OCR - Optical Character Recognition Technical Report

1. Problem presentation

Over years, people have been thinking of ways to incorporate human functions to computers and this has become an interesting and exciting research field. One of them is the machines' ability to "read" and interpret printed textual documents. With this project we propose to solve the OCR – optical character recognition – problem, which basically consists of converting an image of some text, be it handwritten or typewritten, to computer-encoded text. Usually, OCR refers to typewritten text, with the input given as on character (glyph) at a time, and sometimes having typewritten text fragments, the difficulty coming from separating the words (OWR – optical word recognition, but still referred as OCR). But what we actually want to solve is the ICR problem – intelligent OCR – which means that we want to translate an image of handwritten text to machine editable text.

2. State-of-the-art

Most solutions can only operate on a typewritten text, as it is easy to separate the characters, since there is a visible separation between the pixels of different juxtaposing characters. Popular solutions are:

- *Ocrad – The GNU OCR (https://www.gnu.org/software/ocrad/ocrad.html)*: it is an open-source OCR program developed by GNU, based on feature extraction methods. It includes a layout analyzer for separating columns or blocks of text normally found on printed pages. It can be used a stand-alone console app, or as backend to other programs. Its main disadvantage is that it can only read .pbm (bitmap), .pgm (greyscale) or .ppm (color) image formats, and, of course, it can only process typewritten text, but its strength comes from being very fast compared to other OCR solutions. It uses ad hoc algorithms for the recognition, basically comparing the shape differences of the characters, mostly like a binary search, one glyph at a time.

- *OCRFeeder (https://github.com/GNOME/ocrfeeder)*: is an open-source document layout analysis and optical character recognition application, which means it can process documents and convert them to many formats. It also has a GUI and command line interface for easy use. It is superior to *Ocrad* in that it can convert any type of image to the desired format, but it can still only process typewritten text. It actually uses as an engine for character recognition the *Ocrad* engine, or an engine called *GOCR (GNU OCR)* which is very similar to *Ocrad.*

- *Scan2CAD (https://www.scan2cad.com/)*: is a program developed by *Avia Systems*, used for raster and vector conversion and editing CAD (Computer-Aided design). It is not an open-source program like the ones enumerated; it only has a limited time free trial. As the name suggests, it is mainly used for scanning images for CAD software, but it can be used as an OCR and ICR too. Its main advantage is that it can also detect handwritten text, not only printed text. It uses feature extraction like *Ocrad* and *OCRFeeder*, but it also uses neural networks for detecting the corresponding character written.

  Other notable OCR programs are *PDF Scanner, Adobe Acrobat Reader* and *Google Docs*, and ICR programs: *ABBYY* and *TeleForm*.

Important names in the fields and research teams

- The earliest ideas of OCR come from the year 1870, from Charles R. Carey retina scanner, which was similar to an OCR; Fournier d'Albe's Optophone and Tauschek's Reading Machine are the first devices to help the blind read.
- The Intelligent Machines Research Corporation is the first company which creates OCR tools and sells them.
- John Linvill invents the *Optacon* (Optical to Tactile CONverter) is similar to the Optophone, but can read printed material that has not transcribed into Braille.
- Modern Optical Character Recognition research teams come from the companies selling such software, like *ABBYY, Tesseract, Adobe Acrobat Reader* and *Google Docs*.
- New research can now be done in the handwritten optical character recognition, as the *MNIST* database *(http://yann.lecun.com/exdb/mnist/)* has been created; it consists of handwritten digits, but it was later extended to handwritten letters too.

3. Our solution

The steps

The user uploads an image containing the handwriting that they want to convert into computer-encoded text. The image is taken over by the NodeJs server which sends it to backend. After the image preprocessing, a Python Flask server works as a service and is responsible of returning the result (the recognized text or an error message in case of problems). It helps us to keep the neural network in memory for faster image processing.

The image preprocessing

For the image preprocessing, our aim is to get each letter separated from the others. In order to do this, we followed the next process. First, after the image is received, it is converted to grayscale and after that it is converted to binary scale. The next step is to get the contours using the convex hull. Depending on the contour and the x coordinate, we can establish whether there is a blank space following. Afterwards, we sort the contours by the x coordinate and for each one of them we extract from the image a rectangular area which contains the whole contour. We then resize each image to 28*28 and send it to the neural network.

The neural network

The artificial neural network is a network of simple elements called artificial neurons, which receive input, change their internal state (activation) according to that input, and produce output depending on the input and activation. In this solution, we used a convolutional neural network, since they are great for capturing local information (such as neighbor pixels in an image) as well as reducing the complexity of the model (faster training, needs fewer samples, reduces the chance of overfitting). The model that we used has the following structure:

- A convolutional layer with the relu activation function
- A second convolutional layer with the relu activation function
- Applying max pooling
- Applying dropout with a 0.25 rate
- Flattening
- A dense layer with the relu activation function
- Applying dropout with a 0.5 rate
- The final dense layer with the softmax activation function

The chosen optimizer is Adadelta. The aim of the optimizer is to minimize the cost function and adjust the parameters of the the neural network: the weights and the biases for the layers. We chose Adadelta because it is based on Adagrad and it seeks to reduce the monotically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w. Therefore, it is one of the most efficient optimizers.

The number of epochs is 25, the cost function is categorical crossentropy and the since we chose the batch approach, the batch size is 256.

Used technologies

For the frontend component we used AngularJs, Bootstrap and HTML. For the backend components we used Python with Keras library for the neural network and OpenCV library for image processing.

4. Results

The application has been evaluated using several images and it had successful results. For an image, the time required for processing and for receiving the final text is about 2-3 seconds. As for the accuracy, our neural network model proved to be very effective, achieving an accuracy of 90%.

One important step was adding the second server, the one using Flask, which brought big advantages, such as being able to keep de model in memory. This is important because of 2 reasons: the model can be easily changed, and the time required for returning the final text is faster now. In the first version, the time required was about 10

seconds, while now it only needs 2-3 seconds. In terms of web responses, this is a very nice improvement.

Another important phase in developing the presented solution was the creation of the model. At the beginning, we used a classical neural network model, without convolutions. After several versions and changes, the accuracy was around 87%. Then, we created a new model, based on convolutional neural networks and the results were better, obtaining 90% accuracy.

For the activation functions, the best results were for the relu – softmax combination: using the relu activation function on the inner layers and the softmax activation function on the last layer. One reason why we chose relu instead of sigmoid is that relu is much faster.

Another improvement that we noticed was using the dropout layers. It refers to dropping out units (both hidden and visible), resulting in the reduction of overfitting by preventing the network to be too specialized on the training data.

5. Comparison with other solutions

The OCR solution we provided, while very similar to other OCR readers, is in fact an ICR reader, which means it can also read hand-written characters. Little to no results has been studied on this matter.

The advantages of our solution are the possibility to recognize hand-written characters, using neural networks, and being an easy-to-use online tool, where you just input your picture and have the text outputted to you -> no need to install heavy packages, tools, libraries, etc.

The disadvantages are that it doesn’t check big word dictionaries for the resulted words for similarities. Also, it doesn’t check first if the text is type written, as type-written characters are very easy to recognize for the OCR problem, so software like ABBYY Company’s OCR report an accuracy of almost 100% for texts in the English language.

So, as we said, even though our solution has an accuracy difference of about 10%, in comparison with other OCR tools (like ABBYY), our solution handles hand-written text too, so the accuracy difference is almost irrelevant.

6. Future work

As for future work, one idea would be to transform the application into a smartphone application. For this, we could also combine it with a camera app that would instantly take photos of hand writing and then convert it to text. The app could also have some improvements: possible filters that would make the processing easier such as increased contrast, luminosity and clarity.

7. Conclusions

For this project, we proposed an OCR implementation in the form of a web application, using a convolutional neural network. Our implementation offers high performance and accurate results and it is therefore a valid solution to the OCR problem.

8. Relevant links, Resources and tools available

- Python for the implementation (*https://www.python.org/downloads/*).
- Neural Networks for character recognition: Razvan Benchea's *Neural Networks* and *Advanced Chapters of Neural Networks* courses (*https://sites.google.com/view/rbenchea/*)
- EMNIST Database *(https://www.nist.gov/itl/iad/image-group/emnist-dataset*).
- A possible schema for the algorithm: *https://ijret.org/volumes/2015v04/i01/IJRET20150401062.pdf*.
- *https://docparser.com/blog/what-is-ocr/;*
- Ravina Mithe, Supriya Indalkar, Nilam Divekar, *Optical Character Recognition* (2013);
- Karez Abdulwahhab Hamad, Mehmet Kaya, *A Detailed Analysis of Optical Character Recognition Technology* (2016);
- Nikhil Pai, Vijaykumar S. Kolkure, *Optical Character Recognition: An Encompassing Review;*