Assignment 4: Recognizing House Numbers with Deep Neural Network

Dalia Ibrahim[1] and Carlos Dasaed Salcedo[2]

Studnet ID: [1]201893217, [2]201892008

March 29, 2019

1 Introduction

For this assignment, we have implemented a program that takes as input a training data set and a testing dataset from the SVHN samples in .mat format, creates a deep convolutional neural network, and saves a trained model as Model.h5. We used SciKit Learn libraries, numpy, pandas, matplotlib, h5py, and the Keras functional API from Tensorflow to generate the trained a Deep Convolutional Neural Network(DCNN) model. To test and validate the model, several different types of layers were tested until an accuracy of 90% was achieved.

2 Data Pre-processing

The primary task at hand was to classify house numbers given in pictures, which makes the colours of the image almost irrelevant as a red five or a blue five would still be classified as a 5. Based on this reasoning, we decided to convert all the training and testing data sets to grey scale. This drastically reduces the initial input of the DCNN as we deal with with a single value per pixel, rather than the three values obtained from the RGB format. Since the colours of each pixel will always range from 0 to 255, we decided to apply a simple normalization step by dividing the value of each pixel by 255. This way, all pixels will now have a value between 0 and 1.

3 MODEL PARAMETERS

The model parameters were all selected and tested by hand, as there is not an efficient method for selecting the number of layers, type of layers, activation functions, amongst other features, other than by trial and error. We started with the default values of the examples found in the Tensorflow documentation, and then proceeded to modify various parameters. Since the program was being executed locally in our computers, we did not have the computational power of servers generally used for this kind of tests as that of IBMs Watson, or Googles Alpha Go projects.

- Activation Function Two activation functions were tested: ReLU and Sigmoid. ReLU is generally considered to a good default function for any ML model, and hence, was the first function we decided to test. For testing purposes, we selected Sigmoid, as we wanted to compare the performance different activation functions. Ultimately, the ReLU function proved to be faster and yielded more accurate results than the Sigmoid function.
- Optimizer The first optimizer tested was ADAM. Since we are more familiar with stochastic gradient descent, we also tested SGD, and obtained WORSE results in both, accuracy of the model and training time. Our final selection was ADAM.
- Number of epochs and batch size Although, in the literature and in tutorials, values ranging from 10 to extremely large numbers may be selected, we decided to use an epoch value of 10, as our computational power was limited, and we wanted to avoid overfitting. Likewise, our batch parameter was set to 100 to keep our running as low as possible without compromising too much on accuracy.
- Metrics For this assignment, we only focused on accuracy, rather than the AUPRC which was used in previous assignments, since our goal was on obtaining the most accurate model possible from the given training samples.
- Regularization In an effort to reduce the generalization error, we have used the dropout function offered in the Keras library.

4 RESULTS

We have created a table containing the parameters modified and the accuracy obtain from the various tests that were done with the code. We decided to use convolutional layers in the initial layers as convolutional neural networks have been proven to be more computationally efficient in terms of number of operations than other MLPs. The pooling layers were used to help reduce the variance of the model and obtain a more generalizable model. The last layer of the model consist of a dense layer with softmax as the activation function that outputs what our model thinks is the most probable classification of the input image. Our final model was selected based on our trial and error tests. Figure 1 displays a sample classification of 15 images classified by the model, while table in the last page contains the results and configurations used in various tests. A Jupyter notebook with an explanation of our code can be found in the following link: https://github.com/daliaibrahim/RecognizingHouseNumbers/blob/master/A4.ipynb

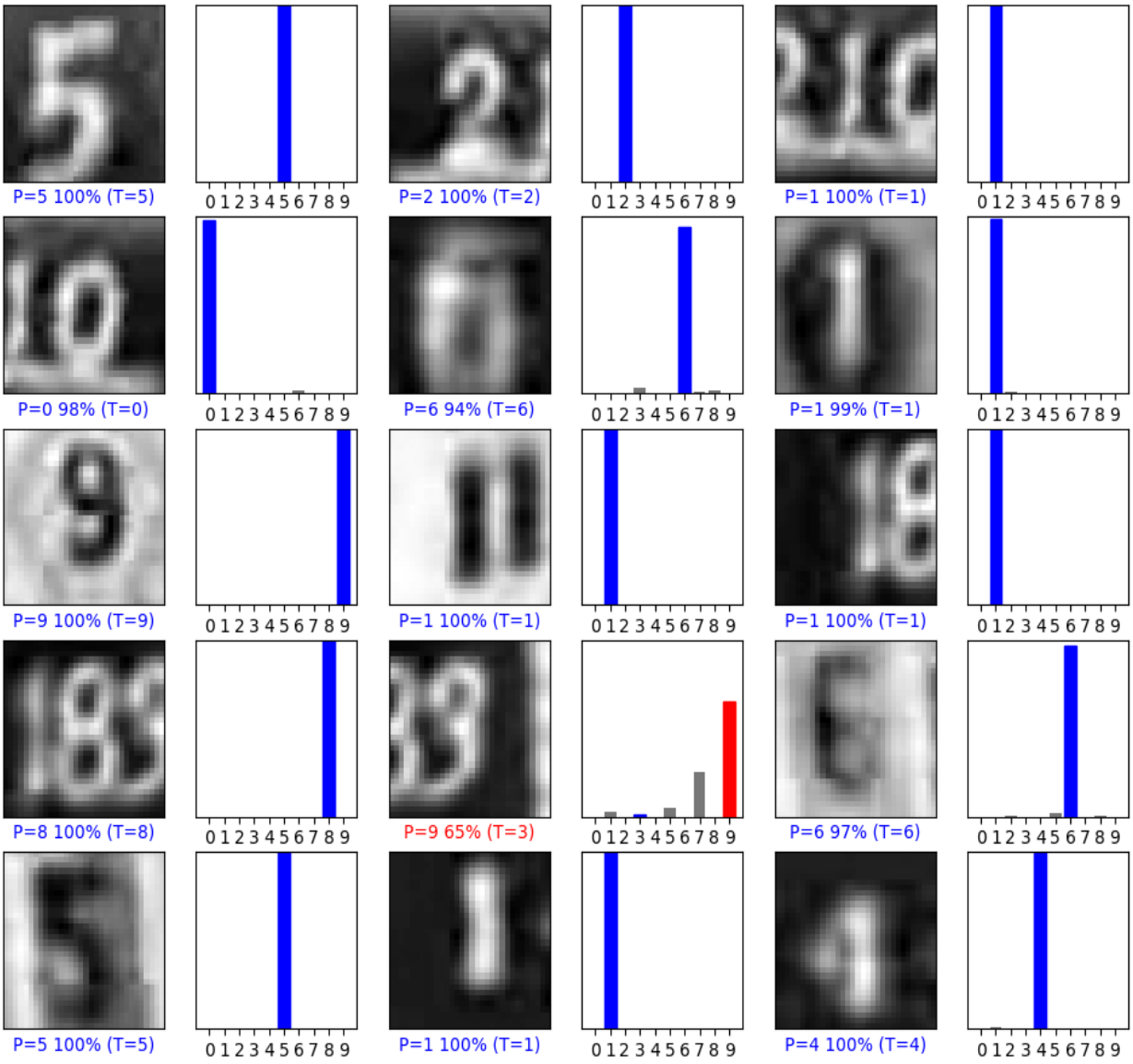

Figure 1: Sample output

	Architecture 1	**Architecture 2**	**Architecture 3**	**Architecture 4**	**Architecture 5**
	CNN 32 hidden units Activation function = RELU Kernel_ size (3 X 3)	CNN 32 hidden units Activation function = RELU	CNN 32 hidden units Activation function = RELU AllowPadding	CNN 32 hidden units Activation function = RELU AllowPadding	CNN 32 hiden units Activation function = SigmoidAllowPadding
	CNN 64 hidden units Activation function = RELU	CNN 64 hidden units Activation function = RELU	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = Sigmoid AllowPadding
	Max pooling size (2 x 2)	CNN 64 hiden units Activation function = RELU	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = SigmoidAllowPadding
	Dropout (0.25)	CNN 64 hidden units Activation function = RELU	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = RELU AllowPadding	CNN 64 hidden units Activation function = SigmoidAllowPadding
	Flatten	Max pooling size (2 x 2)	Max pooling size (2 x 2)	Max pooling size (2 x 2)	Max pooling size (2 x 2)
	Dense 128 Hidden units Activation = RELU	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
	Dropout (0.25)	Flatten	Flatten	Flatten	Flatten
	Dense 10 Hidden units Activation = Softmax	Dense 128 Hidden units Activation = RELU	Dense 128 Hidden units Activation = RELU	Dense 128 Hidden units Activation = RELU	Dense 128 Hidden units Activation = Sigmoid
		Dense10 Hidden units Activation = Softmax	Dense 10 Hidden units Activation = Softmax	Dense 10 Hidden units Activation = Softmax	Dense 10 Hidden units Activation = Softmax
optimizer	adam	adam	adam	**SGD**	adam
loss Function	categorical_crossentrop	categorical_crossentrop	categorical_crossentrop	categorical_crossentrop	categorical_crossentrop
Accuracy	87.00%	87.00%	**89.70%**	88.00%	60.00%

Table 1 Tested Architectures