cs512 Assignment 3

Chandni Patel
Department of Computer Science
Illinois Institute of Technology

November 9, 2020

Abstract

This report contains description of the convolutional neural network problem, the algorithms used for solving the problem, and analysis of the results obtained for the programming part of Assignment 3.

1 Problem Statement

The goal is to write a program to implement a basic convolutional neural network for classifying images of numbers in the MNIST dataset as either even or odd using GPU framework with following requirements:

1. **Construct CNN**: Load MNIST dataset and split it into training, validation, and testing subsets with 55000, 10000, and 5000 examples, respectively. Convert digit labels to even or odd using 0 and 1. Construct and train CNN using 2 convolution layers with pooling, a dropout layer, two fully connected layers, appropriate loss function, optimizer algorithm and evaluation matrix. Plot training and validation loss and accuracy as a function of epochs.
2. **Hyper-parameter Tuning**: Evaluate different variations of basic network by changing network architecture, receptive field, stride, optimizer, loss function, parameters like dropout, learning rate, number of epochs, weight initializers, batch normalization, and layer normalization. Also, measure performance with each change.
3. **Inference**: Take images with handwritten digits as input. Perform image preprocessing like resizing, converting to grayscale and then to binary image using threshold. Display original images and binary images side by side. Classify input images into odd/even labels using the final CNN obtained after hyperparameter tuning.

2 Proposed Solution

Solution for this programming assignment is created using Python, TensorFlow, Keras, and OpenCV. Python notebook for coding was created on Google Colab.

3 Implementation Details

Construct and train CNN

- MNIST data is split it into training, validation, and testing subsets with 55000, 10000, and 5000 examples, respectively. Then the labels are replaced to 0 for even digits and 1 for odd digits using the implemented GetOddEvenLabels() function.
- To construct the initial convolutional neural network, following layers are added:
    i. A convolution layer with 32 filters of size 3x3 and ReLU activation
    ii. A max pooling layer with pool size (2, 2)
    iii. A convolution layer with 64 filters of size 3x3 and ReLU activation
    iv. A max pooling layer with pool size (2, 2)
    v. A convolution layer with 64 filters of size 3x3 and ReLU activation
    vi. A dropout layer with rate = 0.3
    vii. A flatten layer
    viii. A fully connected layer with 64 units and ReLU activation
    ix. A fully connected layer with 2 units and sigmoid activation as we have 2 labels to classify.
- Selected SGD as optimizer, Binary Cross Entropy as loss function and Accuracy evaluation matrix. Choosing 5 epochs and 64 batch size to fit the model.
- Then plotted training and validation loss as a function of epochs and training and validation accuracy as a function of epochs using the implemented GetLossAccuracyPlot() function.

Hyper-parameter Tuning: Implemented CNN() function to modify and recreate models with different parameters. Hyper-parameter tuning is further discussed in detail in the next section.

Inference: Created images for handwritten digits from 0 to 9 of original size 280x280 pixels. Inference is further discussed in detail in the next section.

Instructions to run the program

I. When running the programs, the Jupyter notebook and the input images for inference will have to be in the same folder location.

II. I have the code to upload files on Google Colab which I used initially so you can comment it if not required.

4 Results and Discussion

Here are the convolution neural networks, model evaluations and loss & accuracy plots obtained:

Construct and train CNN

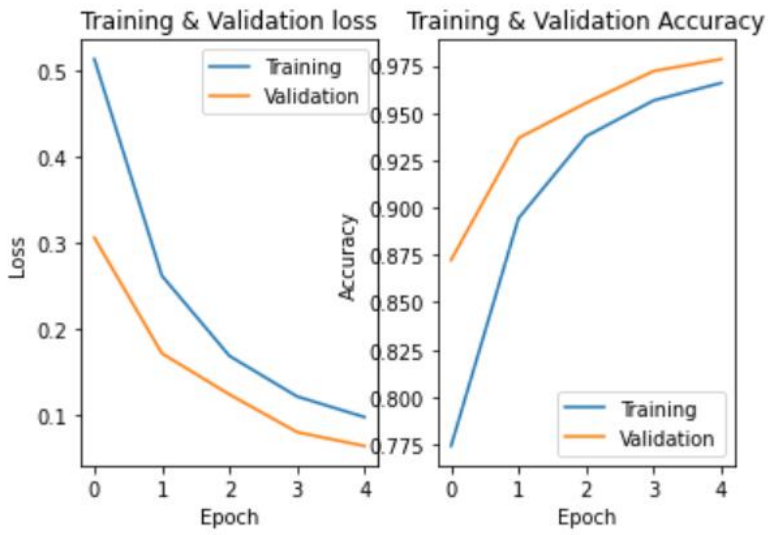
All details are discussed for initial CNN in previous section. Here is the model created:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
dropout (Dropout)            (None, 3, 3, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 92,802
Trainable params: 92,802
Non-trainable params: 0
_________________________________________________________________
```

Training and validation loss and accuracy:

```
Epoch 1/5
860/860 [==============================] - 48s 55ms/step - loss: 0.5138 - accuracy: 0.7741 - val_loss: 0.3059 - val_accuracy: 0.8723
Epoch 2/5
860/860 [==============================] - 48s 55ms/step - loss: 0.2616 - accuracy: 0.8944 - val_loss: 0.1713 - val_accuracy: 0.9367
Epoch 3/5
860/860 [==============================] - 48s 55ms/step - loss: 0.1683 - accuracy: 0.9377 - val_loss: 0.1235 - val_accuracy: 0.9553
Epoch 4/5
860/860 [==============================] - 48s 56ms/step - loss: 0.1211 - accuracy: 0.9566 - val_loss: 0.0796 - val_accuracy: 0.9721
Epoch 5/5
860/860 [==============================] - 48s 56ms/step - loss: 0.0971 - accuracy: 0.9658 - val_loss: 0.0635 - val_accuracy: 0.9784
```

Plot for training and validation loss and accuracy:



Test loss and accuracy:

```
157/157 [==============================] - 2s 10ms/step - loss: 0.0765 - accuracy: 0.9720
```

Hyper-parameter Tuning

1. **Changing the network architecture:** For new model added following layers:
    i. A convolution layer with 64 filters of size 3x3 and ReLU activation
    ii. A max pooling layer with pool size (2, 2)
    iii. A convolution layer with 64 filters of size 3x3 and ReLU activation
    iv. A dropout layer with rate = 0.3
    v. A flatten layer
    vi. A fully connected layer with 64 units and ReLU activation

vii. A fully connected layer with 32 units and ReLU activation

viii. A fully connected layer with 16 units and ReLU activation

ix. A fully connected layer with 2 units and sigmoid activation as we have 2 labels to classify.

By default, taking stride = 1, dilation = 1, optimizer = SGD, loss function = binary cross entropy, dropout rate = 0.3, weight initializer = Glorot uniform, epochs = 5.

Here is model created:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7744)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                495680
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 34
=================================================================
Total params: 535,890
Trainable params: 535,890
Non-trainable params: 0
_________________________________________________________________
```

Final Loss and Accuracy:

	Loss	Accuracy
Training	0.0727	0.9741
Validation	0.0478	0.9838
Testing	0.0556	0.9802

Here, we can see clearly that the new model has lower loss and higher accuracy than the previous model.

2. **Changing the receptive field:** For changing the receptive field, tried various dilation rates like 2, 3. Here, we can see clearly that the new models have higher loss and lower accuracy than the model with default parameters. The best accuracy was obtained by default dilation rate = 1.

   Final Loss and Accuracy for dilation rate = 2:

	Loss	Accuracy
Training	0.0781	0.9727
Validation	0.0556	0.9803
Testing	0.0657	0.9768

   Final Loss and Accuracy for dilation rate = 3:

	Loss	Accuracy
Training	0.0956	0.9666
Validation	0.0774	0.9714
Testing	0.0840	0.9684

3. **Changing the stride parameter:** Tried various strides like 2, 3. Here, we can see clearly that the new models have higher loss and lower accuracy than the model with default parameters. The best accuracy was obtained by default stride = 1.

   Final Loss and Accuracy for strides = 2:

	Loss	Accuracy
Training	0.1882	0.9265
Validation	0.1331	0.9518
Testing	0.1347	0.9514

   Final Loss and Accuracy for strides = 3:

	Loss	Accuracy
Training	0.3187	0.8704
Validation	0.2554	0.9024
Testing	0.2645	0.8974

4. **Changing the optimizer:** Tried various optimizers like Adam, RMSProp. Here, we can see clearly that the new models have lower loss and higher accuracy than the model with default parameters. The best accuracy is obtained by optimizer = Adam. Therefore, will be using Adam optimizer for all models ahead.

   Final Loss and Accuracy for optimizer = Adam:

	Loss	Accuracy
Training	0.0140	0.9954
Validation	0.0183	0.9942
Testing	0.0260	0.9910

   Final Loss and Accuracy for optimizer = RMSProp:

	Loss	Accuracy
Training	0.0193	0.9938
Validation	0.0186	0.9941
Testing	0.0236	0.9932

5. **Changing the loss function:** Tried loss functions like poisson, categorical cross entropy. The least loss and best accuracy were obtained by binary cross entropy (default).

   Final Loss and Accuracy for loss function = poisson:

	Loss	Accuracy
Training	0.5068	0.9953
Validation	0.5095	0.9940
Testing	0.5118	0.9924

   Final Loss and Accuracy for loss function = categorical cross entropy:

	Loss	Accuracy
Training	0.0115	0.9962
Validation	0.0190	0.9940
Testing	0.0237	0.9918

6. **Changing the dropout parameter:** Tried dropout amount like 0.5, 0.7. The least loss and best accuracy were obtained by default dropout = 0.3.

   Final Loss and Accuracy for dropout = 0.5:

	Loss	Accuracy
Training	0.0161	0.9943
Validation	0.0185	0.9935
Testing	0.0215	0.9928

   Final Loss and Accuracy for dropout = 0.7:

	Loss	Accuracy
Training	0.0239	0.9919
Validation	0.0175	0.9940
Testing	0.0209	0.9936

7. **Changing the learning rate:** Tried learning rates like 0.1, 0.01. The least loss and best accuracy were obtained by default learning rate = 0.001.

   Final Loss and Accuracy for dropout = 0.1:

	Loss	Accuracy
Training	0.6944	0.5021
Validation	0.6934	0.5074
Testing	0.6932	0.5100

   Final Loss and Accuracy for dropout = 0.01:

	Loss	Accuracy
Training	0.0333	0.9894
Validation	0.0225	0.9935
Testing	0.0318	0.9886

8. **Using weight initializer:** Tried weight initializers like He uniform, He normal. The least loss and best accuracy were obtained by He normal.

   Final Loss and Accuracy for weight initializer = He uniform:

	Loss	Accuracy
Training	0.0218	0.9921
Validation	0.0200	0.9937
Testing	0.0224	0.9920

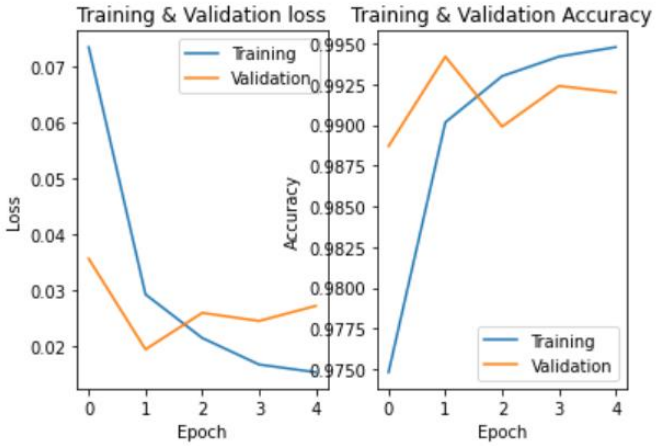
Final Loss and Accuracy for weight initializer = He normal:

	Loss	Accuracy
Training	0.0130	0.9956
Validation	0.0160	0.9946
Testing	0.0207	0.9946

9. **Adding batch normalization:** Added batch normalization layers after convolution and max pooling layers. Not much change was observed in the loss or accuracy.

	Loss	Accuracy
Training	0.0155	0.9948
Validation	0.0273	0.9920
Testing	0.0299	0.9906

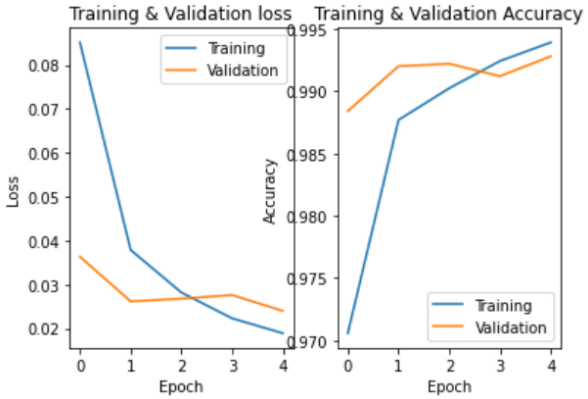
A spike is observed in 2nd epoch and then some fluctuating behavior is noticed.



10. **Adding layer normalization:** Added layer normalization layers after convolution layers. Not much change was observed in the loss or accuracy.

	Loss	Accuracy
Training	0.0189	0.9939
Validation	0.0240	0.9928
Testing	0.0272	0.9916

In 4th epoch some fluctuating behavior is noticed.



11. **Changing the epochs:** Tried epochs like 10, 15. The least loss and best accuracy were obtained by epochs = 15. But the concern here is that it might overfit the model so taking epoch = 5 is better.

Final Loss and Accuracy for epochs = 10:

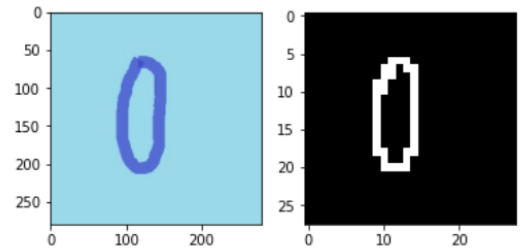
	Loss	Accuracy
Training	0.0057	0.9980
Validation	0.0302	0.9917
Testing	0.0378	0.9910

Final Loss and Accuracy for epochs = 15:

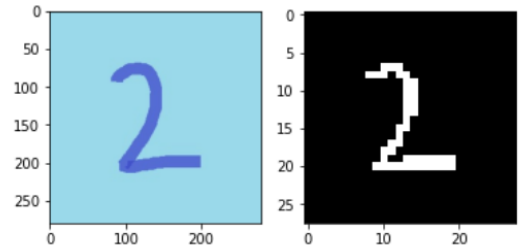
	Loss	Accuracy
Training	0.0062	0.9979
Validation	0.0201	0.9947
Testing	0.0219	0.9936

Inference

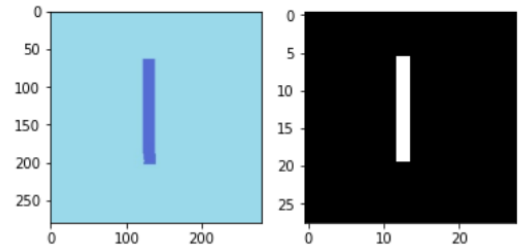
Here are the inference results obtained using images of handwritten digits. 9 out of 10 labels are predicted correctly. The only incorrect label is for 0. This could be because no model can be 100% accurate and some overfitting.



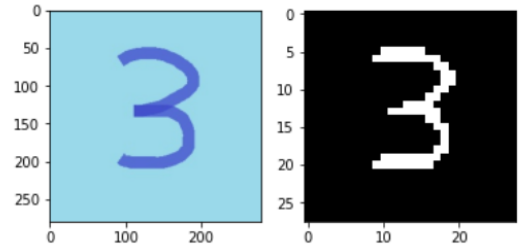
Digit is ODD



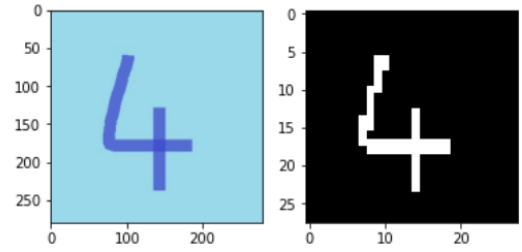
Digit is EVEN



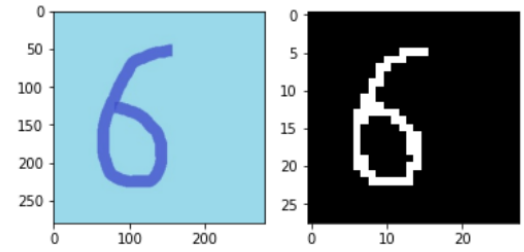
Digit is ODD



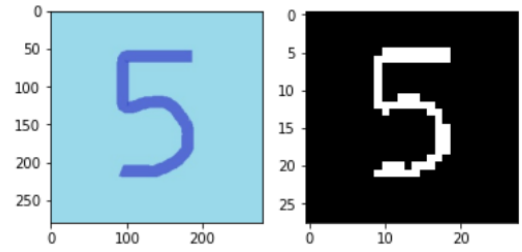
Digit is ODD



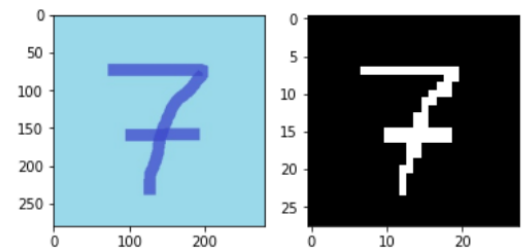
Digit is EVEN



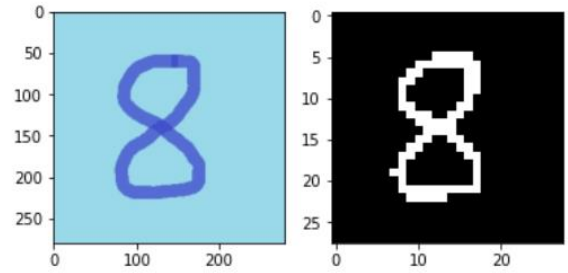
Digit is EVEN



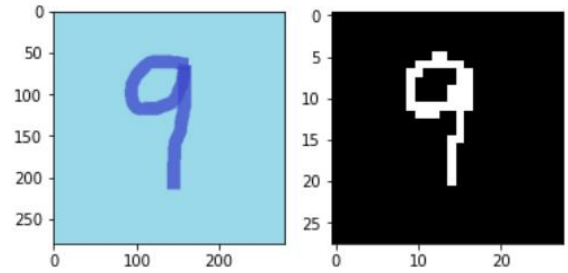
Digit is ODD



Digit is ODD



```
Digit is EVEN
```



```
Digit is ODD
```

5 References

Keras Layers: https://keras.io/api/layers/