Amaan Rahman
ECE472: Deep Learning
Processor Curro

Assignment 3: MNIST Classifier

Architecture:

```
_________________________________________________________________
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, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 20)                16020
_________________________________________________________________
dropout (Dropout)            (None, 20)                0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                420
_________________________________________________________________
batch_normalization (BatchNo (None, 20)                80
_________________________________________________________________
dropout_1 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                1050
_________________________________________________________________
batch_normalization_1 (Batch (None, 50)                200
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 27,848
Trainable params: 27,708
Non-trainable params: 140
```

Output:

```
Epoch 1/5
1500/1500 [==============================] - 23s 15ms/step - loss: 2.2247 - accuracy: 0.8947 - val_loss: 0.6271 - val_accuracy: 0.9351
Epoch 2/5
1500/1500 [==============================] - 22s 15ms/step - loss: 0.6271 - accuracy: 0.9325 - val_loss: 0.5500 - val_accuracy: 0.9448
Epoch 3/5
1500/1500 [==============================] - 25s 17ms/step - loss: 0.5755 - accuracy: 0.9352 - val_loss: 0.4908 - val_accuracy: 0.9537
Epoch 4/5
1500/1500 [==============================] - 23s 15ms/step - loss: 0.5446 - accuracy: 0.9391 - val_loss: 0.4562 - val_accuracy: 0.9558
Epoch 5/5
1500/1500 [==============================] - 26s 17ms/step - loss: 0.5155 - accuracy: 0.9404 - val_loss: 0.4193 - val_accuracy: 0.9659
313/313 [==============================] - 2s 6ms/step - loss: 0.4191 - accuracy: 0.9647
```

Final test dataset accuracy: **96.47%**

```
import struct as st
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, regularizers
from tensorflow.keras.layers import (
    Dense,
    Conv2D,
    Dropout,
    Flatten,
    MaxPooling2D,
    BatchNormalization,
)

SHUFFLE_SIZE = 1000
BATCH_SIZE = 32
TRAIN_SPLIT = 0.8
EPOCHS = 5

dataset = {
    "train-images": "../datasets/MNIST_DATASET/train-images-idx3-ubyte",
    "train-labels": "../datasets/MNIST_DATASET/train-labels-idx1-ubyte",
    "test-images": "../datasets/MNIST_DATASET/t10k-images-idx3-ubyte",
    "test-labels": "../datasets/MNIST_DATASET/t10k-labels-idx1-ubyte",
}


def parseDataset():
    images_set = []
    labels_set = []
    for k, v in dataset.items():
        if "images" in k:
            with open(v, "rb") as f:
                magic, num_imgs, num_rows, num_cols = st.unpack(">IIII", f.read(1
6))
                images_set.append(
                    np.fromfile(f, dtype=np.dtype(np.ubyte))
                    .newbyteorder(">")
                    .reshape(num_imgs, num_rows, num_cols, 1)
                )
        elif "labels" in k:
            with open(v, "rb") as f:
                magic, num_items = st.unpack(">II", f.read(8))
                labels_set.append(
                    np.fromfile(f, dtype=np.dtype(np.uint8)).newbyteorder(">")
                )

    return ((images_set[0], labels_set[0]), (images_set[1], labels_set[1]))


def main():

    # load data:
    # [0] => IMAGES
    # [1] => LABELS
    train, test = parseDataset()
    normalized_train, normalized_test = (train[0] / 255.0, train[1]), (
        test[0] / 255.0,
        test[1],
    )

    model = models.Sequential()
    model.add(
```



```
        Conv2D(
            64,
            kernel_size=(3, 3),
            activation=tf.nn.elu,
            input_shape=(train[0][0].shape[0], train[0][0].shape[1], 1),
            kernel_regularizer=regularizers.l2(l2=0.01),
        )
    )
    model.add(MaxPooling2D())
    model.add(
        Conv2D(
            64,
            kernel_size=(3, 3),
            activation=tf.nn.elu,
            kernel_regularizer=regularizers.l2(l2=0.2),
        )
    )
    model.add(MaxPooling2D())
    model.add(Flatten())  # Flattening the 2D arrays for fully connected layers
    model.add(
        Dense(
            10, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.05)
        )
    )
    model.add(Dropout(0.05))
    model.add(
        Dense(
            20, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.3)
        )
    )
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(
        Dense(
            50, activation=tf.nn.leaky_relu, kernel_regularizer=regularizers.l2(
l2=0.3)
        )
    )
    model.add(BatchNormalization())
    model.add(Dense(10, activation="softmax", kernel_regularizer=regularizers.l2(
)))
    model.summary()

    model.compile(
        optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]
    )
    model.fit(
        x=normalized_train[0],
        y=normalized_train[1],
        batch_size=BATCH_SIZE,
        epochs=EPOCHS,
        steps_per_epoch=np.math.ceil(0.8 * train[0].shape[0] / BATCH_SIZE),
        validation_split=0.2,
    )
    model.evaluate(x=normalized_test[0], y=normalized_test[1])


if __name__ == "__main__":
    main()
```

```
compile:
	black mnist_classify.py
	flake8 --ignore=E,W mnist_classify.py
	python3 mnist_classify.py

pdf:
	a2ps mnist_classify.py -o mnist_classify.ps --pro=color
	a2ps Makefile -o Makefile.ps --pro=color
	ps2pdf mnist_classify.ps
	ps2pdf Makefile.ps
	gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=classif_AR.pdf ou
tput.pdf mnist_classify.pdf Makefile.pdf

clean:
	rm *.ps* bin_class.pdf Makefile.pdf
```