**Project 2: Neural Networks and machine learning Model**

**Student Name: Sireesha Maguluri**

**Student ID: 005007985**

**Subject: MSDS534 - M50 Deep Learning**

**Professor: Dr. Yufeng Zheng**

**Importing TensorFlow Modules:**

**# Set up a tensor flow**

import tensorflow as tf

print ("TensorFlow version:", tf.\_\_version\_\_)

import numpy as np

from tensorflow import keras

**#for tensorboard**

from tensorflow.keras.callbacks import TensorBoard

tensorboard\_callback = TensorBoard('.logdir')

A screenshot of a computer code

Description automatically generated with low confidence

Figure 1: necessary modules for building, training and evaluate model.

By running this code, you will have TensorFlow set up along with the necessary libraries and the TensorBoard callback for monitoring your training progress.

**# network and training:**

EPOCHS = 5

BATCH\_SIZE = 128

VERBOSE = 1

NB\_CLASSES = 10   # number of outputs = number of digits

N\_HIDDEN = 128

VALIDATION\_SPLIT=0.2 # how much TRAIN is reserved for VALIDATION

DROPOUT = 0.3

The above lines are for network and training code to control various aspects of the training process, such as the number of epochs, batch size, and model architecture.A screenshot of a computer

Description automatically generated with low confidence

Figure 2: Network and Training

1. **Loading data set:**

**#Loading data set**

**# loading MNIST dataset**

**# verify**

**# the split between train and test is 60,000, and 10,000 respectly**

**# one-hot is automatically applied**

mnist = keras.datasets.mnist

(X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

**Output:** Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 [==============================] - 0s 0us/step

A picture containing text, font, screenshot

Description automatically generated

Figure 3: Loading dataset.

By executing this code, the MNIST dataset is loaded and split into training and testing sets, ready for further processing and model training.

**#X\_train is 60000 rows of 28x28 values --> reshaped in 60000 x 784**

**RESHAPED = 784**

**#**

X\_train = X\_train.reshape(60000, RESHAPED)

X\_test = X\_test.reshape(10000, RESHAPED)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

**#normalize in [0,1]**

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

print(X\_train.shape[0], 'train samples')

print(X\_test.shape[0], 'test samples')

**#one-hot**

Y\_train = tf.keras.utils.to\_categorical(Y\_train, NB\_CLASSES)

Y\_test = tf.keras.utils.to\_categorical(Y\_test, NB\_CLASSES)

**Output:**60000 train samples

10000 test samples

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 4: Preprocessing Data set

The above code performs the following steps:

1. Reshape the input data to flatten the images into 1D arrays.
2. Convert the data type to float32.
3. Normalize the pixel values by dividing them by 255.0.
4. Print the number of samples in the training and testing sets.
5. Perform one-hot encoding on the target labels.

These preprocessing steps are commonly used when working with image datasets and are essential for preparing the data to be fed into a neural network for training and testing.

1. **Build a neural network machine learning model that classifies the images.**

**#Build a machine-learning model**

model = tf.keras.models.Sequential()

model.add(keras.layers.Dense(N\_HIDDEN, input\_shape=(RESHAPED,),

      name='dense\_layer', activation='relu'))

model.add(keras.layers.Dropout(DROPOUT))

model.add(keras.layers.Dense(N\_HIDDEN, name='dense\_layer\_2', activation='relu'))

model.add(keras.layers.Dropout(DROPOUT))

model.add(keras.layers.Dense(NB\_CLASSES, name='dense\_layer\_3', activation='softmax'))

# Summary of the model

model.summary()

Output:

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_layer (Dense) (None, 128) 100480

dropout\_8 (Dropout) (None, 128) 0

dense\_layer\_2 (Dense) (None, 128) 16512

dropout\_9 (Dropout) (None, 128) 0

dense\_layer\_3 (Dense) (None, 10) 1290

=================================================================

Total params: 118,282

Trainable params: 118,282

Non-trainable params: 0

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 5: Building a model and Summary of the model.

The code creates a Sequential model and adds layers more compactly using a list. The layers include fully connected (dense) layers with the specified number of neurons, activation functions, and dropout layers for regularization.

The model. summary () statement is included to print an overview of the model's architecture.

This code will build a machine-learning model with the specified layers and activation functions, ready to be trained on the MNIST dataset.

1. **Train this neural network:**

# compiling the model

model.compile(optimizer='Adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

#training the model

model.fit(X\_train, Y\_train,

    batch\_size=BATCH\_SIZE, epochs=EPOCHS,

    verbose=VERBOSE, validation\_split=VALIDATION\_SPLIT,

    callbacks=[tensorboard\_callback])

**Output:**

Epoch 1/5

375/375 [==============================] - 4s 8ms/step - loss: 0.5005 - accuracy: 0.8473 - val\_loss: 0.1878 - val\_accuracy: 0.9443

Epoch 2/5

375/375 [==============================] - 3s 9ms/step - loss: 0.2310 - accuracy: 0.9317 - val\_loss: 0.1402 - val\_accuracy: 0.9592

Epoch 3/5

375/375 [==============================] - 3s 9ms/step - loss: 0.1771 - accuracy: 0.9463 - val\_loss: 0.1160 - val\_accuracy: 0.9663

Epoch 4/5

375/375 [==============================] - 3s 8ms/step - loss: 0.1464 - accuracy: 0.9550 - val\_loss: 0.1008 - val\_accuracy: 0.9707

Epoch 5/5

375/375 [==============================] - 3s 8ms/step - loss: 0.1279 - accuracy: 0.9608 - val\_loss: 0.0973 - val\_accuracy: 0.9709

<keras.callbacks.History at 0x7f318a7b2fb0>

A screenshot of a computer

Description automatically generated with medium confidenceThis step trains the model using the training data. The X\_train and Y\_train datasets are provided as input. The batch\_size parameter determines the number of samples per gradient update, epochs specifies the number of training iterations, verbose controls the verbosity mode during training, validation\_split determines the fraction of the training data used for validation, and callbacks specifies any additional callbacks to be used during training (in this case, the TensorBoard callback).

Figure 6: Training the model.

This code will compile the model with the specified optimizer, loss function, and metrics. Then, it will be trained on the MNIST dataset using the provided training data, batch size, and number of epochs. The training progress and performance will be logged to the TensorBoard log directory specified earlier.

1. **Evaluate the accuracy of the model.**

**#evalute the model**

test\_loss, test\_acc = model.evaluate(X\_test, Y\_test)

print('Test accuracy:', test\_acc)

**# making prediction**

predictions = model.predict(X\_test)

predictions

**Output:**

313/313 [==============================] - 1s 2ms/step - loss: 0.0915 - Accuracy: 0.9710

Test accuracy: 0.9710000157356262

313/313 [==============================] - 1s 2ms/step

array ([[2.93383351e-07, 1.90402989e-06, 7.87562094e-05, ..., 9.99830723e-01, 3.52416123e-07, 3.25357396e-05], [1.21880760e-07, 3.92998889e-04, 9.99580085e-01, ..., 1.18285373e-08, 1.14662589e-05, 7.15899978e-11], [4.41169391e-07, 9.99626100e-01, 8.29816781e-05, ..., 1.05709405e-04, 5.23845738e-05, 3.66460313e-06], …., [1.26874497e-07, 1.05511538e-06, 1.04957991e-07, ..., 1.05550083e-04, 8.63354398e-06, 1.77111229e-04], [9.56634722e-07, 2.28138344e-08, 5.15385679e-10, ..., 1.96858210e-08, 2.99346066e-05, 1.79865323e-09], [6.25170344e-07, 7.86595511e-10, 5.44064414e-06, ...,

1.52742624e-10, 9.56569401e-09, 9.18203624e-10]], dtype=float32)

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 7: Evaluate the accuracy of the code.

This code uses the evaluate method to compute the test loss and accuracy by providing the test data (X\_test and Y\_test). The resulting test accuracy is then printed.

Next, the prediction method generates test data (X\_test) predictions. The projections are stored in the predictions variable and printed.

By running this code, you can evaluate the model's performance on the test dataset and obtain the predicted outputs for the test samples.

**References:**

1.Gulli, A., Kapoor, A., Pal, S. (2019). Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and More with TensorFlow 2 and the Keras API, 2nd Edition. United Kingdom: Packt Publishing.