Alphamoon
Recruitment assignment

Marcin Gruza

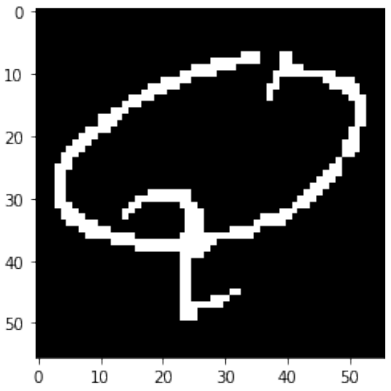
April 2021

1 Introduction

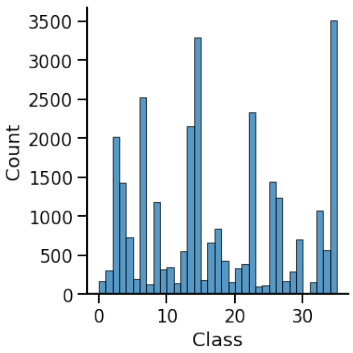
This report summarizes my work on the recruitment assignment of creating a simple number and letter classifier. The code is available at GitHub.

2 Dataset

The collection contains 30,134 binary images of 56x56 pixels, representing letters and digits. Each picture belongs to one of 36 classes. The analysis of the dataset shows that it is strongly unbalanced. Class *30* contains only one image (which looks like 'N' letter) while the remaining classes contain 102 - 3291 images. The distribution of classes counts is presented in figure .



(a) Sample image from the datset.



(b) Distribution of classes for the dataset.

3 Data preprocessing

Since the images in the dataset have fairly uniform and clean characteristics, I have not used any particular preprocessing techniques or data cleaning methods. The only preprocessing step for neural network models was to normalize images by substracting their global mean and dividing by global standard deviation.

To validate implemented methods, the dataset was randomly splitted into three parts: train, validation and test, with 22634, 2500 and 5000 samples respectively.

4 Models

To solve the problem, I chose and tested two main models and two simple baselines to compare the results.

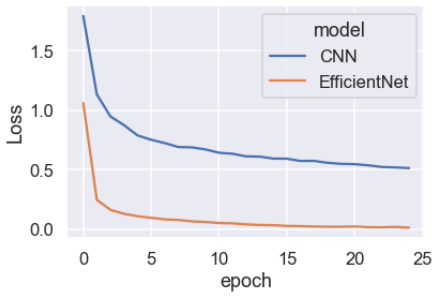
4.1 Baseline models

I selected two basic models: **Logistic Regression** and **SVM**, to set reasonable baseline solutions for the task. They both operate on images treating them as standard feature vectors.

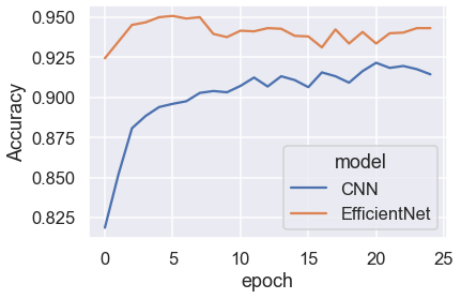
Both implementation for logistic regression and SVM were taken from sklearn library. For fair comparison, I merged train and validation parts of dataset before fitting the models. The maximum number of iterations of both algorithms was increased to 1000. The SVM model used rbf kernel.

4.2 Convolutional Neural Network

In my first advanced approach, I designed and tested my own convolutional neural networks. The architectures I tested consisted of several convolutional layers with max pooling followed by flattening last convolutional activations and feeding them into fully connected layers. Each layer was followed by ReLU activation function. Additionally, to prevent overfitting, I used dropout layers.



(a) Train cross entropy loss during training for CNN and EfficientNet models.



(b) Validation accuracy during traning for CNN and EfficientNet models.

The model was implemented using PyTorch library with PyTorch Lightning framework. The data was wrapped with DataModule for easier access and managing. The model metrics were logged with TensorBoard.

The hyperparameters for the model were searched manually. I experimented with number of convolutional layers, their kernel sizes and number of filters. The learning rate was determined manually, by selecting maximum possible value that keeps the learning curve stable. The number of epochs were set to 25, which ensured that the learning curves were flattened. Models with best validation cross entropy loss were checkpointed and saved.

4.3 EfficientNet

The last model I tested was a pretrained CNN EfficientNet b0 model. This is the simplest model from the EfficientNet models family and it consists of approximately 4 milion parameters. To adapt the model to my task, I replaced the model's classifier (last fully connected layer) with my own. Despite the use of pretrained weights, I didn't freeze any parts of model during training. The learning curves for both CNN and EfficientNet are presented in Figures 2a and 2b

The EfficientNet model and its pretrained weights were imported from timm library and implemented as PyTorch module. The hyperparameter search and training process were the same as in the CNN model.

5 Results

Due to class imbalance, models were evaluated with two metrics: accuracy and F1 score with macro averaging over classes. The evaluation results are presented in Table 1.

As expected, Convolutional-based models outperformed both baselines. The simple CNN model achieved only 3 % less in accuracy than EfficientNet, but the EfficientNet achieved much better result on F1 score. This suggests that the best model significantly improved its performance on the underrepresented classes.

Confusion matrix for the best found model is presented in Figure 4. It shows that there are several classes that cannot be correctly classified by the model. The most often misclassified classes are 9 (confused with 25), 2 (confused with

Model	Accuracy	Macro F1 score
Logistic Regression	78.4	67.9
SVM	87.8	74.4
CNN	92.0	79.0
EfficientNet	**95.0**	**88.0**

Table 1: % accuracy and F1 score for test dataset.

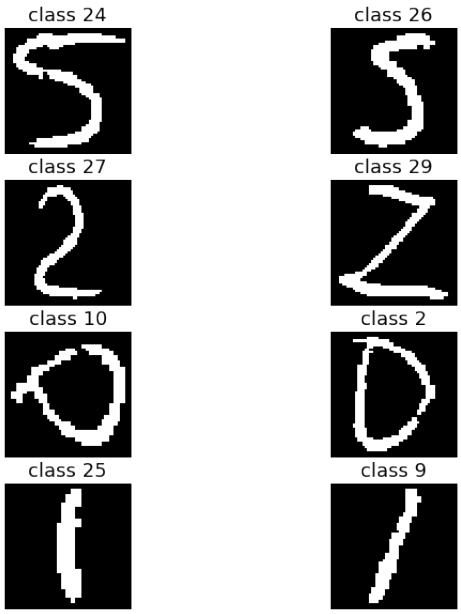


Figure 3: Examples of the most frequently confused classes by the best model.

10), 29 (confused with 27) and 26 (confused with 24). Examples of images from these classes are presented in Figure 3. As you can see, these are typical letters and numbers that are often confused with each other even by humans. Namely, these are digit '5' and letter 'S', digit '2' and letter 'Z', digit '0' and letter 'O', letter 'l' and digit '1' (or letter I).

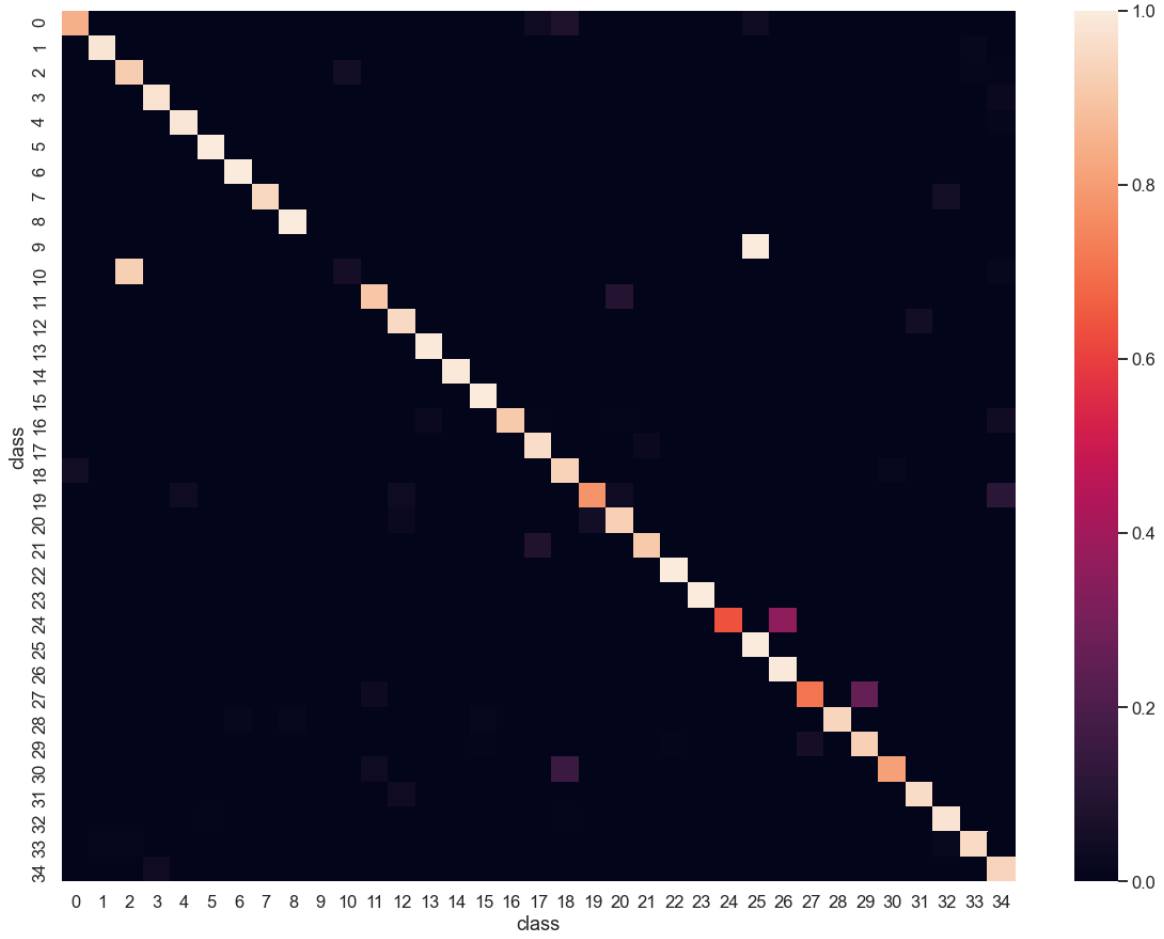


Figure 4: Confusion matrix on test dataset for the best model. The matrix was normalized by dividing each element by sum of elements in its row, so that all elements in a row sums to 1.

6 Conclusions

The results of the best implemented model appear to be quite good and even comparable to human performance. Both the metrics and confusion matrix shows that the model can easily recognize most of the characters in the dataset.

The biggest problem with this method is distinguishing between classes of characters that are very similar to each other. In this case, the model usually predicts the more frequent class. However, it can be difficult even for humans to correctly distinguish those characters without any context.

6.1 Future work

To slightly improve the experimental setup, the hard split of the dataset into train, validation and test parts should be changed to stratified k-fold cross-validation.

In order to further improve the model's performance, it would definitely be worth experimenting with hyperparameters (for example with grid search) and bigger network architectures (bigger EfficientNet, ResNet). To maximize the quality of predictions, we could make an ensemble of few models learned on different parts of the dataset and predict the final label as majority vote of the models. Data augmentation could help to improve quality of predictions for underrepresented classes and classes that are often confused with each other.

I think that data augmentation for problematic classes and more advanced pretrained models are the most important and promising steps that could improve the final solution.