out in the previous section. The test accuracy reached 88.5% which could be improved.

Convolutional Neural Network

The more thorough approach to the problem solution included the use of a convolutional neural network. The TensorFlow GPU framework with Keras library have been used to build that one.

While the parameters like number of filters and their sizes would be assessed by the hyperparameters optimization, the choice of the architecture required much more attention. Fortunately, there has been a similar problem already solved with very good results: a CNN classifier for handwritten digits. The details of the solution can be found here:

https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6

The other problem focused on recognition of digits only and has been trained using 16bit grayscale images. While the problems and datasets differ, the difference is small enough that the architecture has been adapted for this solution as well. This would be:

```
In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten -> Dense
-> Dropout -> Out
```

The following hyperparameters have been identified for optimization:

- Number of filters in the 1st convolution layer $\{16, 32, 48\}$
- Shape of filters in the 1st convolution layer $\{5, 7, 9\}$
- Pooling size after the 1st convolution layer $\{5, 6, 7\}$
- Number of filters in the 2nd convolution layer $\{16, 32, 48\}$
- Shape of filters in the 2nd convolution layer $\{3, 5, 7, 9\}$
- Pooling size after the 2nd convolution layer $\{1, 2, 3\}$
- Type of activation function $\{relu(x), \tanh(x)\}$
- Usage of dropout $\{True, False\}$

- Optimizer $\{adam, rmsprop\}$
  - $\beta_1 \in \{10^{-2}, 10^{-1}\}$
  - $\beta_2 \in \{10^{-4}, 10^{-3}, 10^{-2}\}$
  - $\epsilon \in \{10^{-12}, 10^{-11}, 10^{-10}\}$
  - $\rho \in \{1 - 10^{-2}, 1 - 10^{-1.5}, 1 - 10^{-1}\}$
  - $decay \in \{10^{-3}, 10^{-2}, 10^{-1}\}$
  - $l_r \in \{10^{-4}, 10^{-3.5}, 10^{-3}, 10^{-2.5}\}$
- Number of hidden units $M \in \{64, 128, 192, 256, 320\}$
- Usage of image augmentation $\{True, False\}$
- Usage of class weights $\{True, False\}$
- Usage of learning rate reduction $\{True, False\}$

The augmentation consisted of randomly rotating and shifting the input images during the learning phase for better generalization and overfitting reduction.

Class weights were used to penalize the learning algorithm for misclassifying the "minority" classes. The weights w_i were calculated as the ratio of the biggest class C_m to the smaller class C_i.

$$w_i = \frac{|C_m|}{|C_i|}$$

The dropout ratio and augmentation parameters were hold fixed for the sake of simplicity.

The optimization has been carried by 4-fold cross-validation on a GPU. The random approach has been taken: initial parameters' values have been chosen randomly and in each iteration randomly modified. Then validation accuracies have been compared to the best obtained so far.

This way the following set of hyperparameters has been chosen:

- 32 filters of size $7x7$ in the 1st convolution layer
- Pooling window of size 5 after the 1st convolution layer
- 48 filters of size $7x7$ in the 2nd convolution layer
- Pooling window of size 3 after the 2nd convolution layer $\{1, 2, 3\}$
- $\tanh(x)$ activation function
- Usage of dropout
- $rmsprop$ optimizer
  - $\epsilon = 10^{-11}$
  - $\rho = 0.9$
  - $decay = 10^{-3}$
  - $l_r = 10^{-3}$
- Number of hidden units $M = 192$
- Usage of image augmentation
- Omitting class weights
- Omitting learning rate reduction

The model has been trained for 20 epochs and saved. After evaluation on the benchmark test set, the test accuracy reached 95.3%.

Summary

After training the final model, the misclassified examples have been examined. First, the confusion matrix has been drawn for all of the classes (see Figure 3). As somehow expected, the most of the misclassifications are made for letters/digits of similar shape:

- 54 out of 67 ‘1’s have been classified as ‘I’s
- 58 out of 59 ‘0’s have been classified as ‘O’s
- 23 examples of ‘U’/’V’ have been confused with each other
- 20 examples of ‘2’/’Z’ have been confused with each other

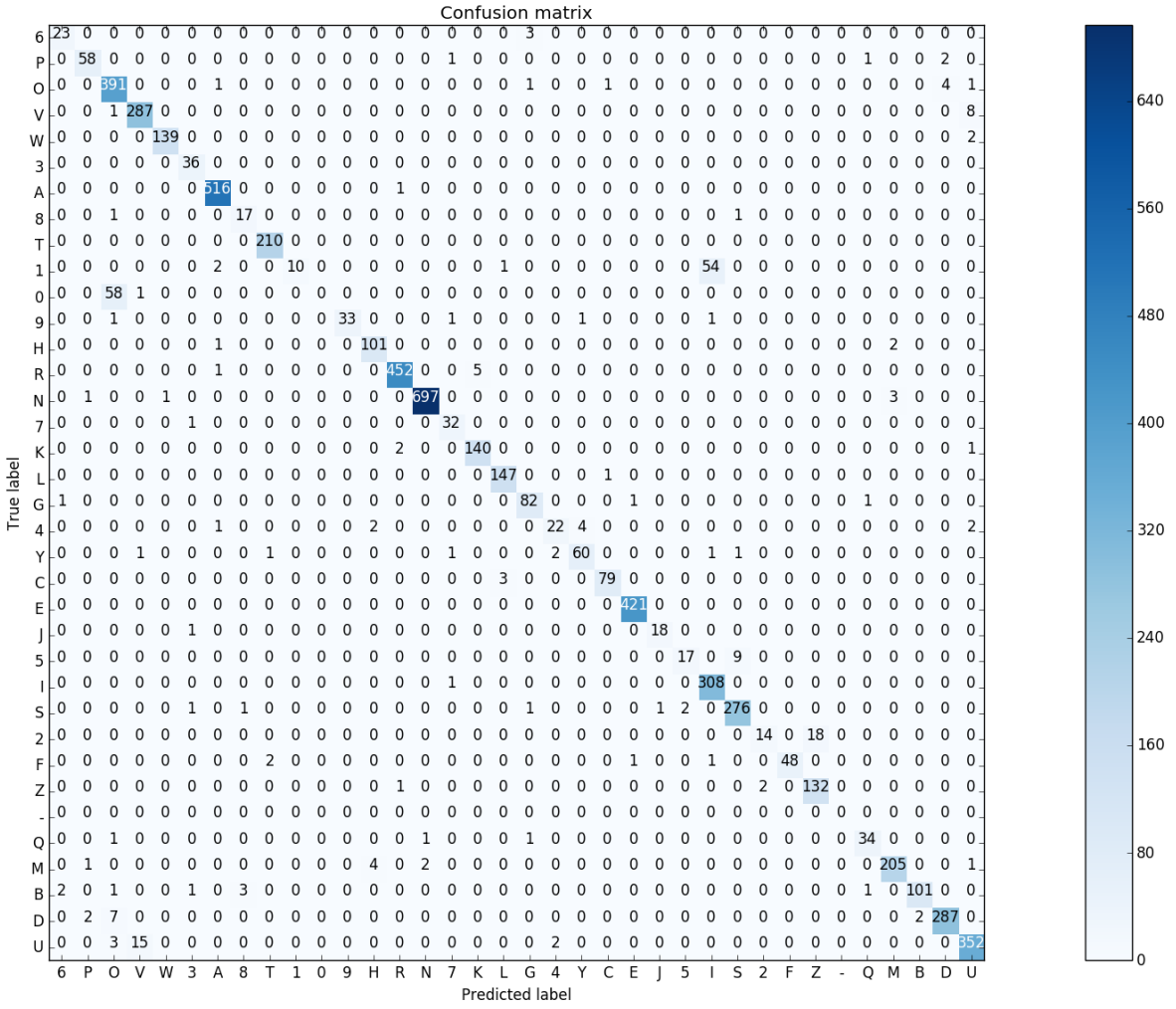

Figure 3 Confusion matrix for the test set

The closer look at the other, not so naturally similar classes revealed that the particular examples were alike. See Figure 3.

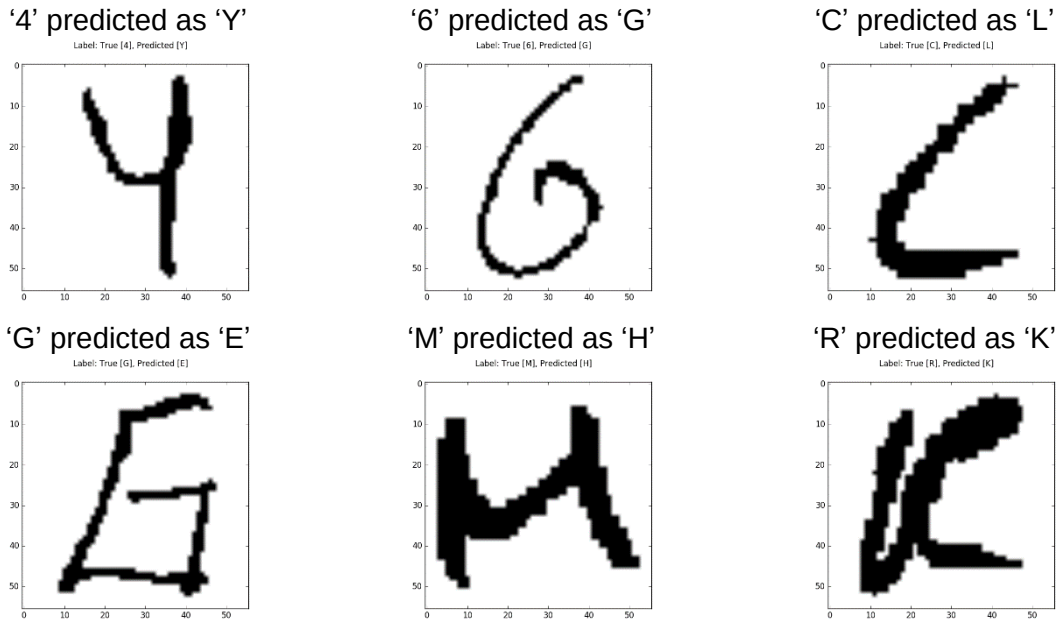

Figure 4 Exemplary misclassifications from the test set

As can be seen, the shapes of the letters/digits are very similar to the other, mistaken class. Some of the examples would even be problematic to humans – experts in handwriting ;-)

To further improve the model, the semantics/blocks of the characters could be provided. A round, circular character would be fairly easy to classify if found in “2018” or “Alphamoon” strings.