A Tutorial on Optical Character Recognition in Mathematical Domain

Oleh Onyshchak [1]

Abstract

Optical Character Recognition(OCR) is one of the earliest addressed computer vision tasks. But there are few solutions available for its application in specific domains such as parsing of mathematical formulas. Thus in this paper we will solve this problem in a simple educative way, providing a comprehensive introduction into Computer Vision(CV) field with a possibility to extend the base solution at the end. The resulting program incorporates segmentation of input image into characters and then character recognition itself based on Convolutional Neural Network(CNN). The main motive of the paper is to give a self-containing introduction into CV field.

1. Introduction

With computational resources and storage getting cheaper and cheaper, a window of possibilities opens for searching, reusing, and processing information. Thus everything around gets digitized, including all kind of paper documentation. And till now most of that work is done manually, which is extremely expensive. And for mathematical papers specifically, it becomes even more time-consuming, since it's far harder to type a mathematical formula rather than a regular word.

For that reason, hundred and hundred years of math history is still stored in paper format exclusively and cannot be processed efficiently. And while we already have comprehensive tools to parse scanned text[7], but it's not applicable to math expressions, which has a big complexity for parsing. Although there are some decent tools[8], they are proprietary and have limited functionality, in particular, it cannot parse multiline expressions or even the entire document. For that reason, we will be working on an open-source project to parse math expressions so that it could be freely reused and help developing tools, which can convert the entire math paper at once. Only then we will have a dramatic effect of digitizing all math literature.

But since it's a project in the field of OCR, which is a good way to start for newcomers in Computer Vision, we decided to describe our project in self-contained and educative manner. That will allow the reader to enter the field of CV smoothly by exploring and playing with our solution. We will also encourage readers to contribute to the project, which they will know by heart until the end of the paper.

We will assume that a reader has basic background in machine learning and geometry.

2. Problem Definition

Right now, Optical Character Recognition is a solved task for a regular text. We have plenty of approaches and available solution to parse entire files with either handwritten or scanned typed text.

But for some specific class of documents, such as math papers, the problem still exists. The main complexity here is that parsing math expression is untrivial, since, depending on the context, some character will overlap or appear on the top of each other, or even span multiple lines. Thus we want to create an open-source project tackling this problem so that it could be reused by a community and eventually move us a little closer to the destination of the document-wise parser of math papers.

3. Related Work

From the solution perspective, we need to mention that there is a commercial application crafted for the same domain - Mathpix[8]. Although, it's source code isn't freely available, and it's restricted to parsing one statement at a time. Also, there is an OCR library Tesseract[7] open-sourced by Google, but it's for a parsing a regular text and thus not applicable for the math domain.

From the perspective of implementing the solution from scratch, there are two main approaches: deep learning and classical one. The classical approach mainly consists of preprocessing the image, contour detection, and character recognition subtasks. But it's a hard task to perform a contour detection in a generic use case since background and font density can vary. On the other hand, the deep learning approach solves those problems. In particular, we can use

[1] Ukrainian Catholic University, Lviv, Ukraine. Correspondence to: Oleh Onyshchak <o.onyshchak@ucu.edu.ua>.

a Recursive Neural Network with Connectionist Temporal Classification[11].

But in our case, we won't be dealing with images in the wild but only math expressions on the paper/whiteboard. In our domain we also have a simplification that all symbols are written fairly separately to each other, comparing to a regular text. So it's feasible to configure contour detection with a solid performance. And since the simpler solution we equally solve our problem, it was rational to move forward with a classical approach.

4. Solution Architecture

As we mentioned shortly in the previous section, in a domain of mathematical formulas, we always have a consequence of either digits, or alphabetical characters, or mathematical operators. Any of those are written separately to each other, when comparing to simple text, and so it will be efficient to use classical approach for the recognition task. That is:

- image preprocessing
- segment line into separate characters
- perform character recognition

5. Data

As for any machine learning problem, acquiring and preprocessing of relevant data is one of the essential parts for overall project success. On first glance, CROHME's dataset[1] is the most appropriate choice. It contains over $100,000$ images of handwritten math operators, English and Greek alphabet characters. But after more thorough exploration, you can find that nearly 75% samples are duplicates. Also, the dataset is unbalanced; thus after clearing it from duplicates, some characters have only a few hundred examples, which is clearly not enough.

At this point, we will get help from the famous MNIST dataset[2], or rather its extended version with English alphabetical characters[3]. It's a well-made dataset containing more than $150,000$ images of handwritten alphanumerical characters, stored in CSV file format. Thus we might convert pictures of math operators from the first dataset into the EMNIST format to have a complete dataset.

5.1. Data Preprocessing

So after we identified and removed duplicates in CROHME dataset, we need to add its math operators to EMNIST dataset. To do so, we randomly select 2800 images of each symbol. First 2400 will go into train dataset, other 400 - into test dataset. Then we resize each image to 28x28 pixels and convert it to a CSV format.

Those were preprocessing techniques applied instantly, but there also a few of which were discovered only after long troubleshooting of problems with poor performance. Those are:

- Binarise images: EMNIST dataset has greyscale samples. And when model trains on it, it's put a lot of attention to patterns in the colors rather than to patterns in character shapes. For that reason, formulas written with the simple black pen were poorly recognized. To correct this issue, we binarise images: all pixels with color bigger than 128 were set to 1, and others - to 0.
- Dilute images: CHRONME dataset has examples with far thinner ink, than in EMNIST. Also, it has bigger images, and thus we need to downsize them while converting. Two factors combined created samples with a lot of pixel artifacts in the enriched dataset. And for that reason, our model has extremely poor performance on characters converted from CHRONME dataset. To solve this issue, we dilated images. That is we scan the entire image and whenever we encounter a border between black and white pixels, we swap some white pixels to black, making the final symbol bolder.
- Rotated images: After a long investigation of why the model cannot parse a single character correctly, it was discovered that EMNIST dataset stores its images in rotated and mirrored format. Thus to fix this problem, we rotate each image by 270 degrees and then mirror it in respect to Y-axis. To perform those operations on the entire dataset at a time, and thus make it far efficient, we implemented our versions on rotation and mirroring for a matrix of flattened images (1 image = 1 row).

6. Algorithm

6.1. Line Segmentation

First of all, we need to preprocess image properly[5]:

- Image binarisation. Just the same as we do in data preprocessing for character recognition, here we need to make model generic to any medium, type of ink, etc. To do so, we force the model to learn only based on the shape of symbols by converting the image to greyscale and then binarizing it.
- Noise removal. We might get a lot of noise on input images such as accidental dots lines made by ink during writing, or some artifacts on the background. Thus before segmentation, we need to remove that meaningless noise so that it won't confuse our model. To do so we perform an opening: an erosion followed by dilation[4].

To put it simply, those are two polar operations. After identifying a border between 0 and 1 pixels, erosion swaps some 1 to 0, while dilation - some 0 to 1. Thus firstly we are making any meaningful character thinner by erosion and then trying to restore its original shape with dilation. On the other hand, a small noise will disappear after erosion, and then on the dilation phase, there will be nothing to enlarge. And thus at the end, we will remove the noise from the image

- Skew Correction. So in order for both: segmentation and recognition work properly, we need to make sure that text is horizontal. To do so with a single line, we will need to find the smallest bounding box around all characters, i.e. the smallest box which contains all black pixels. After that, we will rotate the entire image so that sides of a bounding box are parallel to axes of the image. The above condition might be satisfied by rotating the images by either α, or $90 - \alpha$ degrees, but it's imperative we always choose the smaller angle so that we won't end up with a vertical text.

Now that we properly processed image, we can parse it character by character[9]. We do that by identifying all contours we have on the image, using the following algorithm[10], and when we have a bounding box for each symbol, we might extract them one by one and save into separate files.

In the about workflow, we also developed some specific features to identify symbols consisting of multiple contours. For example, ”=” symbol contains two different contours of the ”-” sign. So once segmentation is done, we check whether there are two contours on top of each other. If so, we merge them as a single ”=” symbol.

And that's how our segmentation works, which was build on top of this script[13]. It gets an image of a math expression as an input, and produces an image per character as output. Later on, the character recognition module will use this output as its input and produce the parsed string.

6.2. Character Recognition

For character recognition, the most frequently used tool is a neural network, since they are the best option when one need to distinguish some patterns. And we will use a basic version of neural network popular variation - convolutional neural network(CNN). If you are not familiar with the basic theory of CNN, we recommend watching this material[6] and also any other you will find on the Internet. But in any case, below, we will give a very high-level explanation of why it works with computer vision better than a classical neural network.

So in the classical neural network, we have an input and output vector. Our input will be the flattened image, that is if we have a 28x28 image represented as a matrix, we can also express it as a vector with 784 elements, where first 28 items equivalent to the first row of a matrix, the second 28 items - to the second row, etc. But with this approach, we are not enforcing our model to work on the level of some groups of related pixels but rather as an independent set of 784 input signals. While in real life, a pixel just above or below has a very strong relationship with each other, while those influence far lower with pixels in another part of an image.

And convolutional neural network solves this issue by working one level higher than merely on the pixel scale. It transforms the image by scanning it with a rectangular kernel, thus learning from the patterns on a level of that rectangular window sliding through the image. It also can reduce the dimension of input image following some rule, e.g., replaces 2x2 part of the image with a single pixel, which value equals the maximum from 4 pixels in the original image. After that, we can scan the picture again with a rectangular window and learn some more high-level features.

Convolutions allow us to learn a pattern depending on the geographical location of images. And pooling enables us to get rid of some local and specific patterns and work with something more high level as corners and lines of an image.

In our implementation[12] we used the same ideas. The first is the convolutional layer, which is like a set of learnable filters. There are 32 filters for the two firsts convolutional layers, and 64 filters for the two last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter.

It is followed by a pooling layer, which decreases the dimension of the image. These are used to reduce the computational cost and, to some extent, also minimize overfitting. Moreover, it's useful to allow consequent convolutional layers to learn more high-level features.

Also we have dropout layers, which are used to randomly ignore some fraction of output nodes from previous layers. This way we force the network to learn features in a distributed way, which, in turn, improves generalization and reduces the overfitting.

And at the end, we flatten our final feature maps into a one-dimensional vector, which will combine all founded local and global features. And then based on that feature vector, we output a vector where i-th elements is the probability of i-th character being on the input picture

During this process, we also do standard improvements to the quality of the model and the training time. For example, we can decrease the latter by using a dynamically-changed learning rate. In the beginning it will be bigger, and when the precision of the model doesn't increase significantly, we'll decrease the learning rate by a factor of two. What concerns quality enhancements, here we incorporate data augmentation as well. Although, we only use a little zooming, horizontal and vertical shifts, and a small-angle rotation so that not corrupt the image. That is, if we applied vertical flipping to "6" symbol, we'd get a "9" symbol with the old label. Or if we rotate "l" too much to the left, we will get something more similar to division sign "/". All those confusions will decrease the precision of the network.

7. Evaluation

Here we will demonstrate the algorithm described above with some specific example. You can also reproduce the results and examine source code by copying the project's repository:

https://github.com/OlehOnyshchak/OCR

We strongly encourage the reader to contribute to this open-source project by enhancing the existing functionality or developing the new one, described in the section "Future Improvements".

Firstly, let's discuss how we evaluate the performance of each module and then show how the complete pipeline works on a specific example.

The first module of image segmentation is not a machine learning algorithm, so we don't have data to measure its performance. In the future, we might develop some way to automatically measure the performance of this algorithm on some dataset, but currently, it was out of the scope of the project.

The second module is a character recognition part. Here we have a supervised machine learning problem with a labeled dataset. Thus we are evaluating the module by checking its precision on test subset of the dataset. The final precision is 88%. You can also see train&validation accuracy on each epoch of network training in Figure 1.

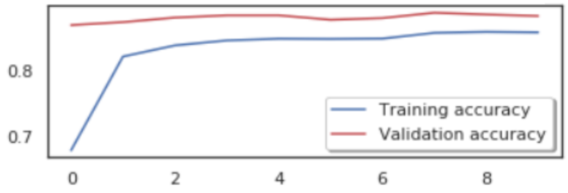

Figure 1. CNN accuracy

And now let's look at a specific example of parsing $x^2+3x+5=0$, which you can see in Figure 2. In the file Main.ipynb you could find the main pipeline of our project, where we will specify the file path to the image we are parsing.

After that, we properly preprocess the image as was discussed in the "Algorithm" section and perform character segmentation, output of which you can see in Figure 3.

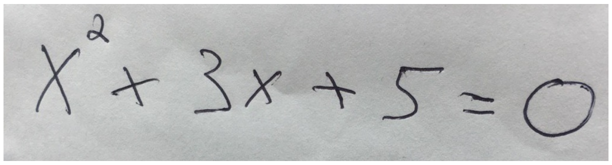

Figure 2. Input image

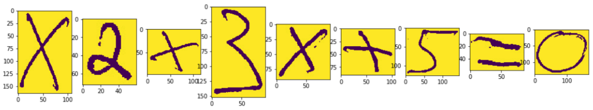

Figure 3. Results of segmentation

Then we process each identified character one by one. That is, we convert them to EMNIST format and then feed them to our Character Recognition module. After that we map character's class to its Latex representation, concatenate all character together and output the final result of our solution. In this case, program will output $x2+3x+5 = 0$. As we can see, we correctly predicted all characters, but the program still lacks the functionality of identifying superscripts.

8. Future Improvements

Since the topic has a lot of complications, there are multiples possibilities to improve the solution. Currently, it's working in its basic form, that is parsing a single-line formula with brackets, subtraction, addition, multiplication, and division.

The next improvement should be the ability to parse subscript and superscript properly. As a functionality which correctly identifies "=" sign, that extension requires only pure Computer Science enhancements. That is, we already have a functionality, which determines the bounding box for each character. Now if we draw a horizontal line in the middle of a total bounding box, we could identify sub- and superscript. That is, when a bounding box is crossed by that line; then it's a pure character. Otherwise, it's superscript or subscript character when bounding box is above or below the line respectively.

Besides that, there plenty other more complicated features that can be added. Starting from something relatively simple as identifying a fraction, where we could also use a calculation based on bounding boxes, or parsing trigonometric operators to something more complex such as $\sum_{i=1}^{n}$, or multiline system of linear equations.

We could also improve the quality of existing functionality. While string segmentation works precisely, we can extend it to work with formulas in the wild(whiteboard, street art, etc.) as well. And what concerns character recognition module, here we have a huge room for improvement, since its precision is only 88%, where most of the precision was lost due to problems with identifying visually similar characters such as ”1” and ”l”.

9. Conclusion

So in this work we tackled the problem of optical character recognition in the math domain. To do so, we incorporated two fundamental techniques of that field: character recognition and string segmentation. Former gave us experience in applying classic machine learning techniques with CNN, while in latter we used a lot of graphical algorithms from computer science field. But the most efforts, improvements, and troubleshooting were dedicated to data preprocessing part. Since not only we were required to adjust the dataset to a specific need of our project, but also merge two different datasets, which opened us another huge task of adjusting one dataset format to another.

As for someone with no prior ML experience, this project was extremely valuable in learning basic but vital parts of most computer vision problem. It gave us a broad experience of computer vision tools: preprocessing of image database, graphical algorithms, and neural network. And that is the reason, why we decided to format this report in a self-containing and educative manner. Since we grabbed so broad and fundamental experience, we wanted to share it to other newcomers into ML and CV in particular.

Besides that, we are going to continue the work on this project, which will give us the ability to create a second part of the tutorial, and thus we will produce something more applicable to real-life problems. In that way, our solution will be useful not only as a learning example or repository where anybody could reuse some of the basic functionality for OCR, but also will be valuable as a complete solution on its own.

References

[1] Handwritten math symbols dataset, Kaggle. https://www.kaggle.com/xainano/handwrittenmathsymbols

[2] The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/

[3] EMNIST (Extended MNIST), Kaggle. https://www.kaggle.com/crawford/emnist

[4] Eroding and Dilating, OpenCV. https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[5] Improve Accuracy of OCR using Image Preprocessing, Medium. https://medium.com/cashify-engineering/improve-accuracy-of-ocr-using-image-preprocessing

[6] MIT Introduction to Deep Learning 6.S191: Lecture 3. https://www.youtube.com/watch?time_continue=1&v=NVH8EYPHi30

[7] Tesseract Open Source OCR Engine, GitHub. https://github.com/tesseract-ocr/tesseract

[8] OCR application in Math domain, Mathpix https://mathpix.com/

[9] Equation OCR Tutorial Part 1: Using contours to extract characters in OpenCV. https://blog.ayoungprogrammer.com/2013/01/equation-ocr-part-1-using-contours-to.html/

[10] Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)

[11] Build a Handwritten Text Recognition System using TensorFlow, Medium https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using

[12] Introduction to CNN Keras, Kaggle https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6

[13] Word Segmentation, GitHub https://github.com/githubharald/WordSegmentation