Contents

Problem 1: Implementing a Neural Network with TensorFlow 2

Problem 2: Representing a Convolutional Neural Network as a Fully Connected Neural Network .. 4

Problem 1: Implementing a Neural Network with TensorFlow

Network Implementation:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 256)               803072
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 874,826
Trainable params: 874,826
Non-trainable params: 0
```

Output:

```
Epoch 1/10
860/860 [==============================] - 35s 40ms/step - loss: 0.5841 - accuracy: 0.7884 - val_loss: 0.3255 - val_accuracy: 0.8828

Epoch 00001: val_loss improved from inf to 0.32553, saving model to model.weights.best.hdf5
Epoch 2/10
860/860 [==============================] - 36s 42ms/step - loss: 0.3684 - accuracy: 0.8687 - val_loss: 0.2789 - val_accuracy: 0.9020

Epoch 00002: val_loss improved from 0.32553 to 0.27893, saving model to model.weights.best.hdf5
Epoch 3/10
860/860 [==============================] - 35s 41ms/step - loss: 0.3166 - accuracy: 0.8851 - val_loss: 0.2553 - val_accuracy: 0.9060

Epoch 00003: val_loss improved from 0.27893 to 0.25535, saving model to model.weights.best.hdf5
Epoch 4/10
860/860 [==============================] - 35s 41ms/step - loss: 0.2882 - accuracy: 0.8969 - val_loss: 0.2341 - val_accuracy: 0.9132

Epoch 00004: val_loss improved from 0.25535 to 0.23407, saving model to model.weights.best.hdf5
Epoch 5/10
860/860 [==============================] - 34s 40ms/step - loss: 0.2636 - accuracy: 0.9038 - val_loss: 0.2234 - val_accuracy: 0.9178

Epoch 00005: val_loss improved from 0.23407 to 0.22336, saving model to model.weights.best.hdf5
Epoch 6/10
860/860 [==============================] - 34s 40ms/step - loss: 0.2502 - accuracy: 0.9100 - val_loss: 0.2156 - val_accuracy: 0.9186

Epoch 00006: val_loss improved from 0.22336 to 0.21561, saving model to model.weights.best.hdf5
Epoch 7/10
860/860 [==============================] - 34s 39ms/step - loss: 0.2346 - accuracy: 0.9151 - val_loss: 0.2020 - val_accuracy: 0.9262

Epoch 00007: val_loss improved from 0.21561 to 0.20196, saving model to model.weights.best.hdf5
Epoch 8/10
860/860 [==============================] - 34s 40ms/step - loss: 0.2253 - accuracy: 0.9167 - val_loss: 0.2238 - val_accuracy: 0.9144

Epoch 00008: val_loss did not improve from 0.20196
Epoch 9/10
860/860 [==============================] - 33s 39ms/step - loss: 0.2148 - accuracy: 0.9208 - val_loss: 0.2053 - val_accuracy: 0.9228

Epoch 00009: val_loss did not improve from 0.20196
Epoch 10/10
860/860 [==============================] - 33s 39ms/step - loss: 0.2081 - accuracy: 0.9240 - val_loss: 0.2018 - val_accuracy: 0.9244

Epoch 00010: val_loss improved from 0.20196 to 0.20184, saving model to model.weights.best.hdf5

 Test accuracy: 91.839998960495 %
```

Accuracy – 91.839998%

Problem 2: Representing a Convolutional Neural Network as a Fully Connected Neural Network

TensorFlow Model:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
activation (Activation)      (None, 13, 13, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              11076608
_________________________________________________________________
dense_4 (Dense)              (None, 10)                10250
=================================================================
Total params: 11,087,498
Trainable params: 11,087,498
Non-trainable params: 0
```

Output of TensorFlow Model:

```
Epoch 1/10
860/860 [==============================] - 57s 66ms/step - loss: 0.3716 - accuracy: 0.8666 - val_loss: 0.2719 - val_accuracy: 0.9002

Epoch 00001: val_loss improved from inf to 0.27187, saving model to model1.weights.best.hdf5
Epoch 2/10
860/860 [==============================] - 55s 64ms/step - loss: 0.2441 - accuracy: 0.9098 - val_loss: 0.2450 - val_accuracy: 0.9102

Epoch 00002: val_loss improved from 0.27187 to 0.24498, saving model to model1.weights.best.hdf5
Epoch 3/10
860/860 [==============================] - 62s 72ms/step - loss: 0.1903 - accuracy: 0.9291 - val_loss: 0.2249 - val_accuracy: 0.9208

Epoch 00003: val_loss improved from 0.24498 to 0.22491, saving model to model1.weights.best.hdf5
Epoch 4/10
860/860 [==============================] - 54s 63ms/step - loss: 0.1504 - accuracy: 0.9438 - val_loss: 0.2310 - val_accuracy: 0.9196

Epoch 00004: val_loss did not improve from 0.22491
Epoch 5/10
860/860 [==============================] - 53s 62ms/step - loss: 0.1203 - accuracy: 0.9551 - val_loss: 0.2374 - val_accuracy: 0.9180

Epoch 00005: val_loss did not improve from 0.22491
Epoch 6/10
860/860 [==============================] - 52s 61ms/step - loss: 0.0939 - accuracy: 0.9649 - val_loss: 0.2536 - val_accuracy: 0.9210

Epoch 00006: val_loss did not improve from 0.22491
Epoch 7/10
860/860 [==============================] - 52s 61ms/step - loss: 0.0782 - accuracy: 0.9719 - val_loss: 0.2925 - val_accuracy: 0.9216

Epoch 00007: val_loss did not improve from 0.22491
Epoch 8/10
860/860 [==============================] - 53s 61ms/step - loss: 0.0577 - accuracy: 0.9797 - val_loss: 0.3052 - val_accuracy: 0.9174

Epoch 00008: val_loss did not improve from 0.22491
Epoch 9/10
860/860 [==============================] - 55s 64ms/step - loss: 0.0477 - accuracy: 0.9827 - val_loss: 0.3309 - val_accuracy: 0.9220

Epoch 00009: val_loss did not improve from 0.22491
Epoch 10/10
860/860 [==============================] - 55s 64ms/step - loss: 0.0382 - accuracy: 0.9865 - val_loss: 0.3180 - val_accuracy: 0.9218

Epoch 00010: val_loss did not improve from 0.22491

 Test accuracy: 90.99000096321106 %
```

Accuracy – 90.990000%

Fully Connected Neural Network Implementation:

The weights were obtained by training the TensorFlow model, which were then converted to the weight matrix of a fully connected type of network. This fully connected network has input size of 784 (28*28) neurons. The obtained weight matrix is of the size 784*43264 (28*28,26*26*64), i.e. applying 64 convolutional filters of kernel size 3*3 on the input image. All these filter outputs were flattened which gives out 43264 neurons in the next layer.

Now Maxpooling was applied to this layer having 43264 input neurons. This Maxpooling operation results in the next layer having 10816 (13*13*64) neurons.

---- For convolution and max pooling step we assumed that in the flattened input, the first 676 (26*26) neurons are obtained by applying the 1st convolutional filter to the input image and the next 676 neurons (26*26) are obtained by applying the 2nd convolutional filter to the input image, and so on. This will generate an input layer of 43264 neurons (26*26*64) since there are 64 filters. Please take note that this assumed flattening is different from the default way in which the tf.keras.layer.Flatten() function flattens the resultant neurons after applying convolution.

---- After, Maxpooling step, this disparity in flattening was handled to match that of the default output obtained by tf.keras.layer.Flatten(), since during the training process TensorFlow takes that default flattening and gives out the weight matrices which are to be used in the next step.

Next, the output after re-arranging the flattening, is used as an input to a two-layer fully connected feed forward network. The weights of each of the layers are derived directly from the TensorFlow model.

Comparison between the Predictive Distribution of CNN and FNN:

```
--------------------------------------------------
Output of Tensorflow Model
--------------------------------------------------
[2.6691807e-04 1.9208780e-06 2.3793539e-01 5.3878011e-06 7.3450303e-01
 9.7538191e-07 2.6908152e-02 1.4588852e-07 3.7617437e-04 1.9338640e-06]

--------------------------------------------------
Output of Fully Connected Model
--------------------------------------------------
[2.66917924e-04 1.92088033e-06 2.37935392e-01 5.38780170e-06
 7.34502999e-01 9.75379776e-07 2.69081524e-02 1.45888398e-07
 3.76174608e-04 1.93386079e-06]
```