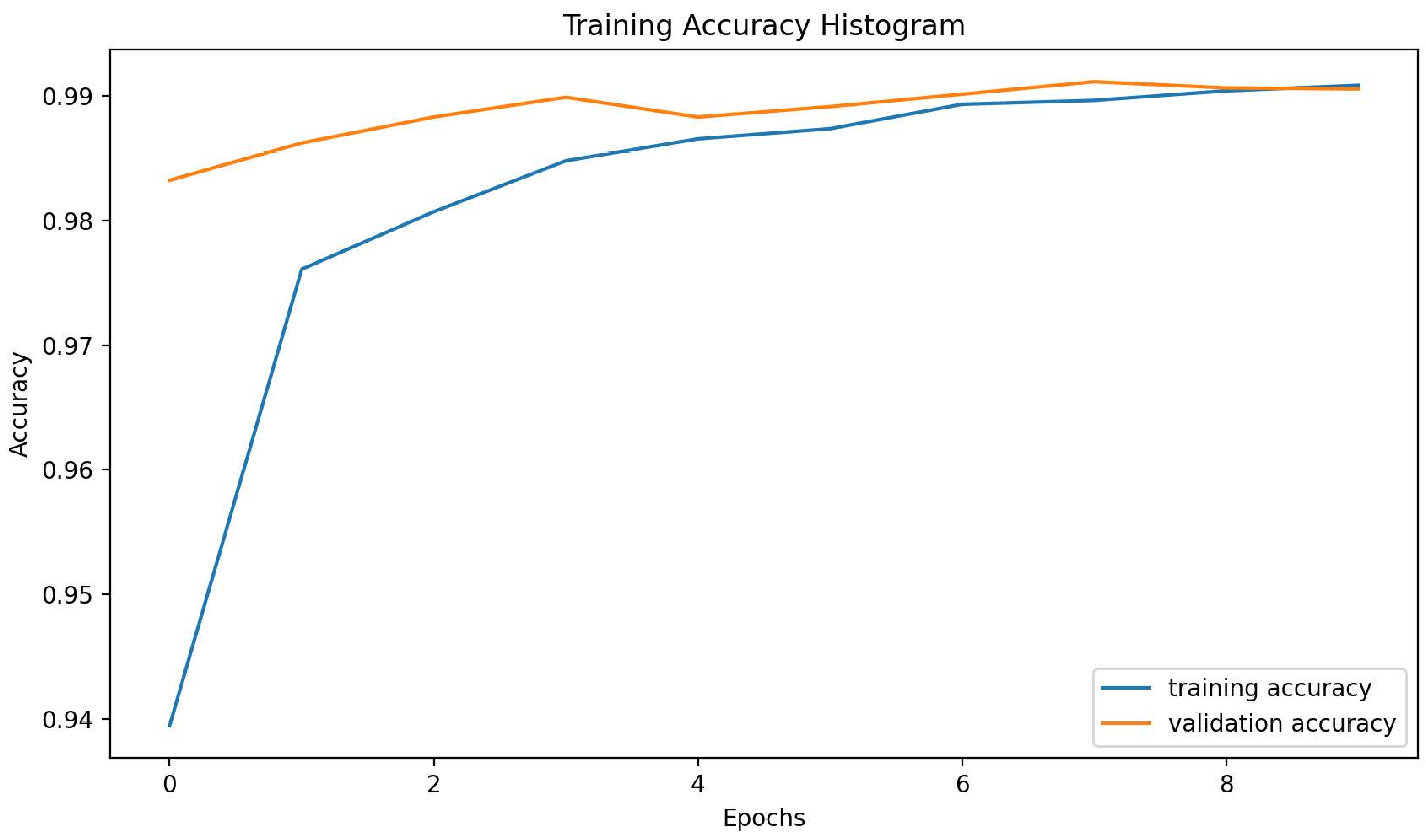
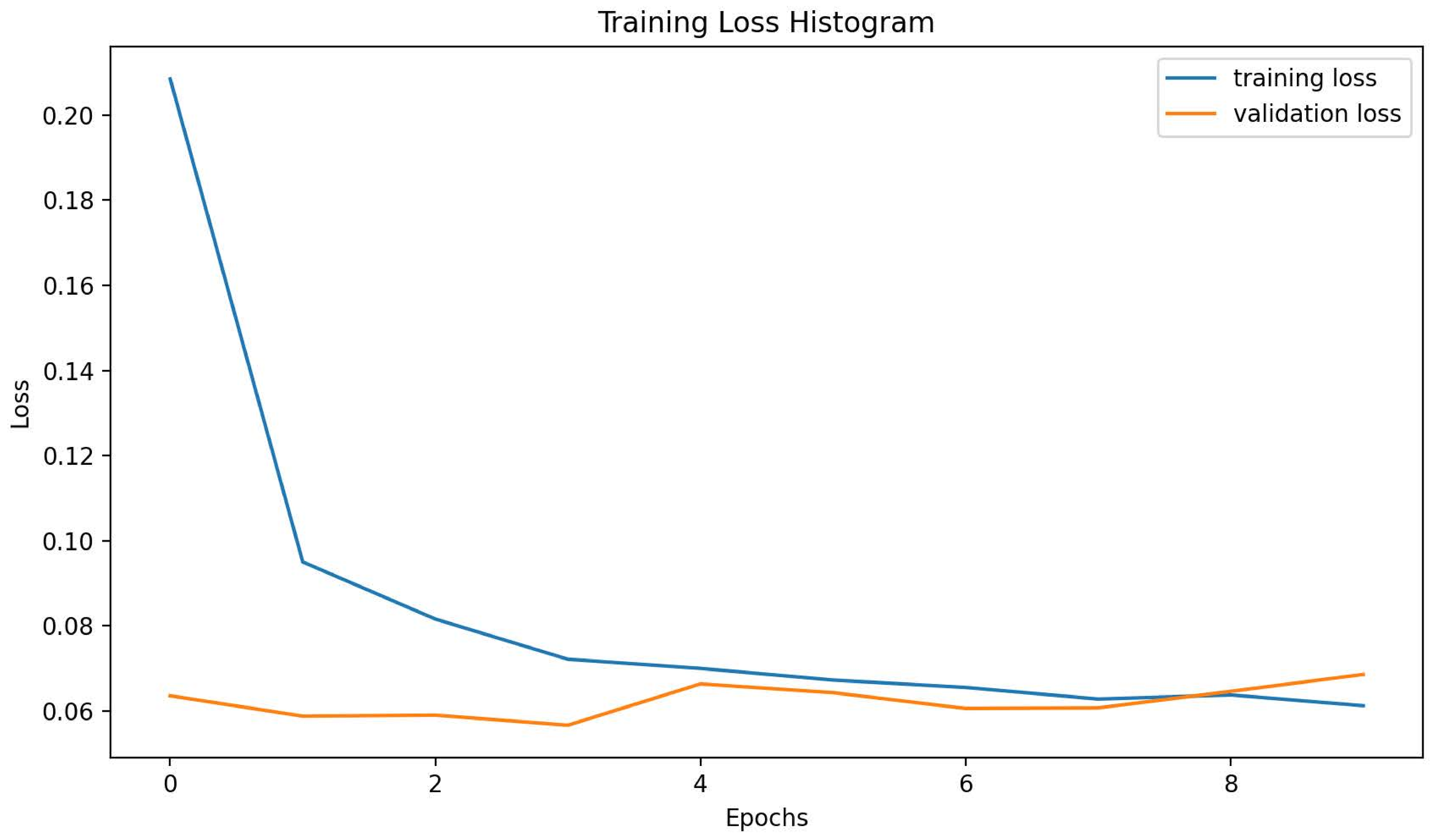
main.py

```python
#!/bin/env python3.9

"""
Allister Liu
"""

import sys
import os

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd

from absl import flags

from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense

FLAGS = flags.FLAGS
flags.DEFINE_integer("sample_size", 1000, "Number of samples in dataset")
flags.DEFINE_integer("batch_size", 32, "Number of samples in batch")
flags.DEFINE_integer("num_iters", 500, "Number of epochs")
flags.DEFINE_integer("random_seed", 31415, "Random seed")


def import_data(rng):
    """
    Import data from the two csv files ("./mnist_train.csv" and "./mnist_test.csv"), separate the label from pixel-wise
    grayscale value of each image, and put them into numpy arrays. Return after shuffling
    ---------------------------------------------------------------------------------------------------------------
    :param rng: random generator
        :return: shuffled data in numpy arrays
    """
    # import data from csv to pandas dataframe
    #   data from Kaggle:
    #       https://www.kaggle.com/datasets/oddrationale/mnist-in-csv
    #   originally using https://www.kaggle.com/competitions/digit-recognizer/ but changed to the new dataset due to
    # the lack of labels in test data
    train_val_data_df = pd.read_csv("./mnist_train.csv")
    test_data_df = pd.read_csv("./mnist_test.csv")

    # put data into numpy array
    train_val_data_arr = np.array(train_val_data_df)
    test_data_arr = np.array(test_data_df)

    # shuffle the data
    rng.shuffle(train_val_data_arr)
    rng.shuffle(test_data_arr)

    # separate labels and image data
    train_val_labels = train_val_data_arr[:, 0]     # shape=(60000,) the first column is the label
    train_val_pixels = train_val_data_arr[:, 1:]    # shape=(60000, 784) grayscale values of each pixel
    test_labels = test_data_arr[:, 0]               # shape=(10000,)
    test_pixels = test_data_arr[:, 1:]              # shape=(10000, 784)

    return train_val_pixels, train_val_labels, test_pixels, test_labels


def preprocess(train_val_pixels, train_val_labels, test_pixels, test_labels):
    """
    Preprocess data to get it ready for training:
        - normalize pixel-wise grayscale value to between 0 and 1
        - reshape the input grayscale values for each image from (784,) => (28, 28)
        - add a channel for grayscale value tf.newaxis
        - split into train, validation, and test dataset
    ----------------------------------------------------------------------------------------------
    :param train_val_pixels: pixel-wise grayscale value of each image for training and validation
    :param train_val_labels: label of each image for training and validation
    :param test_pixels: pixel-wise grayscale value of each image for testing
    :param test_labels: label of each image for testing
        :return: (train_x, train_y), (validation_x, validation_y), (test_x, test_y)
    """
    # normalize the pixel grayscale value to between 0 and 1 by dividing by 255.
    train_val_pixels_normalized = train_val_pixels / 255.
    test_pixels_normalized = test_pixels / 255.

    # reshape the 1d array of each image to 2d (784,) => (28, 28)
    #   suggested by Bob (Sangjoon) Lee
    train_val_pixels_processed = np.array([np.reshape(xs, (28, 28)) for xs in train_val_pixels_normalized])
    test_pixels_processed = np.array([np.reshape(xs, (28, 28)) for xs in test_pixels_normalized])

    # one-hot encode the class labels
    # labels_onehot = np.zeros((len(labels), labels.max() + 1))  # labels.max()+1 = number of classes => 10 classes
    # labels_onehot[np.arange(labels.size), labels] = 1

    # split the data => 80% train + 20% validation
    train_range = range(0, int(0.8 * len(train_val_labels)))
    val_range = range(int(0.8 * len(train_val_labels)), len(train_val_labels))

    train_pix_arr = train_val_pixels_processed[train_range]     # shape=(48000, 28, 28)
    train_lbl_arr = train_val_labels[train_range]               # shape=(48000, 1)
    val_pix_arr = train_val_pixels_processed[val_range]         # shape=(12000 , 28, 28)
    val_lbl_arr = train_val_labels[val_range]                   # shape=(12000 , 1)
    test_pix_arr = test_pixels_processed                        # shape=(10000 , 28, 28)
    test_lbl_arr = test_labels                                  # shape=(10000 , 1)

    # add an additional channel for grayscale value of the images
    #   https://medium.com/@nutanbhogendrasharma/tensorflow-build-custom-convolutional-neural-network-with-mnist-dataset-d4c36cd52114
    train_pix_arr = train_pix_arr[..., tf.newaxis].astype('float32')    # shape=(48000, 28, 28, 1)
    val_pix_arr = val_pix_arr[..., tf.newaxis].astype('float32')        # shape=(12000 , 28, 28, 1)
    test_pix_arr = test_pix_arr[..., tf.newaxis].astype('float32')      # shape=(10000 , 28, 28, 1)

    return train_pix_arr, train_lbl_arr, val_pix_arr, val_lbl_arr, test_pix_arr, test_lbl_arr


def get_model():
    """
    build a CNN model with conv2D, maxPool2D, dropout, flatten, and dense
    using adam as optimizer, sparse categorical cross entropy for loss function
    -----------------------------------------------------------------
    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
    _________________________________________________________________
    dropout (Dropout)            (None, 11, 11, 64)        0
    _________________________________________________________________
    flatten (Flatten)            (None, 7744)              0
    _________________________________________________________________
    dense (Dense)                (None, 128)               991360
    _________________________________________________________________
    dropout_1 (Dropout)          (None, 128)               0
    _________________________________________________________________
    dense_1 (Dense)              (None, 10)                1290
    =================================================================
    Total params: 1,011,466
    Trainable params: 1,011,466
    Non-trainable params: 0
    _________________________________________________________________
        :return: compiled CNN model
    """
    # modified from:
    # https://medium.com/@nutanbhogendrasharma/tensorflow-build-custom-convolutional-neural-network-with-mnist-dataset-d4c36cd52114
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1),
                     kernel_regularizer=tf.keras.regularizers.l2(l2=.00001)))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
                     kernel_regularizer=tf.keras.regularizers.l2(.00001)))
    model.add(Dropout(.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(.00001)))
    model.add(Dropout(.5))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model


if __name__ == "__main__":
    # There is some mismatch version issues with my installed CuDNN and CUDA Toolkit, so I decided not to run on
    # CPU only and disable GPU
    os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

    # Handle the flags
    FLAGS(sys.argv)
    SAMPLE_SIZE = FLAGS.sample_size
    BATCH_SIZE = FLAGS.batch_size
    NUM_ITERS = FLAGS.num_iters
    RNG_SEED = FLAGS.random_seed

    # Set rng seed
    np_rng = np.random.default_rng(RNG_SEED)
    tf.random.Generator.from_seed(RNG_SEED)

    # import and preprocess data
    x_train_val, y_train_val, x_test, y_test = import_data(rng=np_rng)
    train_x, train_y, val_x, val_y, test_x, test_y = preprocess(train_val_pixels=x_train_val,
                                                                train_val_labels=y_train_val,
                                                                test_pixels=x_test,
                                                                test_labels=y_test)

    # train and evaluate model
    myModel = get_model()
    print(myModel.summary())
    hist = myModel.fit(x=train_x, y=train_y, batch_size=BATCH_SIZE, epochs=NUM_ITERS,
                       validation_data=(val_x, val_y), verbose=1)
    test_loss, test_acc = myModel.evaluate(x=test_x, y=test_y, verbose=1)

    print('Test loss\t\t:', test_loss)
    print('Test accuracy\t:', test_acc)

    # plotting the training accuracy and loss
    fig, axs = plt.subplots(2, 1, figsize=(10, 12), dpi=200)
    axs[0].set_title('Training Accuracy Histogram')
    axs[0].set_xlabel('Epochs')
    axs[0].set_ylabel('Accuracy')
    axs[0].plot(hist.history['accuracy'], label='training accuracy')
    axs[0].plot(hist.history['val_accuracy'], label='validation accuracy')
    axs[0].legend(loc='lower right')

    axs[1].set_title('Training Loss Histogram')
    axs[1].set_xlabel('Epochs')
    axs[1].set_ylabel('Loss')
    axs[1].plot(hist.history['loss'], label='training loss')
    axs[1].plot(hist.history['val_loss'], label='validation loss')
    axs[1].legend(loc='upper right')

    plt.show()

```


Training Accuracy Histogram
Accuracy
0.94
0.95
0.96
0.97
0.98
0.99
Epochs
0
2
4
6
8
training accuracy
validation accuracy


Training Loss Histogram
Loss
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
Epochs
0
2
4
6
8
training loss
validation loss

```
C:\Users\allis\AppData\Local\Programs\Python\Python39\python.exe D:/School/ECE472_DeepLearning/hw/ECE472_DeepLearning/hw3/main.py --sample_size 1000 --batch_size 32 --num_iters 10 --random_seed 31415
2022-09-21 18:58:42.689188: E tensorflow/stream_executor/cuda/cuda_driver.cc:265] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
2022-09-21 18:58:42.692451: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: PC-Arristar
2022-09-21 18:58:42.692596: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: PC-Arristar
2022-09-21 18:58:42.693009: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
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
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 7744)              0
_________________________________________________________________
dense (Dense)                (None, 128)               991360
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,011,466
Trainable params: 1,011,466
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.2085 - accuracy: 0.9395 - val_loss: 0.0636 - val_accuracy: 0.9833
Epoch 2/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.0950 - accuracy: 0.9761 - val_loss: 0.0588 - val_accuracy: 0.9862
Epoch 3/10
1500/1500 [==============================] - 17s 12ms/step - loss: 0.0816 - accuracy: 0.9807 - val_loss: 0.0590 - val_accuracy: 0.9883
Epoch 4/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0722 - accuracy: 0.9848 - val_loss: 0.0567 - val_accuracy: 0.9899
Epoch 5/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0700 - accuracy: 0.9866 - val_loss: 0.0664 - val_accuracy: 0.9883
Epoch 6/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0673 - accuracy: 0.9874 - val_loss: 0.0643 - val_accuracy: 0.9892
Epoch 7/10
1500/1500 [==============================] - 19s 12ms/step - loss: 0.0656 - accuracy: 0.9894 - val_loss: 0.0606 - val_accuracy: 0.9902
Epoch 8/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0628 - accuracy: 0.9897 - val_loss: 0.0608 - val_accuracy: 0.9912
Epoch 9/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0638 - accuracy: 0.9904 - val_loss: 0.0646 - val_accuracy: 0.9907
Epoch 10/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0613 - accuracy: 0.9909 - val_loss: 0.0686 - val_accuracy: 0.9906
313/313 [==============================] - 1s 3ms/step - loss: 0.0685 - accuracy: 0.9908
Test loss       : 0.06848642230033875
Test accuracy   : 0.9908000230789185

Process finished with exit code 0
```