CS-512: Computer Vision- FALL 2018

ASSIGMENT 4

Diego Martin Crespo

A20432558

1. PROBLEM STATEMENT

In this assignment a basic Convolutional Neural Network will be implemented for classification. Using Keras or TensorFlow a CNN will be builded and images from the MNIST dataset will be classified.

Deliverable 1: Custom CNN

1. Train your CNN using the following configuration:

a. Use the MNIST train set: 55,000 samples
b. 2 Convolutional and Pooling layers
 I. Layer 1: 32 filters of kernel size 5x5
 II. Layer 2: 64 filters of kernel size 5x5
 III. Pooling in both layers should downsample by a factor of 2
c. Dropout rate of 40%
d. Compute cross entropy loss
e. Use gradient optimization and learning rate of 0.001
f. Train for 5 epochs

2. Report the training loss and accuracy

a. As your model trains log the loss and accuracy for each step/iteration and plot the curve.
b. Also report the loss and accuracy value of the final training step

3. Report the evaluation loss, accuracy, precision and recall using MNIST test set (10,000 samples)

a. For each epoch trained, compute the above-listed metrics and plot the curve.
b. Also report the loss, accuracy, recall, and precision of the last epoch.

Deliverable 2: Parameter Tuning

Evaluate different variations of the basic network as described below and measure performance. In your report, discuss your variations, compare results you obtain and attempt to draw conclusions. You may evaluate the following aspects:

- Changing the network architecture (number of layers or organization)
- Changing the receptive field and stride parameters
- Changing optimizer and loss function (E.g Adam)
- Changing various parameters (e.g dropout, learning rate, number of filters, number of epochs)
- Adding batch and layer normalization
- Using different weight initializers (e.g Xavier, He)
- Using features from a pretrained model (e.g GVGG16)

Deliverable 3: Application

Write a program to use your pretrained custom CNN (Deliverable 1). The program should do the following:

1. Accept as input an image of a handwritten digit. Assume each image contains one digit
2. Using OpenCV do some basic image preprocessing to prepare the image for your CNN
 a. Resize the image to fit your model's image size requirement
 b. Transform grayscale image to a binary image. Consider using the GaussianBlur() and adaptiveThreshold(), or any other type of binary thresholding that performs well.
 c. Display the original image and binary image in two separate windows. 3.
3. Using your CNN classify the binary image.
4. Your program should continuously request the path to an image file, process the image, and output to the console that class (even/odd) of the image.
5. Program should terminate when 'q' or ESC key is entered.

2. PROPOSED SOLUTION

Deliverable 1:

The proposed solution for deliverable 1 it is based on Keras implementation, instead of tensorflow for commodity. However, tensorboard and some TensorFlow functions have been used. For this first part the model is trained using the configuration detailed in the problem statement. It has been executed using Colab platform from google.

Deliverable 2:

The configuration of the model 1 has been modified variating the following parameters:

1. Variation of the batch size
2. Variation of the number of epochs
3. Using a different optimizer function
4. Removing second layer of convolution and pooling
5. Variate the number of filters in convolution in both layers 1 and 2
6. Variation of drop rate

The different results will be discussed.

Deliverable 3:

The model from Deliverable 1 is saved into .h5 format. A different python script is created named cnn_test.py and the saved model is loaded in it. Then this model will be used to predict the input images. In order to do this OpenCV will be used for image processing as in previous assignments and the images will be prepared to fit the model properly.

3. IMPLEMENTATION DETAILS

Deliverable 1:

For the first deliverable which mainly involves the creation of the CNN model. The following steps and discussions have been faced:

First it has been decided to use Keras implementation instead of TensorFlow because the code seems to be more understandable and easier to implement. An example code has been used as referenced and modified as the problem statement.

Second Collab platform form google has been used in order to execute and generate the model. I did try to run it on my personal computer. However, it did take too much long to get the results.

Therefore, using Collab with GPU configuration, the model took about 30 seconds to generate (without using tensorboard) and about 1 min and 30 seconds (using tensorboard).

The graphs have been plotted using Tensorboard environment, which I had some problems with its configuration at first and also whenever I run the program. I manage to solved by resetting all the runtimes.

Finally, when implementing the precision and recall parameters I found two possible solutions from the Keras and TensorFlow communities. One of them makes a wrap function to use TensorFlow implementation of precision and recall functions. And the other is implemented by definition with Keras. The first one does not seem to perform right as the results were always the same in each iteration (precision= 0.5 and recall=0.5). Therefore, the implementation by definition with Keras has been used.

Deliverable 2:

The deliverable 2 was pretty straight forward as we got the initial model from the previous deliberate. We only need to variate the different parameters and see how the accuracy, loss, precision and recall rates variate. The results will be discussed in point 4.

Deliverable 3:

In deliverable 3 many problems have been faced. These are the followed steps:

First the model generated at Collab was saved into a .h5 file and downloaded. Then it was loaded in our cnn_test.py program.

On the other hand, in order to implement the required program, OpenCV library has been used to process images and prepare them for being usable for the CNN model.

In this part I did loss much time by trying to configure the environments of Conda. I got many errors of compiling and I was not sure what was the reason for that. I find out that python 3.5.5 was not compatible with OpenCV3 for any reason. I did manage to generate a new Conda environment with python 2.7 (instead of 3.5) and the program did finally work.

Also, I had some problems with the image dimensions that the model required. Once I figurate it was in the form of (1,28,28,1) I solved it easily.

Finally, to get the program running asking all the time the path for the image and when “Esc” or “q” keys are pressed the program should terminate. I have several problems because the program kept in an infinite loop waiting to the input path, therefor the keys were not listened. To solve this in order to quit the program you should press the “q” or “Esc” (“\x1b”) keys (write at the terminal window) and then press enter.

4. RESULTS AND DISCUSSION

The problem statement is solved using our proposed solution and implementation details. These are the results:

Deliverable 1: Custom CNN

1. Train your CNN using the following configuration:

a. Use the MNIST train set: 55,000 samples

b. 2 Convolutional and Pooling layers

   I. Layer 1: 32 filters of kernel size 5x5

   II. Layer 2: 64 filters of kernel size 5x5

   III. Pooling in both layers should downsample by a factor of 2

c. Dropout rate of 40%

d. Compute cross entropy loss

e. Use gradient optimization and learning rate of 0.001

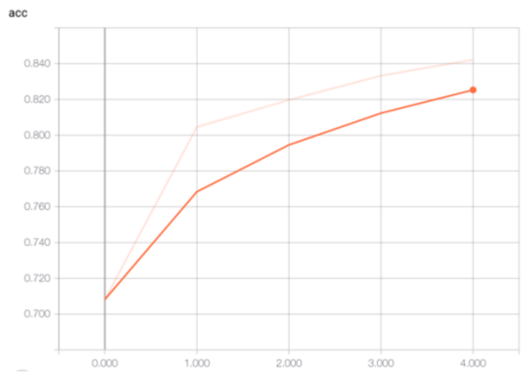
f. Train for 5 epochs

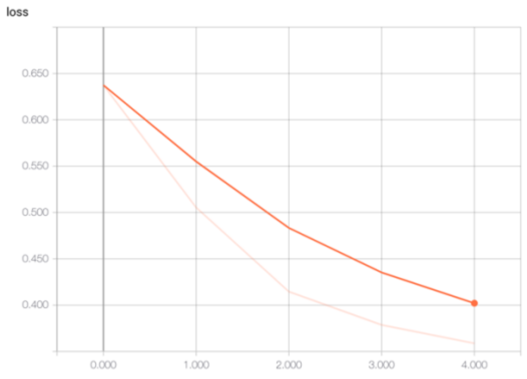
The program is configured so it meets the previous requirements.

2. Report the training loss and accuracy

a. As your model trains log the loss and accuracy for each step/iteration and plot the curve.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [=====] - 8s 138us/step - loss: 0.6462 - acc: 0.6659 - val_loss: 0.5745 - val_acc: 0.8058
Epoch 2/5
60000/60000 [=====] - 8s 134us/step - loss: 0.5147 - acc: 0.7974 - val_loss: 0.4315 - val_acc: 0.8373
Epoch 3/5
60000/60000 [=====] - 8s 134us/step - loss: 0.4130 - acc: 0.8280 - val_loss: 0.3589 - val_acc: 0.8529
Epoch 4/5
60000/60000 [=====] - 8s 135us/step - loss: 0.3647 - acc: 0.8455 - val_loss: 0.3231 - val_acc: 0.8671
Epoch 5/5
60000/60000 [=====] - 8s 135us/step - loss: 0.3370 - acc: 0.8598 - val_loss: 0.2983 - val_acc: 0.8783
```





- ● Train
- ● Test

b. Also report the loss and accuracy value of the final training step

- Loss: 0.2983
- Accuracy: 0.8783

3. Report the evaluation loss, accuracy, precision and recall using MNIST test set (10,000 samples)

a. For each epoch trained, compute the above-listed metrics and plot the curve.

Train on 60000 samples, validate on 10000 samples

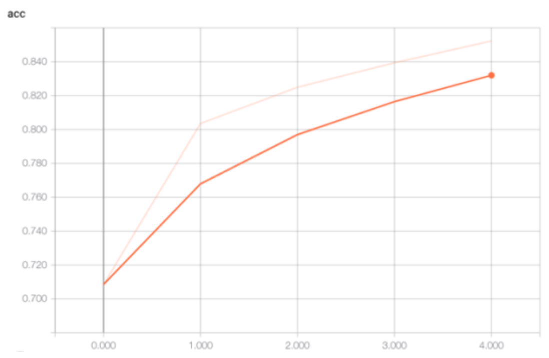
- Using Keras implementation:

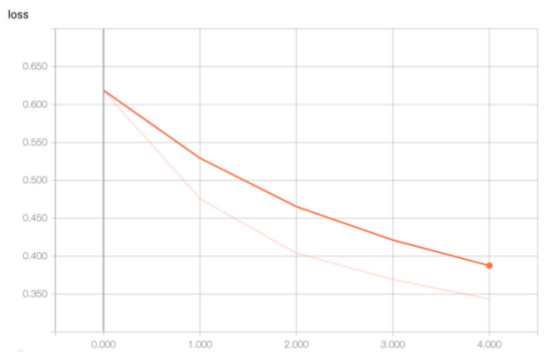
```
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 9s 151us/step - loss: 0.6240 - acc: 0.7248 - precision: 0.4995 - recall: 0.9989
- val_loss: 0.5435 - val_acc: 0.8262 - val_precision: 0.5000 - val_recall: 1.0000
Epoch 2/5
60000/60000 [==============================] - 9s 147us/step - loss: 0.4827 - acc: 0.8132 - precision: 0.5000 - recall: 1.0000
- val_loss: 0.4025 - val_acc: 0.8411 - val_precision: 0.5000 - val_recall: 1.0000
Epoch 3/5
60000/60000 [==============================] - 9s 148us/step - loss: 0.3943 - acc: 0.8316 - precision: 0.5000 - recall: 1.0000
- val_loss: 0.3444 - val_acc: 0.8556 - val_precision: 0.5000 - val_recall: 1.0000
Epoch 4/5
60000/60000 [==============================] - 9s 148us/step - loss: 0.3575 - acc: 0.8467 - precision: 0.5000 - recall: 1.0000
- val_loss: 0.3151 - val_acc: 0.8688 - val_precision: 0.5000 - val_recall: 1.0000
Epoch 5/5
60000/60000 [==============================] - 9s 148us/step - loss: 0.3306 - acc: 0.8596 - precision: 0.5000 - recall: 1.0000
- val_loss: 0.2917 - val_acc: 0.8816 - val_precision: 0.5000 - val_recall: 1.0000
```

- Using TensorFlow wrap function:

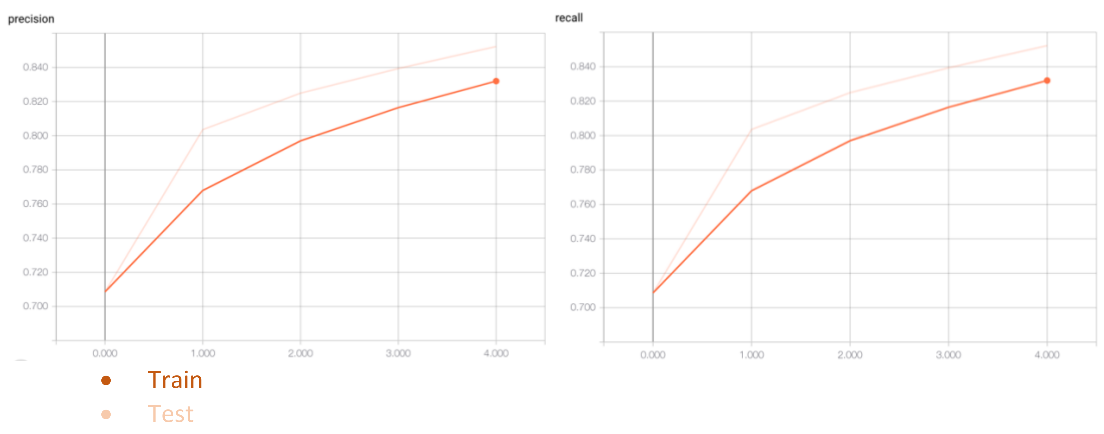
```
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 9s 148us/step - loss: 0.6330 - acc: 0.6950 - precision: 0.6950 - recall: 0.6950
- val_loss: 0.5589 - val_acc: 0.8133 - val_precision: 0.8133 - val_recall: 0.8133
Epoch 2/5
60000/60000 [==============================] - 9s 144us/step - loss: 0.5035 - acc: 0.8001 - precision: 0.8001 - recall: 0.8001
- val_loss: 0.4231 - val_acc: 0.8404 - val_precision: 0.8404 - val_recall: 0.8404
Epoch 3/5
60000/60000 [==============================] - 9s 144us/step - loss: 0.4061 - acc: 0.8314 - precision: 0.8314 - recall: 0.8314
- val_loss: 0.3507 - val_acc: 0.8608 - val_precision: 0.8608 - val_recall: 0.8608
Epoch 4/5
60000/60000 [==============================] - 9s 144us/step - loss: 0.3572 - acc: 0.8490 - precision: 0.8490 - recall: 0.8490
- val_loss: 0.3127 - val_acc: 0.8751 - val_precision: 0.8751 - val_recall: 0.8751
Epoch 5/5
60000/60000 [==============================] - 9s 143us/step - loss: 0.3255 - acc: 0.8627 - precision: 0.8627 - recall: 0.8627
- val_loss: 0.2856 - val_acc: 0.8885 - val_precision: 0.8885 - val_recall: 0.8885
```

Graphs using Keras implementation:





- ● Train
- ● Test



b. Also report the loss, accuracy, recall, and precision of the last epoch.
Seeing both results of keras function and tensorflow functions we decide to use Keras implementation. Therefore, the results are:

- Loss: 0.3127
- Accuracy: 0.8885
- Recall: 0.8885
- Precision: 0.8885

There is to notice that the accuracy, precision and recall are the same. These could be because the false positives and the false negatives are equal. Or because of the implementation with Keras libraries.

Deliverable 2:

The configuration of the model 1 has been modified variating the following parameters:

Initial accuracy 0.885 and loss 0.3127.

1. Variation of the batch size:

   The initial configuration was done using a batch size of 64. If we increase that value to 128 the values of accuracy decrease from 0.8885 to 0.8442 and the loss increases from 0.3127 to 0.3821. Therefore, the smaller the batch size the more precision and less loss the model will have.

2. Variation of the number of epochs:

   The initial configuration was 5 epochs. If we increased to 10 epochs the model logically will behave more accurate as in each epoch it improves from the previous one. The accuracy increases to 0.9244 and the loss decreases to 0.2076.

3. Using a different optimizer function:

   Using Adam() optimizer with the same learning rate intead of the sdg(). The accuracy of the model increases quite a lot up to 0.9939. and the loss rate reduces to 0.0170.

4. Removing second layer of convolution and pooling:

   If the second layer of convolution is removed, the accuracy decreases slightly down to 0.8617 and the loss rate increases a bit up to 0.3235.

5. Variate the number of filters in convolution in both layers 1 and 2:

   If the number of filters is reduced to half in both filters, the accuracy decreases slightly down to 0.8605 and the loss rate increases a bit up to 0.3147.

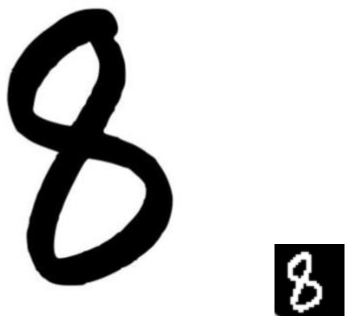
6. <u>Variation of drop rate:</u>

   If we increase de drop rate to 50%. The accuracy decreases down to 0.8530 and the loss rate increases a bit up to 0.3450.

Deliverable 3:

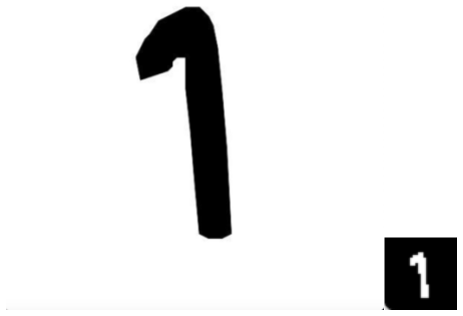
The results from Deliverable 3 are the following:

- When it is passed an image with an even number the program classifies it and prints "Classified as even or class 0".



Original at the left and binary at the right

- When it is passed an image with an odd number the program classifies it and prints "Classified as even or class 1".



Original at the left and binary at the right

Notice that the ratio of white and black is inverted from the original to the binary. This has been done because the MNIST dataset have the trained images from the form of number white and background black. Therefore, it should classify the image more correctly.

5. REFERENCES

https://keras.io

https://faroit.github.io/keras-docs/1.2.2/

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

https://www.dlology.com/blog/quick-guide-to-run-tensorboard-in-google-colab/

https://github.com/flo22/classify-even-odd/blob/master/classify.py

https://github.com/keras-team/keras/issues/2435