Distinguishing Handwritten Symbols of Different Writing Systems

1. Introduction

Optical character recognition is a prominent field in machine learning having a lot of development in recent years with new research articles and commercial solutions improving significantly year after year. One of the earliest consumer devices applying OCR - Apple Newton - got released in 1993, and now, almost 30 years later, OCR is quite a mundane feature used by many to aid in their working and personal life.

Many character recognition systems have to know which writing system the scanned text belongs to in order to recognise it properly, thus we decided to develop a machine learning learning tool to distinguish handwritten symbols of different writing systems: Latin script (English), Arabic script, Cyrillic script (Ukrainian) and Hiragana. In order to train the model we picked, unified and combined datasets of characters for the abovementioned languages. We trained two models to classify the characters: Support Vector Machines and Convolutional Neural Network, evaluated their performance and picked the best. So, the application domain is Optical Character Recognition, recognizing which writing system a symbol belongs to.

Section 1 of this report is the introduction, section 2 talks about more formal problem formulation, section 3 describes the methods applied to solve it, section 4 discusses the results and section 5 finishes the report with the conclusion. In the end there is a list of references and an appendix with the code.

2. Problem formulation

The datapoints are Images of handwritten characters. We decided to use 32x32 pictures as they were easy to obtain, and give enough space to represent symbols of all the writing systems used (confirmed visually). Original datasets either already use this resolution or the images can be easily downsampled to it. Thus, as features we take 1024 numbers representing pixel intensities as a continuous integer from 0 up to 255 (uint8) (for better performance of CNN, the values were rescaled to be between 0 and 1). We agree that the symbols are white on a black background. We use categorical labels encoded as numbers: 0 - “Arabic”, 1 - “English”, 2 - “Cyrillic”, 3 - “Hiragana”. The problem is to predict the label based on the image - it’s a supervised learning task, as the labels are available in the dataset.

The dataset used for training the classifier and evaluating its performance was obtained as a combination of 4 openly available datasets of handwritten symbols of Arabic[1], English[2], Ukrainian[3] and Japanese[4] languages. From every dataset we took 1000 datapoints, converted them to a common format (1024 features-pixels, 1 label) and combined them together to form the dataset used.

3. Methods

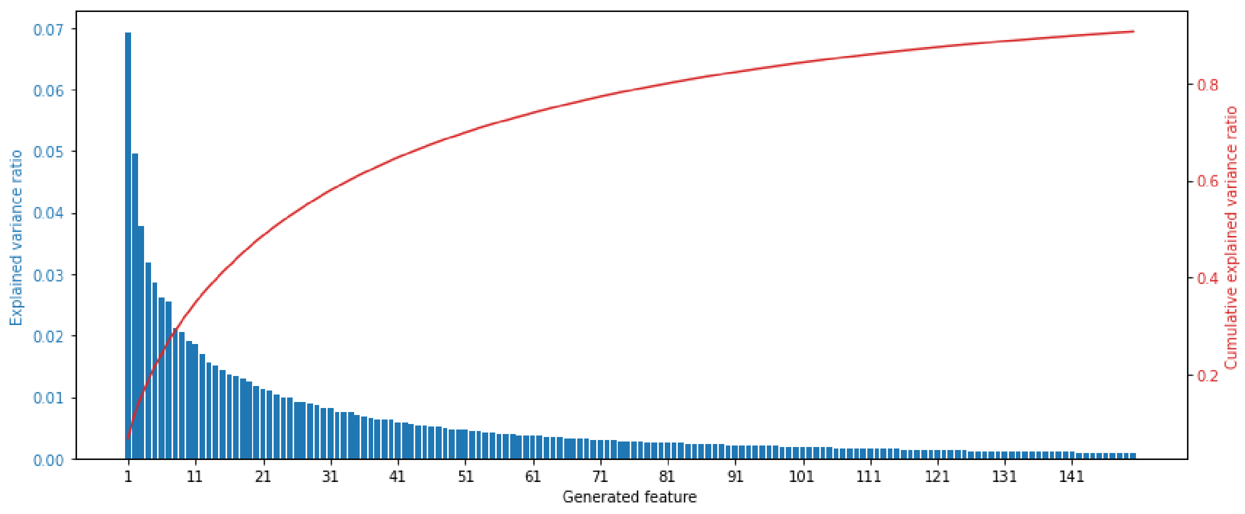
3.1. Dataset

The dataset contains 4000 points, to collect it we downloaded the original datasets, selected a 1000 samples out of each, preserving proportions of different letters, loaded images, downsized them to a common resolution of 32x32, inverted colors for the Latin and Cyrillic datasets to maintain common white-on-black pattern, converted pictures to pandas dataframes, where we left only the values of 1024 pixels and the label, and then combined dataframes into one. We decided to take only 1000 samples out of each dataset because it is the size of the smallest of the source datasets we have - the Hiragana dataset, with Cyrillic dataset being almost the same size. Even though the other two datasets are larger, we wanted to have even representation for every writing system. The dataset collection process can be seen in the code attached in Appendix A.

We make a 60%/20%/20% split between the training, validation and test datasets by using train_test_split twice with stratification by labels. We chose such a split because 4000 datapoints is not so much, so we didn’t want to make the validation and test datasets any larger, but at the same time wanted them to be big enough so that the results we get on them would have significance. Also, by comparing with several ML papers 60/20/20 seems like a popular and viable option.

3.2. Feature engineering

As 1024 features are too much for any machine learning method, and many pixels, being almost always the background, don’t carry as much information as the others, we decided to use Principal Component Analysis technique to reduce the number of features. PCA works by first constructing a new basis for the original feature space such that the first basis vector is in the direction of the maximum variation in data, second one is orthogonal to it and in the direction with the second best variability and so on. Then only k first basis vectors are selected and the dataset is projected into it.



We decided to keep the first 100 basis vectors, as the impact of everything after on the variance of the data is negligible (less than 0.5%)

Note that PCA was used only in combination with SVM method, the CNN was trained and validated on the original image data.

3.3. SVM

Support-vector machines are a family of ML methods for learning a hypothesis to predict a binary label y of a datapoint based on its features x. [5] SVM is similar to other classification methods, such as linear and logistic regression, with the difference in the loss function used. The hinge loss is a (in some sense optimal) convex approximation to the 0/1 loss. Thus, we expect the classifier obtained by the SVM to yield a smaller classification error probability compared to logistic regression which uses the logistic loss. [5]

We decided to use SVM because it is, generally, the best classification method compared to logistic regression. We also think there are specific properties of our dataset which make SVM the best choice: it allows us to prioritize a clear decision boundary over the perfect handling of the training data, which we think is important in distinguishing between latin and cyrillic - those writing systems have some identical letters, so it’s impossible to perfectly distinguish them, but SVM allows us to better capture the general tendencies of the writing systems. When we tried using both logistic regression and SVM, we observed that the former has the most difficulties with distinguishing Latin and Cyrillic, while SVM improves it to match the performance when distinguishing other pairs of the writing systems.

As we used the sklearn library for implementation, we chose the SVM with Radial Basis Function (RBF) and hinge loss, as those are the defaults. So the model (hypothesis space) is weighted sums of RBFs, because it is non-linear, which increases the complexity of the relationship the model can approximate, and also as it is already available in the sklearn library. Usage of hinge loss function is explained by it being the core of SVM methods, so by the same reasons as we mentioned earlier for SVM: better performance than logistic loss, good performance in case of partially intersecting classes (Latin and Cyrillic, in our case).

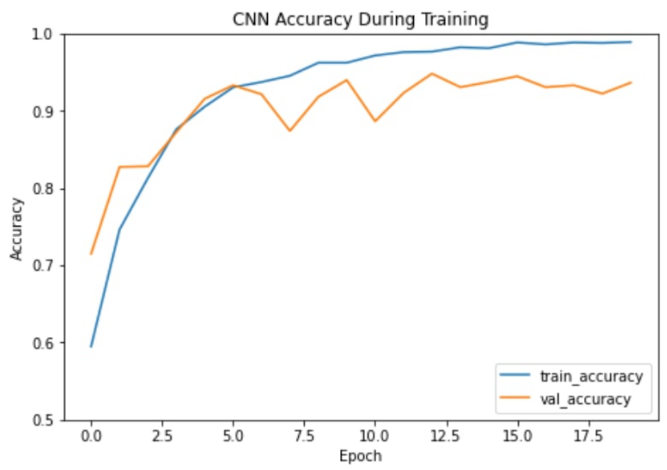
3.4. CNN

Convolutional neural networks is a type of artificial neural networks distinguished by presence of convolutional layers, which, instead of being densely connected to the next layer, have several kernels which are swept through the input of the layer to form the output. Those convolution layers and kernels can be 2- or 3-dimensional, which helps those neural networks perform better in the tasks requiring accounting for the spatial interdependencies, e.g. picture processing. As optical character recognition is also a picture processing task, we decided to use CNN for it. Specifically, our hypothesis space is CNNs with following layers:

1. 2D convolutional with 9 3x3 kernels and ReLu activation function, output shape (30, 30, 9)
2. Max pooling layer, 2x2, output shape (15, 15, 9)
3. 2D convolutional with 64 3x3 kernels and ReLu activation function, output shape (13, 13, 64)
4. Max pooling layer, 2x2, output shape (6, 6, 64)
5. 2D convolutional with 64 3x3 kernels and ReLu activation function, output shape (4, 4, 64)
6. Flattening layer, output shape (1024)
7. Dense layer with ReLu activation function, output shape (64)
8. Dropout layer with 0.2 ratio, output shape (64)
9. Dense layer with softmax activation function, output shape (4)

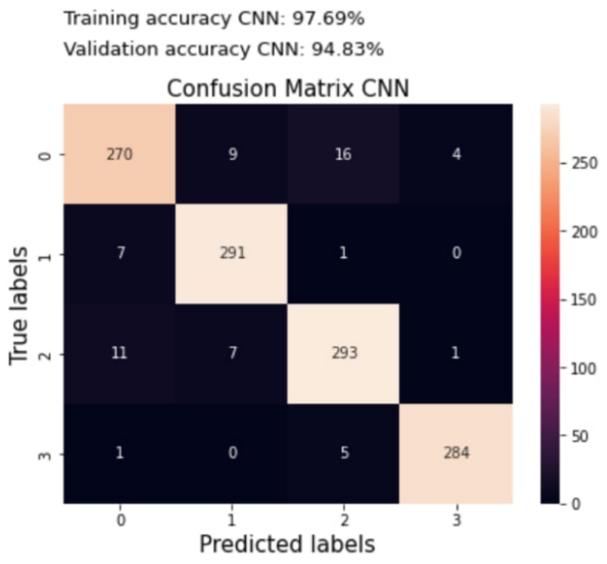
This architecture of the model is quite standard, in particular, most of the shapes of the layers are picked to be the same as in [6], where the CNN is also used for a classification problem on 32x32 images. The key difference is that in our problem the images are in grayscale, and have only one channel instead of three, so kernels on the first layer has only 9 weights => any kernel can be described as a linear combination of 9 linearly independent kernels, so no more than 9 kernels are needed on the first layer (this was experimentally checked). Also a dropout layer was added, which randomly breaks connections during the training. It helps prevent overfitting for the training data.

We use categorical cross entropy loss because it’s well suited for the multiclass classification problems with neural networks, and because it’s already implemented in the tensorflow framework.

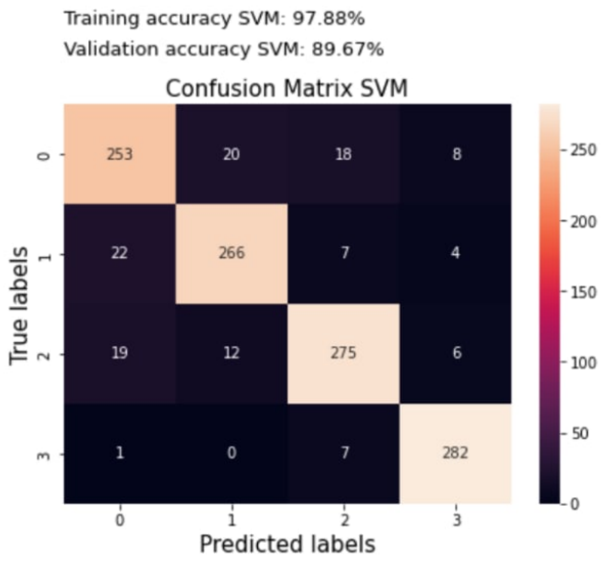


Also, we trained the network for 20 epochs, and, based on the maximum of the validation accuracy, decided to have 13 epochs (with the same value for random state).

4. Results



```
Training accuracy CNN: 97.69%
Validation accuracy CNN: 94.83%
```

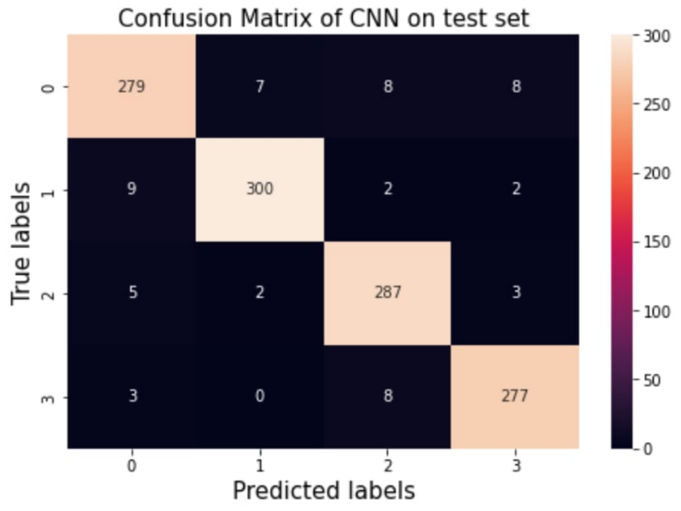


```
Training accuracy SVM: 97.88%
Validation accuracy SVM: 89.67%
```

Both methods achieved very similar accuracies on the training data: 97.69% for CNN and 97.88 for SVM. However, the accuracy on the validation set is higher for the CNN: 94.83% against 89.5%, thus we pick CNN as the best model and the final chosen method.

As we discussed, we took 20% of the initial dataset of 4000 samples - 800 images - as test dataset. Those images were never used in training, nor in choosing between the classifiers, so the accuracy on them correctly represents the chosen method performance on the unseen data. The accuracy of CNN on the test dataset is 95.25%, and the error - measured as the average loss - is 0.146.

```
Test Average Loss CNN: 0.1460
Test Prediction accuracy CNN: 95.25%
```



5. Conclusion

As we can see, both SVM and CNN are suitable ways for the task of distinguishing symbols of different writing systems, but CNN is better. Also, judging by 95% accuracy on the test set, we can say that good performance during training was not due to overfitting, and the model performs well on the unencountered data.

One of our concerns is that, due to the usage of several source datasets, the neural networks ended up training to distinguish those datasets rather than distinguishing the writing systems. We tried to minimize the discrepancies between the datasets as much as possible: we convert them all to the same resolution, all white on black, but we can consider collecting our own dataset for all writing systems as a potential direction for improvement: by collecting all the samples similarly we can ensure the uniformity. It can be done, for example, as a website, or a pop-up data collection point somewhere on campus, so that tablets can be used for better accuracy of collection.

References

1. https://www.kaggle.com/datasets/mloey1/ahcd1?resource=download
2. https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset
3. https://www.kaggle.com/datasets/lynnporu/rukopys
4. https://github.com/inoueMashuu/hiragana-dataset
5. https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf
6. https://www.tensorflow.org/tutorials/images/cnn