Real time symbol detection using neural networks

Patryk Uchman
University of Warsaw
pu347272@students.mimuw.edu.pl

Paweł Uchman
University of Warsaw
pu384374@students.mimuw.edu.pl

Abstract

In this paper we explore methods for detection of handwritten mathematical symbols. In our approach we can identify 2 distinct methods to accomplish this task. They differ in a way of how we find the locations of the symbols.

The first one is based on manual image transformations using numpy package and openCV library in order to end up with an image containing black symbols on white background.

The other one utilizes a convolutional implementation of sliding window algorithm which slides a box of fixed size over the image and classify each box as one of the symbols or the background.

We found that the image transformation approach led to the most accurate results, that could generalize for different lighting conditions much better than the other methods.

The sliding window approach would not generalize well due to the difference in real handwritten data and the data that we produced in the process to feed to the classifiers.

1. Introduction

Object detection is a well-known problem of image processing. The goal is to find all the occurences of an object of interest in a given image. In the past this task has been achieved using machine learning approaches, that often included time consuming feature engineering. With the advent of deep learning new methods arise continuously. This work is an attempt to compare one of those methods to the older more manual approaches.

In this work we applied 2 different techniques for the problem of symbol detection. This problem has a relatively well-defined structure and it is easy to gather lots of data. It is also possible to generate artificial data, based on the data already possessed. We also assume that dark ink at a relatively white background. We also made our solutions work in real-time.

2. Related work

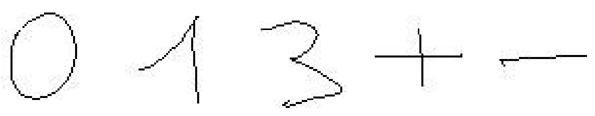
There seems to be some implementations of our image transformation approach available on the internet. We, however, have decided not to copy or utilize those solutions in our work and decided to come up and implement our own. Also, we have not found an application of the latter approach for the problem of symbol detection.

3. Data

As far as the data is concerned, we are using symbols from kaggle competition: https://www.kaggle.com/rtatman/handwritten-mathematical-expressions. The images in the dataset are of size 45x45 pixels and are represented in a grayscale.

The dataset is unbalanced. Some of the classess are underepresented while others are overrepresented. As an example the number of images for '/' is 199, while the number of images for "x" is 3251, while the number of images for '1' is 26520. We worked on a subset of 10% of all the data, due to the fact that it improved our iteration time and still offered impressive results on the test set.

In the sliding window approach we also generated a dataset of backgrounds: patches of 45x45x3 images of white paper(plain or checked). We use it in order to "paste" the symbols on top of the image of a background to simulate a real-life scenario. We gathered it by moving an empty, plain or checked paper in front of a camera, while extracting some of the patches. We gathered over 10k patches over a period of a few minutes in different lighting conditions. Below are some examples of symbols from the dataset



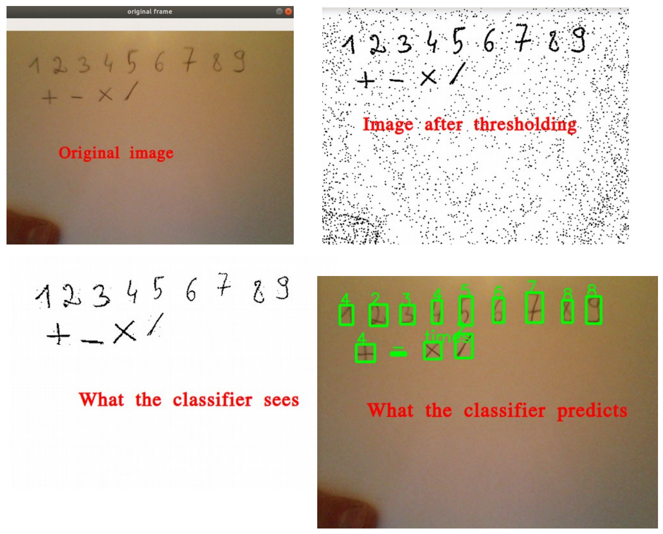
4. Methods

We use 2 different methods for the problems: one based on custom image processing and sliding window algorithm.

a) Custom image processing

In this approach we process the image with subsequent transformations, to end up with an image where the background is white, and any area that is not completely white is treated as a candidate for a symbol.

We start off by converting our initial image into grayscale and apply gaussian thresholding. This step extracts the most relevant image features but introduces some noise. We bolden(erode) the dark parts of an image and then filter out the irrelevant areas by thresholding it with the experimentally found value. For the remaining areas we rescale them into a fixed size of 45x45 and use a CNN classifier to predict it’s class.



b) Sliding window

In a sliding window approach we slide a window of a fixed size 32x32 over the image and predict the class for the content of the window. This naive approach would be too slow for real-time processing and so we use a convolutional implementation of the sliding window algorithm effectively predicting classes of many windows at once. Since the symbols can vary in size, we can utilize many different window’s sizes’ for detection.

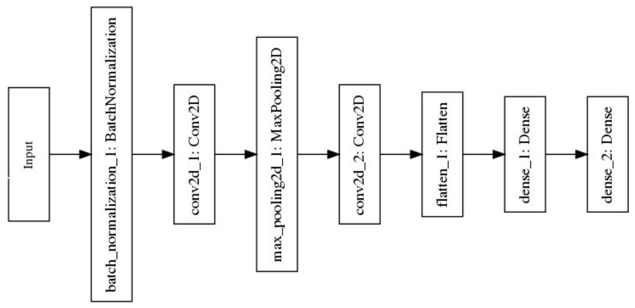
Changing the dimensions of the window would lead to a change in network architecture, and so instead changing the actual window size we reshape the image, so that in a new image the same 32x32 window corresponds to our target window’s shape. We filter out windows with prediction confidence under certain experimentally chosen value, and windows that overlap each other, choosing the one with higher confidence as the prediction.

5. Experiments

a) Custom Image processing

- Architecture

We experimented with a few network architectures and found that the architecture similar to LeNet works well with this problem. After just 1 epoch of training we were able to achieve 96% accuracy on the test set. We found that adding a BatchNorm layer as the first layer of the network greatly improves the network results.



- Data

We found that the dataset that we had did not correspond to the real data well. When trained on the raw, unprocessed symbols the model could fit the data well but would not generalize in a real-life scenario. We found that slightly eroded symbols correspond much strongly to a real-life data.

- Transformations

We found that gaussian adaptive thresholding works best for filtering out irrelevant features. Erosion after this step is a necessary step, because in this approach we assume that a symbol is a connected graph of black pixels, and so erosion creates this connection in places where it may have been filtered out by the low resolution of the camera or in a thresholding process(e.g. when the pen is almost empty, it tends to indent paper rather than leave ink on it)

- Rescaling

In this approach after localization step, we are left with a list of precise positions of the candidates for symbols. They are, however, of different sizes than 45x45, which is our CNN input shape.
We found that rescaling using standard rescaling approaches(e.g. linear or bicubic interpolations) causes problems when the height is much bigger than width of the symbol candidate.
We found that applying padding first to the smaller dimension to fit the other and then using a standard rescaling method yields better results.

- Noise reduction

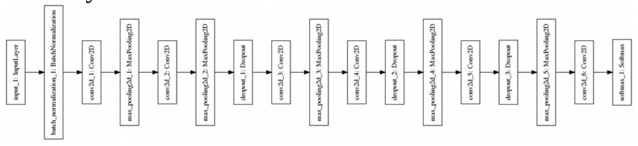
We found that thresholding introduces some noise, but simple filtering based on the number of black pixels in a connected components works pretty well and generalizes for different lighting conditions.

b) Sliding window

- Architecture

A naive implementation of sliding window was too slow to work in real-time. We decided on a convolutional implementation. The network is a series of convolutional, and max pooling layers, ocassionally followed by dropout layers.

The final layer is a pixelwise softmax layer, which predicts the class of the image as one of the symbols or background. Each pixel at the end of the network corresponds to a 32x32 patch of the original image. Applying the pixelwise softmax layer at the end of the network gave us a base for further processing. With this approach we were able to instantly achieve over 90% accuracy on a test set.



- Data

We found that eroding the symbols before pasting them on a background causes the detector to generalize better to real-life data.

In merging the symbol with the background we use the fact that the darker pixels have lower values across channels than brighter ones.

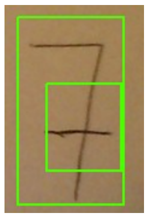
In our first try of merging the symbol with the background, we converted the symbol into a 3-channel bgr image and took minimum value of corresponding pixels across the two images. This resulted in an image where the symbol appear to have been pasted on the symbol.

However, the detector seemed to not generalize and failed to locate any symbols. We found that in a real-life scenario the pixels corresponding to the symbols have much higher values than the pure black ones that we receive in this approach. Also, the differences between the symbol intensities across channels seemed to differ. Hence in the new approach instead of taking minimum across channels we “darken” parts of the backgrounds corresponding to the symbol location by a predefined, experimentally set value.

- IoU

During inference, the predictions for windows may overlap and find the same object. In this case, we would want to predict the existence of one object only, rather than two. We use Intersection over Union metric to assess how much the windows overlap and remove the window corresponding to a prediction with lower confidence.

We found that this approach generates problems in a way that some lower confidence predictions were more appropriate(example).



In the above example, although ‘7’ is the correct prediction, the ‘+’ has higher confidence and so will be chosen instead.

6) Conclusion

We have found that custom image processing worked best with the problem. The structure of this problem is relatively well defined – we know that darker pixels correspond to the ink on paper and that they have much lower pixel intensities than their neighbors. In the image processing approach we are able to utilize this information and find the exact locations of the symbols on an image. This reduced our problem into a classification problem and so the result ended up much better.

We have learnt that when using the deep learning approaches it is essential to have a good quality dataset. We were able to fit the generated data achieving over 90% accuracy, but this did not generalize well in the real-life scenario. We found that in a real-life scenario pixel intensities of handwritten symbols differ across channels and so a more sophisticated method of pasting the symbols could yield better results. Also, we worked with patches of 45x45 paper images in 3 different lighting conditions. Gathering more background data could help the detector generalize better.

One more thing which could be done is the preprocessing of the symbol dataset in a way that adds padding at the borders. This way the classifier can learn to detect only those images in which the whole symbol is present.