Comparison of Machine Learning Models in Cyrillic Text Recognition

Owen Crewe
University of Victoria
ocrewe@uvic.ca

Christopher McLaughlin
University of Victoria
christopherm@uvic.ca

Christian Koutsandreas
University of Victoria
christiank@uvic.ca

Abstract

This report will compare machine learning methods on classifying Cyrillic characters. The models of interest are decision trees, all-vs-all collections of support vector machines, and convolutional neural networks where optimal models and hyperparameters for each respective model will be reached through an iterative process of permuting hyperparameters and comparing results. Results will additionally be compared with results from a deep residual convolutional neural net model generated using the ResNet50 architecture. Metrics of interest for all models include validation accuracy and training time of each model, with the aim being to identify the most effective method given considerations of acceptable error threshold and finite compute resources.

1 Introduction

For an estimated 250 million people in Eurasia [1], the Cyrillic alphabet represents the primary written characterset of their national languages. While in recent years much progress has been made in the field of text recognition, both through optical methods and direct pixel-by-pixel analysis, as it pertains to the Latin alphabet used by a majority of the world's languages, comparatively less time and effort has been invested into the development of text recognition models that specialise in Cyrillic.

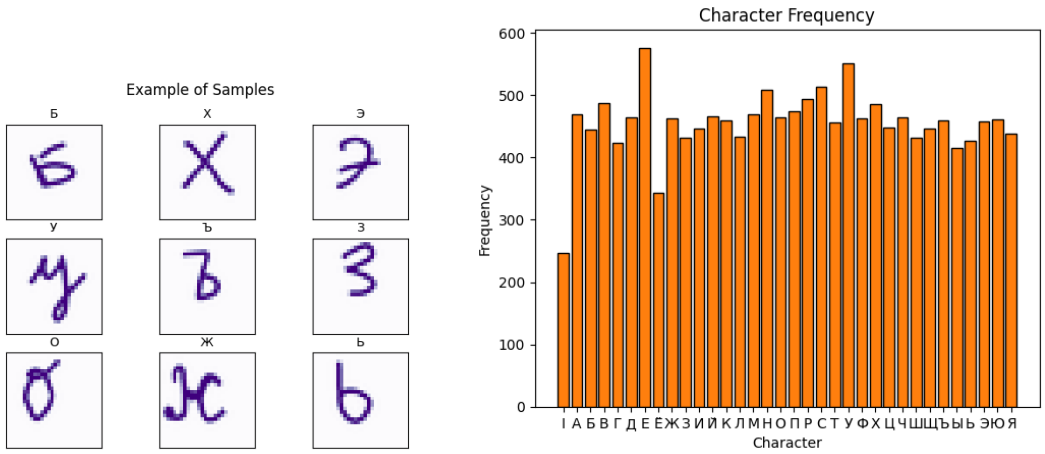
This paper will outline several different machine learning approaches and their application to character classification within the Cyrillic alphabet, comparing their relative accuracy, reliability, and compute-time resource cost in establishing. Classification of the 34 characters in the dataset is performed via decision tree, support vector machine and convolutional neural network models.

The dataset used is sourced from a publicly available repository of images of cyrillic text [2], and processed into a MNIST-like format through our own processing algorithm through a process loosely inspired by the "CoMNIST" dataset project available on Kaggle [3].

2 Dataset and Data Processing into MNIST-like Format

The original intention was to implement our models using the Co-MNIST dataset available on Kaggle [3], however we quickly found the dataset to be incomplete. The provided MNIST-style data covers only a subset of the original image set. As such, we were required to construct our own dataset using the original images used to construct Co-MNIST. These images are available on a publicly accessible GitHub repository[2] but are not in MNIST-like form, rather being direct handwriting samples acquired through crowdsourcing. As such, our first task was to process this raw data into a MNIST-like dataset. This was accomplished by a dedicated python script, with the results saved to files `dataX.csv` and `datay.csv`. These files were then used to construct and evaluate all of the model varieties.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.



(a) (b)

Figure 1: (a) A random selection of 9 characters from the dataset; (b) Dataset character frequency

The dataset itself is composed of 15480 labeled characters, with each character being a 28x28 grid that is reshaped to a single 1x784 array for purposes of data storage and reshaped as needed by each model. Each entry in the array is an unsigned 8-bit integer that represents the single, grayscale, colour channel.

As evidenced by Figure 1a, several pairs of characters are very similar, representing an interesting challenge for each of our models. Additionally, since our MNIST-like dataset was generated from handwriting samples directly, the characters are not as reliably centred in frame as with the standard MNIST dataset. This may cause an additional challenge for model training, but could also build in some tolerance to models, improving real-world efficacy.

Model training was additionally complicated by lack of a uniform distribution of the data, stemming directly from the raw handwriting samples. Figure 1b illustrates the datasets lack of uniformly distributed character frequency. Certain characters, such as “I” and “E”, are notable outliers in the frequency graph. This could negatively impact the models’ biases when classifying these characters.

3 Methodology and Model Construction, with Preliminary Results

3.1 Decision Trees

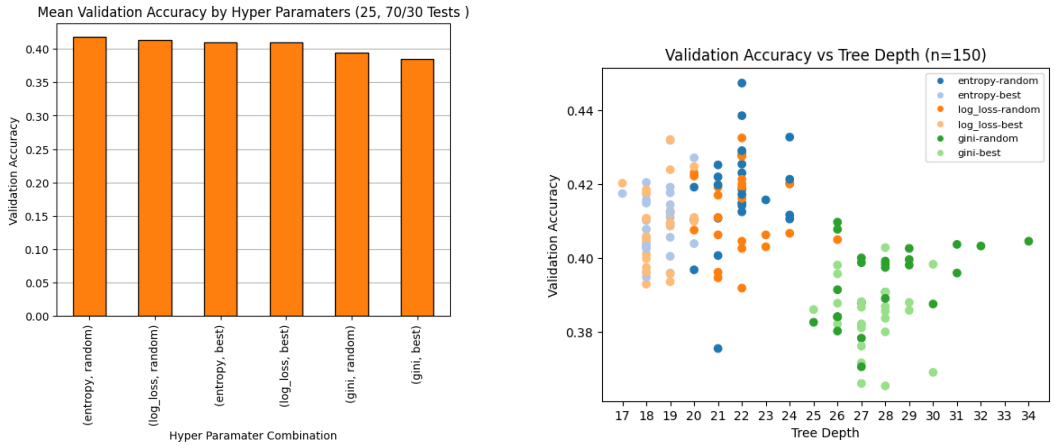
The first machine learning method tested was decision trees. All permutations of criterions `entropy`, `log_loss`, and `gini` were tested, as well as splitters `random` and `best`. These 6 permutations were tested 25 times each in order to avoid potential sampling biases in performance evaluations.

Figure 2a displays the performance results of all tested combinations of hyperparameters for the decision tree model. It can be seen that, while all combinations performed similarly, the combination of `entropy` as the criterion and `random` splitting produced the best results. Although these results were still not very accurate. These findings suggest that decision trees are not well-suited for classifying the dataset being worked with.

Figure 2b shows how the choice of criterion affects the depth of the generated decision trees. It can be seen that the `log_loss` and `entropy` criteria resulted in trees with similar depths, while the `gini` criterion consistently produced the tallest and, per Figure 2a, least accurate trees. This highlights the ineffectiveness of `gini` in producing accurate and storage-efficient results on this dataset.

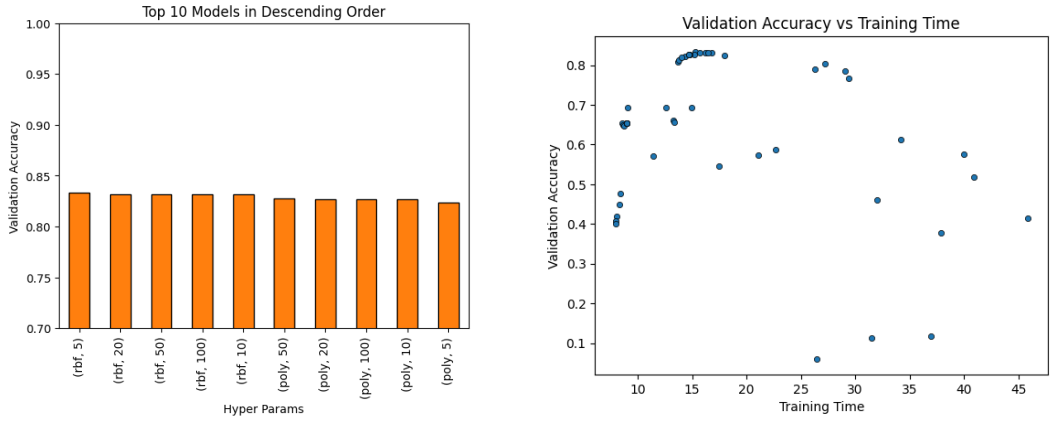
3.2 Support Vector Machines

The next model employed in our experimentation was the support vector machine (SVM). We implemented an all-vs-all (AVA) model of SVM classification, which involves creating pairwise SVMs for each character pair in the dataset, or $n(n-1)$ total pairwise binary classifiers for each



(a) (b)

Figure 2: (a) Mean validation accuracy for each hyperparameter combination of decision-tree model; (b) Validation accuracy and tree depth for each permutation of the decision-tree model



(a) (b)

Figure 3: (a) The top ten best performing AVA SVM aggregations; (b) Training time and validation accuracy for all 60 AVA variations

AVA classifier. In our case, this meant generating 1122 binary classifiers for each AVA, and repeating this process for each permuted hyperparameter in the overall AVA. We tested 60 variations of the AVA hyperparameters (training time vs validation accuracy shown in Figure 3b), generating a total of 67320 binary classifiers. This rendered the SVM our by far most training-time intensive model, with the results typically taking approximately five hours to compute. Hyperparameters tested included each of the `rbf`, `linear`, `polynomial`, `sigmoid`, and `precomputed` kernels, with twelve c values interspersed between 0.05 and 100. Preliminary results show great improvement over our decision-tree based model, but these improvements come at a high price in compute cost. As seen in Figure 3a, the best SVM hyperparameter set in terms of accuracy was, by a narrow margin, the `rbf` kernel coupled with a c value of 5. Notably, all `rbf`-based SVMs outperformed their counterparts across the board.

3.3 Convolutional Neural Networks

Our third machine learning method was convolutional neural networks (CNNs). We performed a thorough hyperparameter search by testing 30 permutations of two key hyperparameters: dense layer outputs and convolutional filters. To avoid sampling bias, we tested each pair of values twice. The convolutional filters were evaluated across a range of values: 34, 64, 80, 96, and 128. The dense layer

(a) (b)

Figure 4: (a) Top ten CNNs by accuracy; (b) Validation accuracy versus training time, coloured by no. of weights in the network

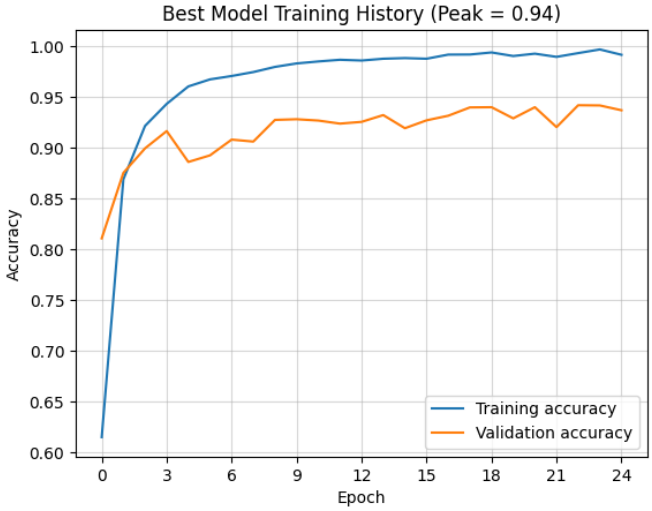


Figure 5: Training versus testing accuracy of the best performing CNN

outputs were tested using the same set of values, with the additional parameter 48 for a larger sample space. We used early stopping with a patience of five epochs to maximize validation accuracy and reduce overfitting.

While we expected CNNs to outperform the other tested methods, as they have been shown to be effective in image recognition tasks, we were nevertheless impressed with the results, which can be seen in Figure 4a. CNNs drastically outperformed even our highly accurate AVA SVM classifiers across the board, for a fraction of the compute cost and training time. As illustrated by Figure 4b, the most parameterised CNNs were not always the most accurate, although there was a general linear trend of increasing accuracy with more parameterisation.

From comparing our constructed CNNs, we can see our best performing model was achieved with 96 convolutional filters and 128 dense outputs. It achieved a validation accuracy of just under 94%, rendering it our most accurate model achieved thus far. This specific CNN's scoring history for each evolutionary epoch is charted in Figure 5.

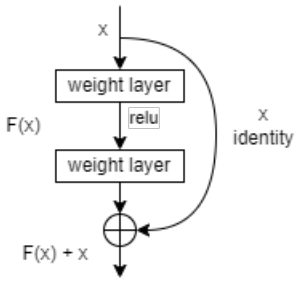


Figure 6: Residual block architecture

3.3.1 ResNet

Residual neural networks (ResNets) are a class of convolutional neural networks that allow training of models with layer counts in the thousands [4]. This is done through identity shortcut connections: some layers are skipped in initial training and expanded later in training to expand coverage of the feature space. "Residual blocks" (Figure 6) are the foundation of ResNet architecture. Residual blocks connect an identity mapping of a previous output to the output of a block of convolutional layers in order to smooth gradient flow and allow deeper networks.

For the purposes of this project, a pretrained network, ResNet50, was used via Keras. This pretrained network is a 50-layer-deep residual neural network that is trained on over one million images from ImageNet database [5]. The model was trained on a training cut of the Cyrillic dataset for approximately 90 epochs over approximately 6 hours until it reached a peak validation accuracy of 95.57%. When the test cut was run through the model, the resulting accuracy was 95.43%. Though untested here, this article by K. Aguas [6] suggests using a higher resolution of data. He runs a similar model using ResNet50 with a comparable image resolution of 32 x 32 pixels, and found a notable improvement when upsampling image resolution to 224 x 224 pixels

4 Comparison of Models and Results and Conclusion

When comparing our results across each of the different models tested and each hyperparameter permutation within each model, we can see some potentially useful patterns emerge. Looking at the best performing model of each model class and comparing the validation accuracy (Figure 10), we see that convolutional neural networks returned the most accurate results by a significant margin, as well as one of the best ratios of validation accuracy to training time. The residual neural net model using the ResNet50 architecture provided the greatest accuracy in testing overall, at 95.43%.

A validation accuracy of 95% is enough to have useful real-world implications, though this model could likely be improved even further. We would recommend the CNN model as the basis of any further refinement in the field of Cyrillic text recognition going forward. The performance of this model dramatically surpassed that of the decision tree-based model, and surpassed our SVM model by a significant amount in terms of validation accuracy and a tremendous amount in terms of validation accuracy per unit of training time.

References

[1] Cyrillic Script. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Cyrillic_script

[2] G. Vial, et al. CoMNIST. GitHub. https://github.com/GregVial/CoMNIST

[3] G. Vial, et al. CoMNIST. Kaggle. https://www.kaggle.com/code/gregvial/load-and-display-cyrillic-alphabet

[4] PyTorch ResNet. run.ai. https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet

[5] ImageNet. http://www.image-net.org

[6] K. Aguas. A guide to transfer learning with Keras using ResNet50. Medium. https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b