Optical Character Recognition of Cyrillic Characters Using a Convolutional Neural Network

March 27, 2017

James Robb jamesr15@ru.is
Sigurður Helgason sigurdurhel15@ru.is
Sævar Óli Valdimarsson saevarv15@ru.is
Reykjavik University
Artificial Intelligence T-622-ARTI

Contents

1 Introduction **3**

1.1 Optical Character Recognition 3

1.2 Inspiration 3

1.3 Technologies and Software Used 3

2 Dragnet - A Convolutional Neural Network **4**

2.1 Overview of Dragnet 4

2.2 Convolution 4

2.3 ReLU 5

2.4 Pooling 5

2.5 Fully Connected Layers 6

2.6 Training Dragnet 6

3 Preparation of Training Data **7**

4 Preparation of Real World Data **7**

5 Reconstructing Documents from Dragnet's Output **8**

6 Results and Conclusion **9**

1 INTRODUCTION

1.1 Optical Character Recognition

Optical character recognition (OCR) is the practice of mechanical or electronic conversion of images of typed, handwritten, or printed characters to a machine encoded format. The goal of this is often to make this text parseable, editable, searchable, or digitally workable in some manner. This can be cumbersome because images can be distorted, blurred, warped, or have other unfavorable attributes that make OCR difficult.

1.2 Inspiration

On February 22nd of 2016 a series of documents were dumped into a lake at the residence of the former president of Ukraine in an attempt to destroy them as the former president was fleeing. Volunteers retrieved the documents from the lake and then proceeded to dry and scan them. The Organized Crime and Corruption Reporting Project (OCCRP)[1] made these scans available online through the YanukovychLeaks National Project[2]. They used the GNU OCR tool[3] to digitize the documents. Unfortunately, given the volume and condition of the documents, along with limited capacity of the volunteer force, much of the text produced to accompany the scanned documents is poor or incorrect. We were interested to see if it was possible to use a neural network to produce comparable or better results than the GNU OCR tool.

In this project we will focus on OCR of Cyrillic characters utilizing a neural network. With the recent release of TensorFlow and the large community behind it, we have selected to use it as a means of performing OCR on the scanned documents from the YanukovychLeaks.

The name of the neural network, Dragnet, was inspired by the policing term dragnet. Wikipedia defines dragnet as "any system of coordinated measures for apprehending criminals or suspects; including road barricades and traffic stops, widespread DNA tests, and general increased police alertness. The term derives from a fishing technique of dragging a fishing net across the sea bottom, or through a promising area of open water."[4].

1.3 Technologies and Software Used

- TensorFlow[5] - Open source library for machine learning. It is used as the base of the neural network.
- ImageMagick[6] - Open source utility for image manipulation. It is used to extract character regions in images.
- Pillow[7] - Open source python library for image creation and manipulation. It used to generate the training data.
- Dragnet source code[8] - The source code for Dragnet. It is written in Python3.

2 Dragnet - A Convolutional Neural Network

2.1 Overview of Dragnet

A convolutional neural network (CNN) is a supervised form of machine learning in where convolution[9] is used to extract features from a data set. The goal of CNNs is to be able to accurately classify input (often images). To demonstrate how this is achieved, we will discuss the structure of Dragnet, a CNN we developed using TensorFlow.

The structure of Dragnet is a typical introductory example of a CNN. It consists of several layers of neurons which can be likened to nodes in a graph. The input layer receives the data to be classified. In the case of Dragnet, this is 900-dimensional vector of real values between 0 and 1 representing a 30×30 pixel image of a character. The values between 0 and 1 are a scale from complete white to complete black for that pixel. After the input layer are two hidden layers followed by an output layer. Hidden layers are called hidden simply because they are between the input layer and output layer. If one imagines the input layer starting on the left, there are directed arcs between each layer towards the output layer on the right as seen in Figure 1. The first and second layers of Dragnet consist of a convolution operation, a ReLU operation, and a pooling operation. The third layer is a fully connected layer with a dropout operation. Finally, the output layer is also fully connected layer whose output reflects the probabilities of the input being a particular character.

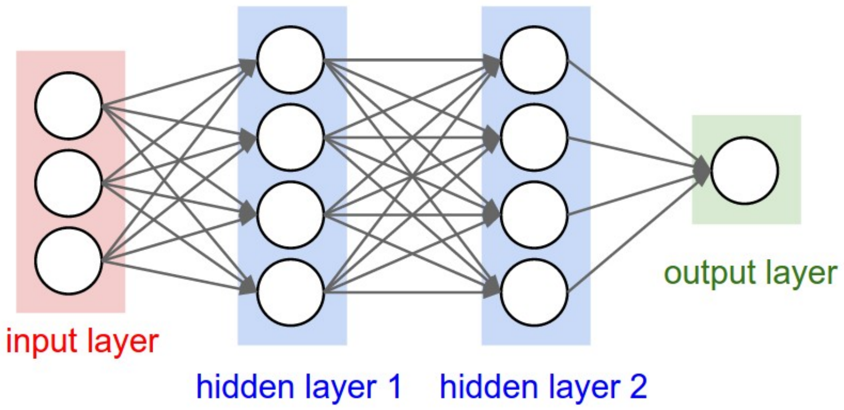


Figure 1: A visual example of a neural network with two hidden layers. Source[10]

2.2 Convolution

Before applying convolution we reshape the 900-dimensional vector into a 30×30 tensor[11], which is essentially an n-dimensional array data structure used for almost everything in TensorFlow. The convolution operation slides a 5×5 window over the tensor. This window is called a *filter* or a *kernel*. There is no requirement that a filter be 5×5 in size; this is just what Dragnet uses. A filter is compromised of weights (real values) and is used to build a feature map via the dot product of the filter and the portion of the image tensor it overlaps. A visual example of this can be see in Figure 2. In Dragnet, 32 and 64 filters are created in layer 1 and layer 2 respectively for the character tensor consisting of TensorFlow variables[12] that will be tuned during the training steps, but initially are drawn from a normal distribution at random. For each feature map generated from the filters, a bias is added, which is also a TensorFlow variable that will tuned during the training steps. The feature maps and their biases (after pooling) can be seen as the weighted edges from one neuron to another in the next layer.

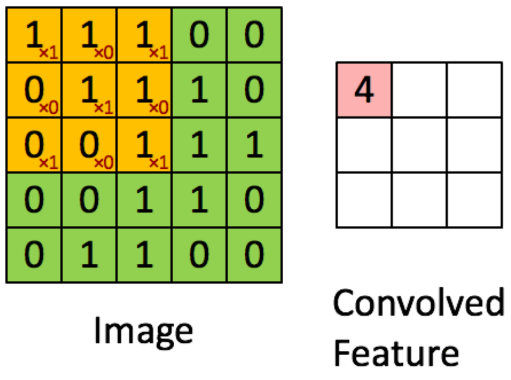


Figure 2: filter being used to build a feature map/convolved feature. Source[13]

2.3 ReLU

Neural networks utilize an activation function for each neuron in order to simulate actual neurons in a brain. There are many different activation functions used, but discussing them is not in the scope of this report. Dragnet utilizes a rectifier activation function[14]. A neuron that utilizes a rectifier activation function is called a ReLU (Rectified Linear Unit). We define the ReLU operation as applying the rectifier activation function to each value in a feature map. The rectifier activation function is defined as $f(x) = max(0, x)$. This introduces non-linearity into the CNN as any negative values are replaced with 0. This is important as the data Dragnet will process is not linear in nature.

2.4 Pooling

Pooling is a means of reducing the spatial size of input to help reduce the amount of computations required in the neural network while maintaining the important features of the input. This helps to reduce over fitting to the training set as we move deeper into the network, as only more abstract features are kept. Several different methods of pooling are available; a very common method is to use max pooling with a 2×2 window and a stride of 2. This is what is utilized in Dragnet. This means that a 2×2 window is slid over the tensor selecting the max value in the overlap of the tensor and the window, and then moving it by the stride each time. This will reduce the dimensions of the tensor by a factor of 2. A visual example of max pooling can be seen in Figure 3

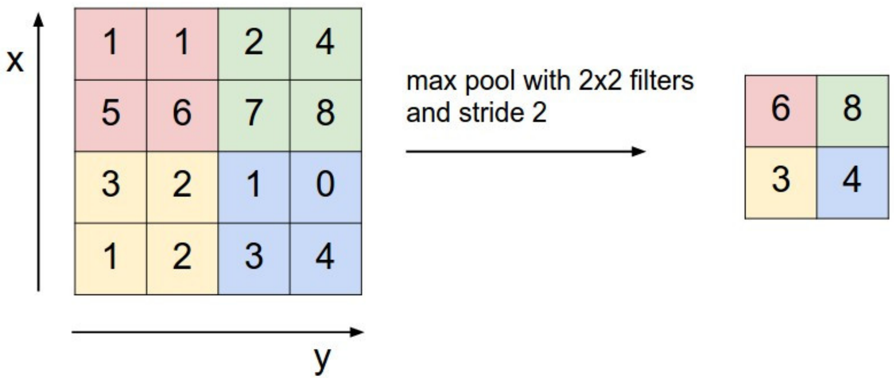


Figure 3: max pooling being applied to a 4 × 4 tensor. Source[10]

2.5 Fully Connected Layers

A fully connected layer is a layer that has a connection between every neuron in the previous layer and every neuron in its layer. The purpose of a fully connected layer in a CNN is to use the features extracted from the previous layers to classify the input data. In Dragnet, 1024 neurons are created in the first fully connected layer (layer 3). Now, instead of using convolution and ReLU, matrix multiplication and ReLU is done with the previous layer so to establish the connections. Within the fully connected layer, pooling is not applied as conceptually the desired level of abstraction on the input has already been achieved. Like the first two layers, the fully connected layers will have weights and biases applied to the neurons which are also TensorFlow variables tuned during the training steps.

Before connecting the first fully connected layer (layer 3) to the second fully connected layer (output layer/layer 4), a dropout operation is performed. Dropout is used to prevent over fitting by randomly turning off neurons throughout the training process. This simulates training many neural networks simultaneously and taking the average of the results. A more in-depth analysis of dropout can be found in the article *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*[15].

The second fully connected layer (the output layer) is the final layer in the network. In Dragnet, 76 neurons are created as this is the amount of classes the CNN classifies data into (uppercase/lowercase Cyrillic letters and numerals). The output layer provides the normalized probabilities that a given input belongs to the different possible classifications.

2.6 Training Dragnet

To train a neural network a large amount of labelled data is required. In the case of Dragnet, circa 14000 images were prepared. This data was split into 80% training data, and 20% testing data. As data is fed into the neural network, the weights and biases in the different layers are tuned to reduce their error using an Adam Optimizer[16] provided by TensorFlow. The goal of the optimizer is to reduce the error as much as possible, as the smaller the error is, the more accurate the classification of the data tends to be. Error is measured using a cross entropy[17] function provided by TensorFlow.

After training is complete, the state of the CNN is saved so that input can be fed through it later for classification, or so the CNN can be trained further.

Dragnet achieved a consistent accuracy of 99.2% on the training/test data split.

3 Preparation of Training Data

For any neural network to be accurate in classifying data it is necessary for it to have been trained on some labelled training data. In order to train Dragnet a large set of black and white images of Cyrillic characters were created. These images were then converted into vectors containing real values between 0 and 1 with increments of 0.1. These values represent the color of the pixel where 0 is complete white and 1 is complete black.

To produce these images the python library Pillow was used to read a directory of fonts and then produce an image for all the uppercase/lower case letters and numerals of the Cyrillic alphabet for each font. Once this initial set of images was created a series of shears, blurs, and rotations were applied to each to mimic what one could expect to find on documents that had been thrown into a lake and then scanned. An example of a character with the various transformations applied to it can be see in Figure 4

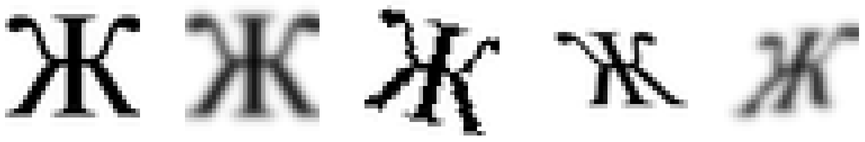


Figure 4: A Cyrillic chracter with blurs, shearing, and rotations applied.

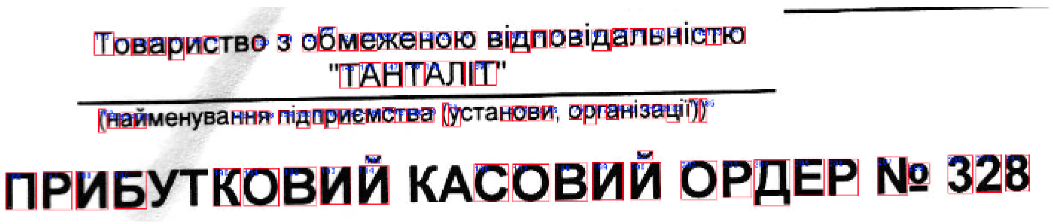
With 5 fonts, 76 characters, and various transformations applied, the resulting dataset consisted of circa 14000 images.

4 Preparation of Real World Data

When working with real world data and a neural network it is necessary that the neural network can understand the data. In some cases the neural network can work with the data directly but where that is not the case pre-processing is required.

The real world data to be processed by Dragnet are scanned images of documents with Cyrillic characters. In order for Dragnet to be able to do anything with this data, the characters in this document first needed to be converted into vectors of real values as described in section 3.

To prepare the data for Dragnet, an ImageMagick script is used to recognize bounding boxes of connected regions within a image of a document. The contents of these bounding boxes are then extracted using a python script and converted into vectors of real values. Using a heuristic as basic as looking for connected regions in an image is not advanced enough to completely avoid recognizing two characters as one, or to recognize a character touching a line or some artifact in the image. For this reason some basic heuristics were also considered like the ratio of the height and width of a bounding box. An example of the bounding boxes recognized in an image after some basic heuristics can be seen in Figure 5.



Товариство з обмеженою відповідальністю
"ТАНТАЛІТ"
(найменування підприємства (установи, організації))

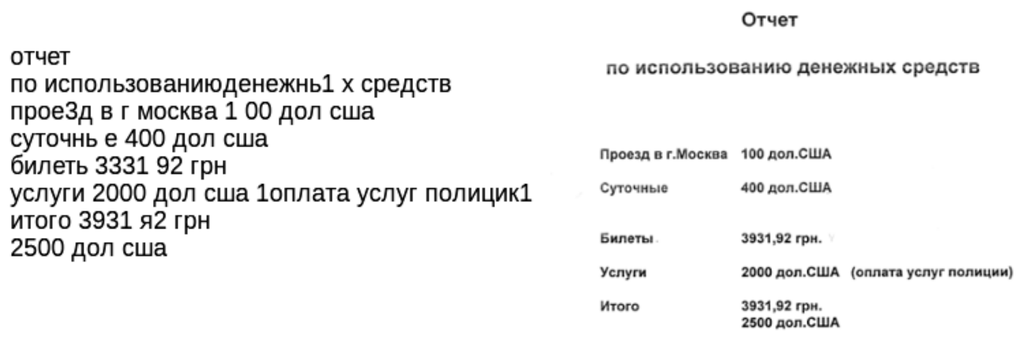
ПРИБУТКОВИЙ КАСОВИЙ ОРДЕР № 328

Figure 5: The bounding boxes of connected regions recognized by the ImageMagick script used in Dragnet.

The python script used to extract the contents of the bounding boxes also pays attention to the vertical relationship of these bounding boxes. In addition to vectors produced, a file containing what lines characters should be on in the assembled document is produced. This allows for reconstruction of the document in a manner that roughly resembles the scanned image of the document.

5 Reconstructing Documents from Dragnet's Output

In order to construct a text document from a scanned image, the corresponding vectors in section 4 need to be fed into Dragnet. Dragnet will then produce an integer label for each of these vectors that corresponds to a character. Using this information and line information for each character, a text document can be constructed. An example can be seen in Figure 6.



```
отчет
по использованиюденежнь1 х средств
прое3д в г москва 1 00 дол сша
суточнь е 400 дол сша
биletь 3331 92 грн
услуги 2000 дол сша 1оплата услуг полицик1
итого 3931 я2 грн
2500 дол сша
```

Отчет

по использованию денежных средств

Проезд в г.Москва	100 дол.США
Суточные	400 дол.США
Билеты	3931,92 грн.
Услуги	2000 дол.США (оплата услуг полиции)
Итого	3931,92 грн. 2500 дол.США

Figure 6: A text document produced by Dragnet beside the corresponding scanned document.

6 Results and Conclusion

Measuring the accuracy of Dragnet on real data required the work to be done manually. This had to be done manually as the intention was to only measure how accurate Dragnet was at classifying characters it recognises. That is, if an artifact in the image or some punctuation was fed into Dragnet, there isn't any expectation of it producing the correct answer and so such input isn't considered in measuring Dragnet's accuracy.

To test Dragnet, 7 documents were chosen from the YanukovychLeaks Project. The results are described in table 1.

Document nr.	Characters count	Errors count	Accuracy (%)
1	204	14	93,1
2	302	91	69,9
3	540	47	91,3
4	481	21	95,6
5	382	31	91,9
6	409	11	97,3
7	149	4	97,3
Total	2467	219	91.1

Table 1: Dragnet's accuracy on sample documents

For characters that Dragnet was taught to recognize, it managed to classify the characters correctly a little over 91% of the time. When compared to the expected results of training/testing with the data set in TensorFlow's MNIST tutorial[18], a typical introductory CNN, 91% isn't particularly high. However, when comparing the output Dragnet produced against a given document from the YanukovychLeaks against the output from the GNU OCR tool, the results were very comparable on average. Given the condition of documents found in the YanukovychLeaks can be quite poor, it is difficult in general for an OCR tool to perform perfectly.

One apparent weakness of Dragnet as an OCR tool was its inability to consistently discern uppercase from lowercase letters. The CNN was still classifying what it was provided with correctly, but the scripts used to extract characters drew bounding boxes tightly around the character. If the lowercase version of a character was just a scaled version of its uppercase counterpart, the CNN was not able to detect this from the image alone.

Dragnet's ability to classify characters more accurately with real data could be augmented by supplying more varied training data to train the CNN with. In the context of machine learning, 14000 pieces of data is still quite small. The results of the Dragnet as an OCR tool could be made more impressive by improving the scripts used to extract character regions from a scanned document. While this task is not that of the CNN, it is the task of a comprehensive OCR tool.

In conclusion, using a CNN for Cyrillic OCR can yield readable and meaningful results.

REFERENCES

[1] *Organized Crime and Corruption Reporting Project.* URL: https://www.occrp.org/en.

[2] *YanukovychLeaks National Project.* URL: http://yanukovychleaks.org/.

[3] *Ocrad - The GNU OCR.* URL: https://www.gnu.org/software/ocrad/.

[4] *Wikipedia - Dragnet (Policing).* URL: https://en.wikipedia.org/wiki/Dragnet_(policing).

[5] *TensorFlow - An open-source software library for Machine Intelligence.* URL: https://www.tensorflow.org/.

[6] *ImageMagick.* URL: https://www.imagemagick.org/script/index.php.

[7] *The friendly PIL fork (Python Imaging Library).* URL: https://github.com/python-pillow/Pillow.

[8] *Dragnet source code.* URL: https://github.com/jamesrobb/dragnet.

[9] *Wikipedia - Convolution.* URL: https://en.wikipedia.org/wiki/Convolution.

[10] *CS231n Convolutional Neural Networks for Visual Recognition.* URL: http://cs231n.github.io/convolutional-networks/.

[11] *Tensor Ranks, Shapes, and Types.* URL: https://www.tensorflow.org/programmers_guide/dims_types.

[12] *Variables: Creation, Initialization, Saving, and Loading.* URL: https://www.tensorflow.org/programmers_guide/variables.

[13] *Feature extraction using convolution.* URL: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution.

[14] *Wikipedia - Rectifier (neural networks).* URL: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[15] Alex Krizhevsky Ilya Sutskever Nitish Srivastava Geoffrey Hinton and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* (2014). URL: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

[16] Diederik P. Kingma and Jimmy Lei Ba. "Adam: a method for stochastic optimistization". In: *Adam: a method for stochastic optimistization* (2015). URL: https://arxiv.org/pdf/1412.6980v8.pdf.

[17] *Wikipedia - Cross entropy.* URL: https://en.wikipedia.org/wiki/Cross_entropy.

[18] *MNIST For ML Beginners.* URL: https://www.tensorflow.org/get_started/mnist/beginners.