Using CNN for letter recognition in images

Daniel Aaron Salwerowicz
Institutt for datateknologi og beregningsorienterte ingeniørfag
Arctic University of Norway, Narvik
Narvik, Norway
dsa014@post.uit.no

***Abstract*—In this report I describe my research related to developing CNN (Convolutional Neural Network) from scratch and then finding the best parameters to train it in recognizing letters in images. It shows how image is processed in between layers using my own implementation of them and then it shows what are the best parameters to use in such a network to develop best working model.**

***Index Terms*—CNN, kernel, stride, letter recognition, object recognition, CIFAR-10**

I. Introduction

This research project revolved around finding the best parameters for learning a neural network to recognize letters in images. Parameters that I looked on were the pooling size, kernel and stride for convolutional layer, and dropout rate in between layers.

Firstly I focused on better understanding how CNNs work by implementing parts of it myself. After getting working model I ran my training set through it to see how images are transformed after being passed through each layer. Then I worked on training CNN to tell A and K letters apart from the rest of letters. Lastly I used CIFAR-10 data set to better train it with a huge data set so that I could get better results.

II. Problem specification

Main problem that I focused on was choosing right kernel and stride size for convolution layers as well as dropout between layers.

To begin with I needed to get better understanding how CNNs work, I done that by implementing a few layers without using any standard libraries like Keras or Tensorflow. Each layer included a convolutional layer with various kernels, a rectifier that is a simple ReLU, and lastly max pooling. Convolution in a layer simply takes a matrix representing the image's colour values (from 0 to 1, inclusive) and applies a kernel to it, starting in the top left corner moving to the right and down with a given kernel. Rectifier activates neuron based on the value put out by it, in the case of simple ReLU it is: $\text{ReLU}(x) = \max(0, x)$. Pooling layer essentially downsizes the image by looking at a portion of a matrix, choosing the maximal or minimal value in that portion before moving along in the similar fashion to convolutional layer.

After that I used Keras and Tensorflow along with other standard libraries to train my CNN model in recognizing letter images.

Lastly I was tasked to used that knowledge and try to set up a simple CNN that would give satisfactory results in categorizing CIFAR-10 dataset.

III. Methods applied

When it comes to the implementation of convolution layer it was done simply by using NumPy's arrays to optimize operation time of my program. When an image is loaded and converted into a matrix it is simply iterated over and each part of it is multiplied piecewise with the kernel matrix and summed which is then put into the result matrix. this is done by utilizing NumPy functions sum and multiply that work much faster than conventional for loops would work. Code can be seen here: `result[x, y] = sum(multiply(map, kernel))` where map is a portion of feature map that gets multiplied with kernel.

Pooling works in similar fashion where I iterate over feature map and choose the biggest element in the portion of it. ReLU itself is a mere line long where I create a lambda method that returns x or 0 depending on which one is bigger, vectorize this lambda and used vectorized method on the feature map.

To learn my model to recognize letters I used Keras and Tensorflow and their Sequential model with 3 convolutional layers LeakyReLU and MaxPooling in between them before flattening and condensing the result. I also used 4 dropout layers in between to avoid overfitting. Same techniques were used to categorize CIFAR-10 dataset.

IV. Results

Results from my own implementation of CNN can be seen in fig. 1, as one can clearly see here the edges of letter are visibly enchanced while values for noise are brought down by a huge factor. This is shown by red-ish pixels around A before processing which show values between 0.9 to 1.0, which have been reduced to at most 0.05. Edges are emphasized since I am using an edge kernel in my convolutional layers, shown in eq. (1)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{1}$$

Results for most of the letters are quite satisfactory, however since the sample size I have gotten was of really bad quality (noise from jpeg compression), not all letters look as good. Though I have managed to reduce it by sharpening the images.

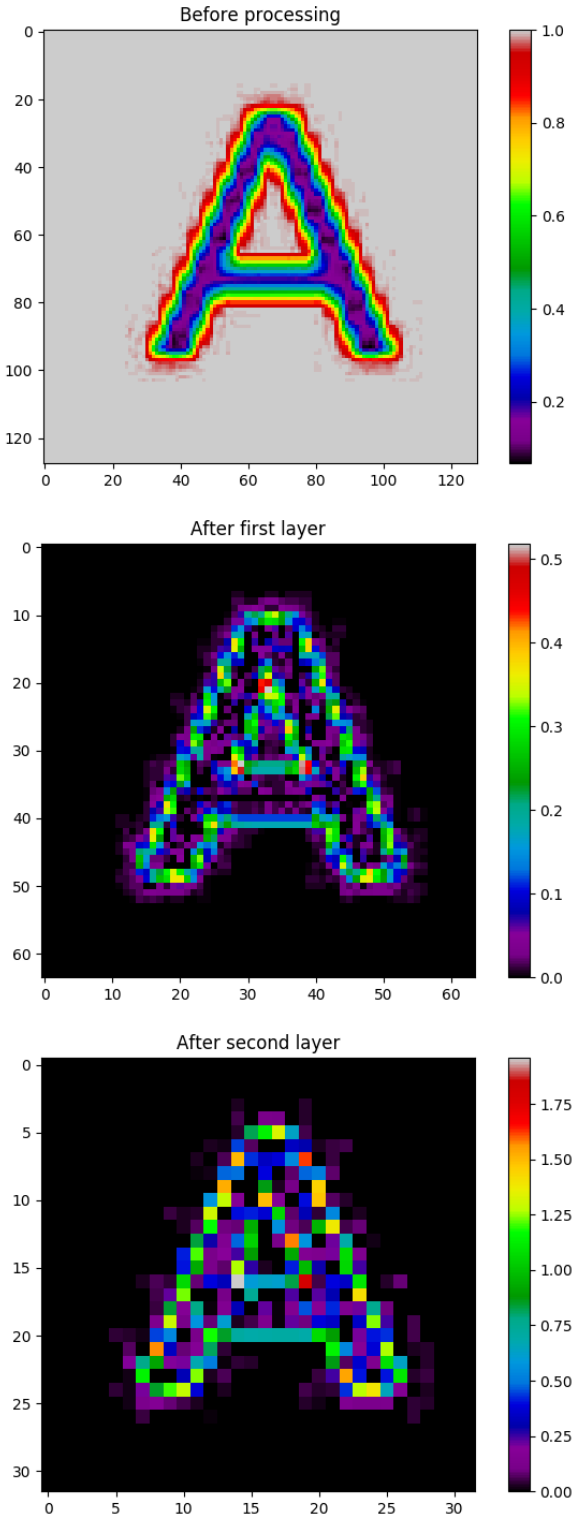

Fig. 1. Visualization of a matrix for letter A before and after it went through two layers of my CNN

V. Discussion

Acknowledgments

I would like to thank Christopher Kragebøl Hagerup, Kent Arne Larsen, Hans Victor Andersson Lindbäck, and Olav Kjartan Larseng for helping me underway. We brainstormed a lot of ideas on how to solve this problem, and I worked a lot together with Christopher on the first part of this project.

References