Automatic Text Transcription (OCR)

Task definition

The goal of the project is to train and evaluate neural network for automatic text transcription (OCR). We decided that the resulting application will be able to train and save model on given data and then be able to use that model to transcribe text from images of lines of text (single image contains single line of text).

Short review of existing solutions

Tesseract-OCR

Tesseract-OCR is OCR engine by Google. It is available either as command line application, or python library (pytesseract). It uses many moder techniques including *long short term memory*. The model can be trained on custom data and there also exist pretrained model. The model supports many languages.

Pero-OCR

Pero is OCR engine that can be used to detect and transcribe text in images. It can detect paragraphs and lines of text and than transcribe the the text in the image. It was mostly trained on european handwritten texts, mostly czech. It is available as command line application and python module.

Description of solution

We decided to use recurrent convolutional neural network implemented in python using pytorch. To speedup the training, we decided to use pretrained model for the convolutional part.

As a training dataset we have chosen the pero dataset, which contains images of lines of handwritten text in european languages. The dataset is quite large and we don't have enough hardware to properly train and time the network on all of the data so we filtered the dataset to contain only czech texts.

Data preparation

Before the image can go to the Neural Network itself, it has to be prepared first. We set the fixed image height to 64 pixels and resized all the images proportionally to the fixed height, to preserve the aspect ratio. Next we normalized the image values to the range of $\langle 0; 1 \rangle$.

Another important step was adding padding to the images, since we allow setting custom batch size. Each batch size has to be able to stack the images on top of each other, so they have to have the same size. The images are extended with zeros to match the width of the widest image.

Additionally for training we convert the dataset to the DataLoader type, which allows us to do the batches more easily and also have random sampling, which makes the learning faster.

Neural Network

The core of the solution is the Convolutional Recurrent Neural Network. This type of neural network is very commonly used for Optical Character Recognition, because it combines the core functionality from the Convolutional Neural Networks and Recurrent Neural Networks. CNNs are capable of extracting the spacial features, while the RNNs are designed for processing the sequantial data, such as text, which is what we need.

Convolutional Network

As a base for the CNN we use pre-trained ResNet-18 network. CNN contains two layers, where the second one was added later for making the receptive field larger.

Recurrent Network

The features from CNN are then fed into Bidirectional Recurrent Network layer, with the hidden size of 1024. We use bidirectional so the prediction doesn't depend only on the past frames, but also on the future frames. This is crucial for handwritting, where the letter connections change depending on the letters themself.

Loss function

To evaluate the Loss, we decided to use Connectionist Temporal Classification (CTC) Loss. Advantage of using CTC is that it doesn't care about the position of the letter. To ensure the stability of learning, we enabled `zero_infinity` parameter in PyTorch's implementation. This zeros infinite loss values, which would result in `NaN` gradients.

Optimization

We also implemented some optimization techniques. Firstly we used Adam optimizer with a learning rate of 0.001, which changes the learning rate based on the gradients. Next we implemented Adaptive Learning Rate Scheduler with patience parameter set to 5. When 5 consecutive epochs have similar Loss, it can mean that it's bouncing around the bottom of the valley, so the scheduler reduces the LR by the set factor value, which in our case is 0.8.

Transcription

The output of the RNN is projected through linear layer to classes, where classes are list of unique characters from the dataset, plus a blank symbol used for CTC.

Training evaluation

Firstly we need to decode the model outputs to the actual predicted text using CTC decoder. It select the class with the highest probability, merges repeated characters and removes blank symbols. Then we were able to calculate the Character Error Rate based on the Levenshtein distance between the predicted text and the target text. This allows for more better accuracy metric, since normal accuracy would increase only when all the characters in the prediction are correct, which is quite rare considering the length of the text lines.

Experiments

We experimented with writing our own text, but this uncovered issues connected with the way we had to edit the dataset, which will be described later. In general it was able to predict, but in some cases when the writing was distinct enough from the writings in the dataset, it started to struggle.

For example this text model predicted as "Ahůz, jak se máš.", which is quite close, but it's not perfect.

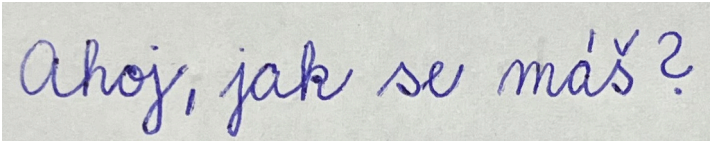

Figure 1: Handwritten text: "Ahoj, jak se máš?"

Or this text, where model didn't make that many mistakes, but it changed it enough for it to be difficult to understand: "Hello aecrld, cursiv wores?".

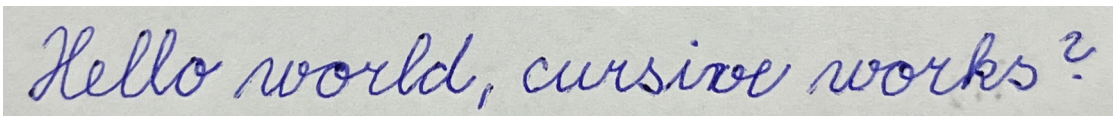

Figure 2: Handwritten text: "Hello world, cursive works?"

And in some cases the model was really far from the correct prediction: "kus vs teds o kranuptesr-".

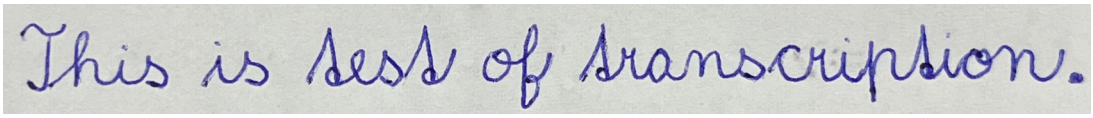

Figure 3: Handwritten text: “This is test of transcription.”

The dataset mostly contained text in cursive. Because of this, non cursive text works much less than text in cursive. The predicted text is “Žest na lextu pšaném tiskadm.”.

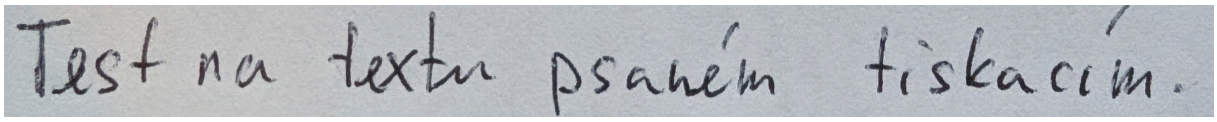

Figure 4: Handwritten text: “Test na textu psaném tiskacím.”

On the other hand, the model has learned to transcribe text quite well with the handwriting from the dataset. The following image comes from the original dataset and has not been used in the training in any way and yet the model predicts the text quite well as: “z 3 milionů na 1/4 Milionu, každý atcivnáš dostane místo” even though I have hard time reading it myself.

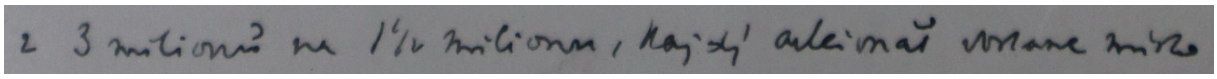

Figure 5: Handwritten text: “z 3 milionů na 1 1/2 milionu, každý akcionář dostane místo”

Overall the model has learned quite well to recognize text in the handwriting that it has been trained on. On the other hand, it has problems recognizing different handwritings. This is propably because handwriting of different people differs a lot and the training dataset doesn’t contain enough different handwritings for the model to learn the general features of different letters well enough.

Evaluation results

The model makes very obvious mistakes. This is because it recognizes each character separately and doesn’t use much of the context to determine what character this is based on the other characters. This could be improved by using attention which will allow the model to use more of the context to determine what character there is. The disadvantage of that is that the model will become much more language dependant and it will try to form words in the language that it was trained on.

The model was trained and evaluated on Handwriting Adaptation Dataset from Pero project, where the dataset was split to training dataset (90%) and testing/evaluating dataset (10%). We also used random sampling strategy. The dataset is really large and we don’t have enough powerful device to make the training last reasonable amount of time. For that reason we had to select only some datasets from the original dataset, which reduced the training duration to managable length.

For evaluation we used the already mentioned CER metric. We ran the training for 100 epochs, after which model with the largest accuracy given by the CER metric was selected. The accuracy was measured on the testing dataset. Our best model reached accuracy of 94.6% in epoch 74.

Progress of the model during the training may be seen in the following plots that show how loss and accuracy changes during training:

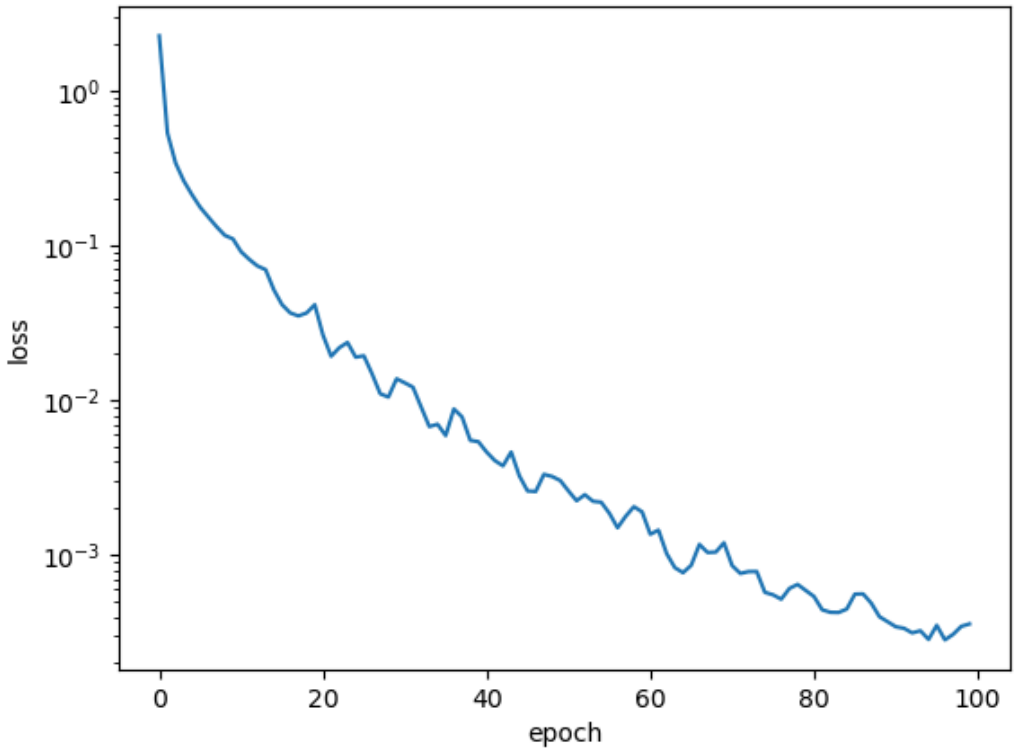

Figure 6: Log scale plot showing how loss changed during training.

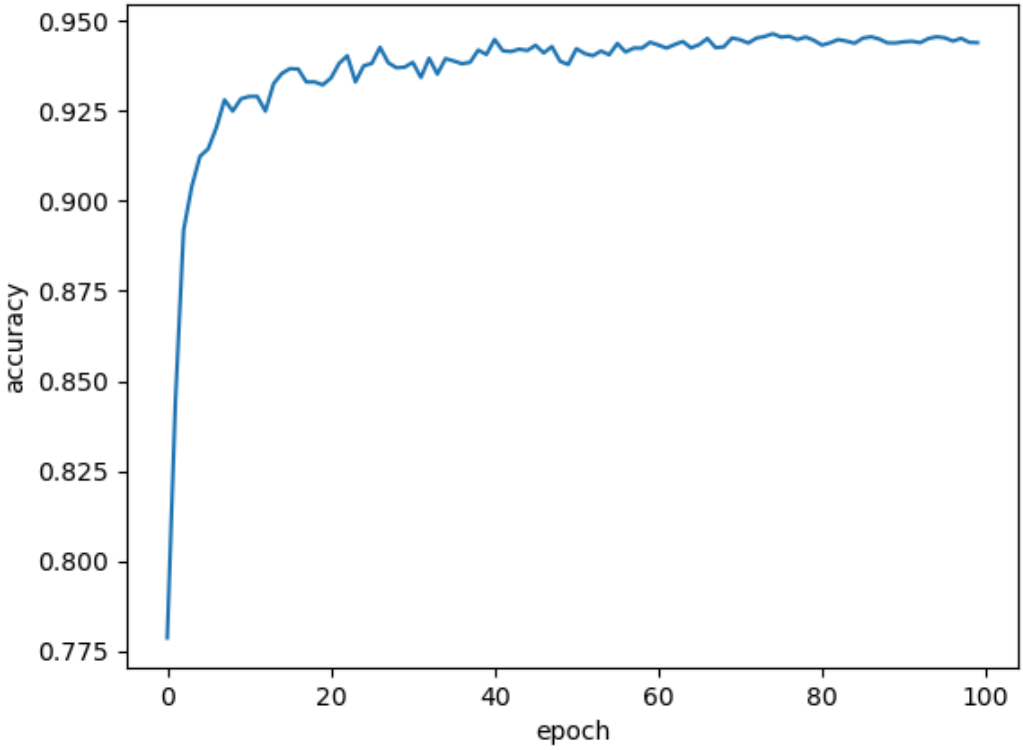

Figure 7: Plot showing how accuracy changed during training.