Memorial University of Newfoundland

Computer Science 6915 Assignment 4

Deep Neural Network Model for Image Classification

Isaac Adejuwon & Sourav Sarker

Submission Date: March 30, 2019

Introduction: The purpose of this assignment is to use TensorFlow to create a Deep Neural Network model to classify house numbers. Image data from Street View House Number (SVHN) Dataset (Format 2) available via Stanford http://ufldl.stanford.edu/housenumbers/ is used for the assignment. The obtained train data is used to train the networks and test data is used to evaluate network performance respectively.

Network Architecture and Justification: Convolutional Neural Network was used because of its high success rate in processing data that has a grid structured topology such as 2D images. In addition, its computational efficiency compared to fully connected networks makes it suitable for the House Number dataset used for this assignment. To build the model, network layers were first setup in order to get a good representation from the dataset. For each layer, 64 number of units was used. The network uses 73,257 images for training and 26,032 images to evaluate how accurately the network learned to classify the images. The images used for training and testing accuracy of the network are 32*32 in size, and accounted for the 3 colours present in the images.

How the Neural Network is Built: For this assignment, we focused to score better accuracy. So, to build the Neural Network, the following procedures were used: data was imported, explored, preprocessed, model was built (layer setup and model compiling), model was trained, model was evaluated with the testing data and predictions were made respectively.

Data Pre-processing: After importing the data, the training and test data was split into X and y labels respectively. Before feeding data into the network, the images was first normalized by scaling using an assumption that images have a minimum value of 0 and maximum value of 255 pixel data. To keep the all pixels value between 0 to 1, we divided each pixel by 225.

```
X_train, X_test = X_train / 255.0,
X_test / 255.0
```

During exploration of the data, a transpose is done to both the training and test data in order to adjust shape of the data to a format that is suitable for the network.

Model Parameters: To maintain a standard neural network architecture, we started with default values of the parameters from the Keras tutorials. As this program takes much longer time to execute and we did not have computational resources for this program to test all the features, we tried to run our program by using Google cloud service (google-colab). We have fine tuned following parameters:

- **Optimizer:** First we applied “adam”, got accuracy 78% then applied “adadelta”. By that our out of sample accuracy was inclined but the out of sample loss was also increased. To get rid of high out of sample loss, we used Stochastic Gradient Descent (SGD) as an optimizer with some fine tuning in learning rate, momentum which directs the optimizer to the relevant direction, decay and Nesterov momentum.
- **Epochs with batch size:** We might get higher accuracy with higher epochs depending on the size of training dataset but it can lead our model to be over-trained. So, we choose 10 epochs. As we have over 70 thousand training samples, we went with batch size of 128.

- **Regularization:** For data regularization, we deployed dropout for making our network less sensitive to the specific weight of neurons. The dropout size is 0.25.
- **Activation Function:** Two activation functions named “relu” and “sigmoid” were tested. However, the “softmax” was applied for classification at the end layer. From the performance’s perspective we got better result with “relu”. Therefore, we moved with “relu”.
- **Metrics:** In this assignment, our only objective was to find an optimum model with best accuracy.

Building the model: A sequential model is used for this assignment. Selecting sufficient number of layers for compiling a model is tricky. As a result, we implemented two architectures: one with two convolution layers and another with 4 convolution layers. Because of getting high out of sample loss (39%) we skipped the model with two convolution layers. We built our model as follows:

- For the first two convolution layers, we used 32 output filters and for the last two layers, we took 64 output filters with 3-by-3 kernel size to define the width and height of the 2D convolution window. As we are given RGB images with 32-by-32 pixel, so the input shape we made based on that in the first convolution layer which is 32,32,3.
- Our activation function is “relu” for setting zero threshold.
- To prevent over-fitting the model, after two convolution layers we deployed “Dropout” with a value of 0.25.
- In every hidden layer, batch normalization has been performed to normalize the input layer by scaling the activation function, This technique also helped to speed up learning.
- In the max pooling layer, 2, 2 pool size is used to halves the input in both spatial dimension which means down-sampling by factor 2.
- After four groups of layers, we again deployed “Dropout” with the same value and then flatten performed the input role. Next dense layer came into play to densely connect layer with a value of output space (200).
- In output layer, it follows “relu” activation, “BatchNormalization”, “Dropout” and “Dense” layer with a value of output space (10). The last fully connected layer has 1 output and “softmax” activation functions.

For this assignment, we are given a classification problem, so to compile the model, loss is compiled using categorical cross entropy and an optimizer is compiled using SGD. Accuracy is the metric measured for by the model.

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
optimizer=sgd,metrics=['accuracy'])
```

Here is the summary of our model:

Layer (type)	Output Shape	Param #
conv2d_77 (Conv2D)	(None, 32, 32, 32)	896
activation_109 (Activation)	(None, 32, 32, 32)	0
batch_normalization_111 (Bat	(None, 32, 32, 32)	128
conv2d_78 (Conv2D)	(None, 30, 30, 32)	9248
activation_110 (Activation)	(None, 30, 30, 32)	0
batch_normalization_112 (Bat	(None, 30, 30, 32)	128
max_pooling2d_45 (MaxPooling	(None, 15, 15, 32)	0
dropout_55 (Dropout)	(None, 15, 15, 32)	0
conv2d_79 (Conv2D)	(None, 15, 15, 64)	18496
activation_111 (Activation)	(None, 15, 15, 64)	0
batch_normalization_113 (Bat	(None, 15, 15, 64)	256
conv2d_80 (Conv2D)	(None, 13, 13, 64)	36928
activation_112 (Activation)	(None, 13, 13, 64)	0
batch_normalization_114 (Bat	(None, 13, 13, 64)	256
max_pooling2d_46 (MaxPooling	(None, 6, 6, 64)	0
dropout_56 (Dropout)	(None, 6, 6, 64)	0
flatten_23 (Flatten)	(None, 2304)	0
batch_normalization_115 (Bat	(None, 2304)	9216
dense_44 (Dense)	(None, 512)	1180160
activation_113 (Activation)	(None, 512)	0
batch_normalization_116 (Bat	(None, 512)	2048
dropout_57 (Dropout)	(None, 512)	0
dense_45 (Dense)	(None, 10)	5130
activation_114 (Activation)	(None, 10)	0

```
Total params: 1,262,890
Trainable params: 1,256,874
Non-trainable params: 6,016
```

Figure 1: Model Summary

Train the model: To train the model, a batch size of 128 is passed, validation slit of 10% is used and an epoch of 10 is used to train.

After running the model, we got a 93% validation accuracy and 24% validation loss after 10 epochs on the training dataset.

```
[ ] model.fit(X_train, y_train, validation_data= (X_val, y_val), batch_size = 128, epochs= 10, verbose= 1, callbacks=[tensorboard])

Train on 58605 samples, validate on 14652 samples
Epoch 1/10
58605/58605 [==============================] - 24s 408us/step - loss: 0.9391 - acc: 0.7079 - val_loss: 0.5160 - val_acc: 0.8433
Epoch 2/10
58605/58605 [==============================] - 19s 322us/step - loss: 0.4555 - acc: 0.8607 - val_loss: 0.4579 - val_acc: 0.8633
Epoch 3/10
58605/58605 [==============================] - 19s 324us/step - loss: 0.3776 - acc: 0.8847 - val_loss: 0.3180 - val_acc: 0.9069
Epoch 4/10
58605/58605 [==============================] - 19s 324us/step - loss: 0.3341 - acc: 0.8970 - val_loss: 0.2883 - val_acc: 0.9157
Epoch 5/10
58605/58605 [==============================] - 19s 323us/step - loss: 0.3015 - acc: 0.9086 - val_loss: 0.3656 - val_acc: 0.8898
Epoch 6/10
58605/58605 [==============================] - 19s 323us/step - loss: 0.2759 - acc: 0.9157 - val_loss: 0.3348 - val_acc: 0.8989
Epoch 7/10
58605/58605 [==============================] - 19s 323us/step - loss: 0.2554 - acc: 0.9218 - val_loss: 0.2540 - val_acc: 0.9297
Epoch 8/10
58605/58605 [==============================] - 19s 323us/step - loss: 0.2359 - acc: 0.9284 - val_loss: 0.2553 - val_acc: 0.9281
Epoch 9/10
58605/58605 [==============================] - 19s 322us/step - loss: 0.2237 - acc: 0.9324 - val_loss: 0.2600 - val_acc: 0.9268
Epoch 10/10
58605/58605 [==============================] - 19s 321us/step - loss: 0.2052 - acc: 0.9368 - val_loss: 0.2471 - val_acc: 0.9303
<keras.callbacks.History at 0x7f6c3433e438>

After training our model, we have got

- 93% out of sample accuracy and 24% out of sample loss &
- 94% in-sample accuracy and 21% in-sample loss

after 10 epochs.
```

Figure 1: Model Training

We also saved some log files which is generated by TensorBoard to visualize the performance of the model per epoch.

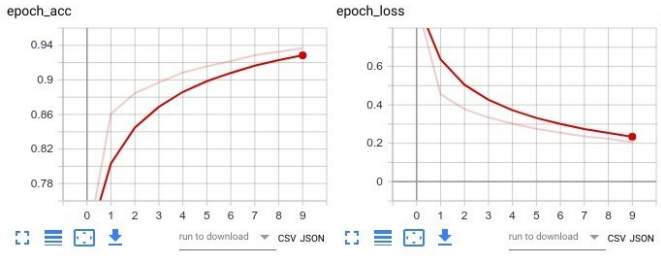

Figure 2: In-sample accuracy and loss

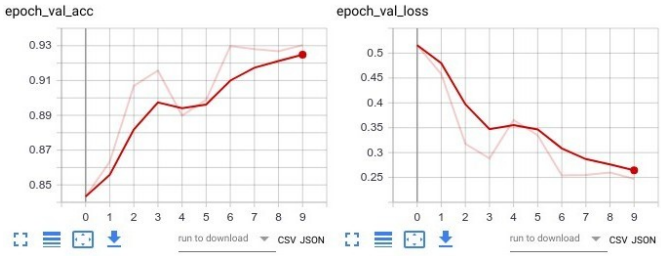

Figure 3: Out-of-sample loss and accuracy

Model Evaluation and Prediction: We evaluated the model, with the given testing dataset and we scored 92% validation accuracy and a 27% validation loss on the testing dataset. An image with number 5 as shown below was tested.

Several images with their predicted accuracies is shown in figure 4 below: Correct prediction labels are show in blue colours and incorrect prediction labels are shown in red. Accuracies for each images is shown using a scale ranging from 0-100%.

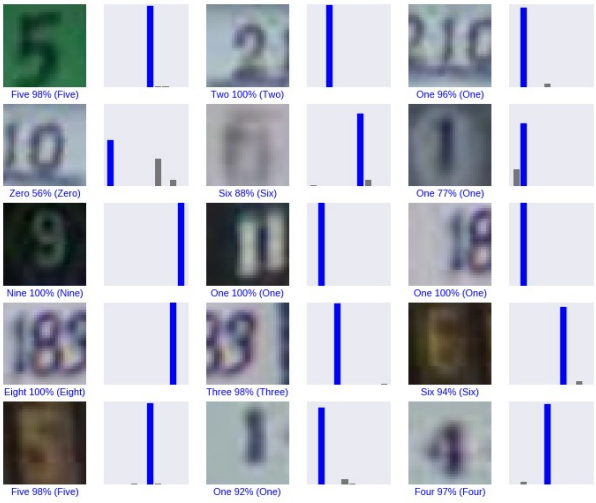

Figure 4: Several Prediction of Images

Conclusion: A 93% out of sample accuracy and 24% out of sample loss after 10 epochs on the training dataset was adequate to make fairly accurate predictions on any test data used. Because a convolutional neural network was used for the assignment, there were cases where the model could not accurately recognize all numbers on images with more that 1 digit. To improve predictions for such cases, a separate model would need to be built to account for such specific cases. Overall our model performed as expected.

Code Implementation can be accessed vial GitHub link below:

https://github.com/souravskr/ducspond_ai/blob/SVHN_CNN/SVHN_CNN.ipynb