CPSC 501 A4

Teodor Tebeica

30046038

MNISTModel.py

```
--Fit model--
Epoch 1/10
469/469 - 1s - loss: 0.2829 - accuracy: 0.9193
Epoch 2/10
469/469 - 1s - loss: 0.1220 - accuracy: 0.9645
Epoch 3/10
469/469 - 1s - loss: 0.0834 - accuracy: 0.9748
Epoch 4/10
469/469 - 1s - loss: 0.0631 - accuracy: 0.9811
Epoch 5/10
469/469 - 1s - loss: 0.0501 - accuracy: 0.9845
Epoch 6/10
469/469 - 1s - loss: 0.0410 - accuracy: 0.9877
Epoch 7/10
469/469 - 1s - loss: 0.0331 - accuracy: 0.9900
Epoch 8/10
469/469 - 1s - loss: 0.0289 - accuracy: 0.9910
Epoch 9/10
469/469 - 1s - loss: 0.0235 - accuracy: 0.9928
Epoch 10/10
469/469 - 1s - loss: 0.0211 - accuracy: 0.9935
--Evaluate model--
313/313 - 0s - loss: 0.0617 - accuracy: 0.9818
Model Loss:    0.06
Model Accuracy: 98.2%

Process finished with exit code 0
```

Results: 99.35% on training data, 98.18% on test data

To run: python MNISTModel.py

For this part of the assignment, I have had to change the layers of the neural network model as well as the optimizer, batch_size and number of epochs, as follows:

1. Keras model (now 4 layers)
    a. Flatten
        i. Stayed the same as the starter code
    b. Dense

        i. tf.kears.layers.Dense(512, activation='relu')
        ii. a densely connected neural network layer, with 512 outputs with ReLU activation which helps rejecting very wrong guesses giving this layer better performance
    c. Droput
        i. tf.keras.layers.Dropout(0.2)
        ii. speeds up the training process, forces nodes within a layer to take more or less responsibility given a probability (0.2) in this case
    d. Dense
        i. tf.keras.layers.Dense(10, activation='softmax')
        ii. a densely connected neural network layer, with 10 outputs with softmax activation which helps being more favorable for output when more than 2 categories need to be considered.
2. Optimizer
    a. Changed the optimizer to adam – seems to be more commonly used and after testing other optimizers as well, I concluded that adam delivers the most optimized performance for this model
3. Batch size has been increased to 128 speeds up the time spent in each epoch without losing much accuracy.
4. Number of epochs has been increased to 10 allowing model to train on more training data before going for the testing data.

notMNISTModel.py

Originally, I have created this model similar to the MNISTModel.py one.

The model was as follows:

- tf.keras.layers.Flatten(input_shape=(28, 28)),
- tf.keras.layers.Dense(256, activation='relu'),
- tf.keras.layers.Dropout(0.1),
- tf.keras.layers.Dense(10, activation='softmax')
- Epochs set to 10

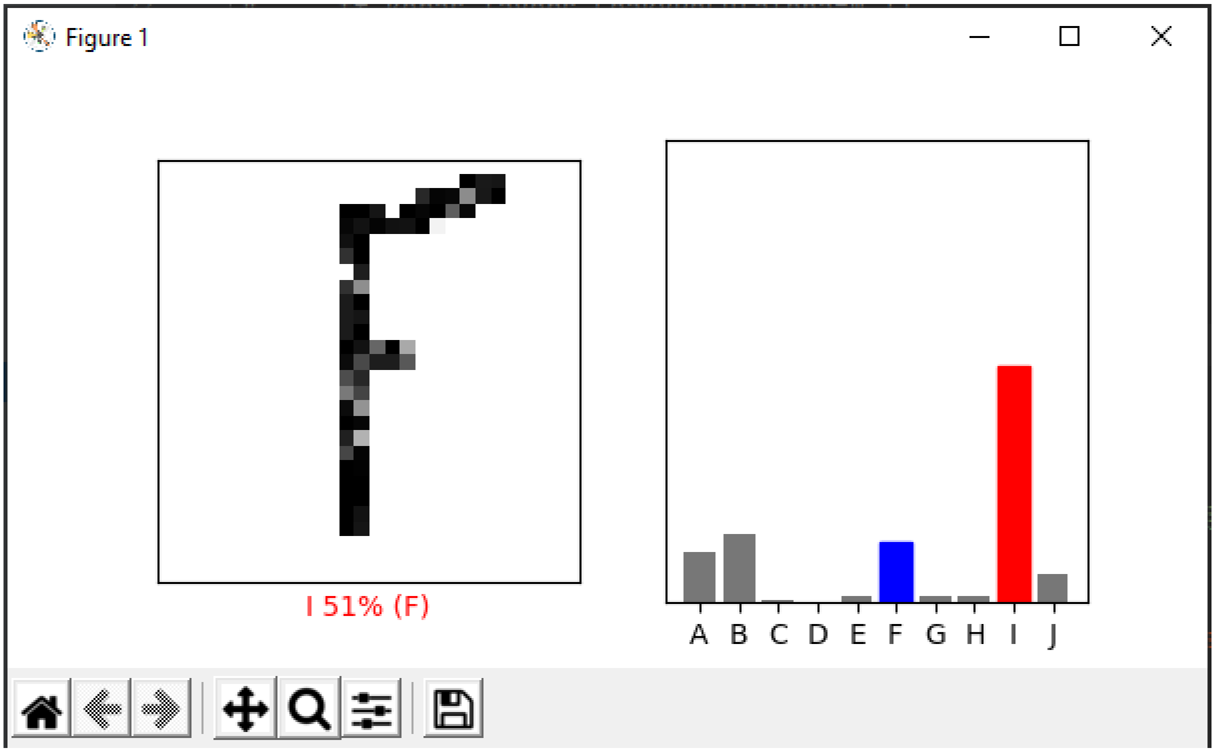
Model works well with capital letters from the predict_test.py application.

```
--Evaluate model--
313/313 - 0s - loss: 0.2183 - accuracy: 0.9387
Model Loss:    0.22
Model Accuracy: 93.9%
```

Modifications for predict files:

- Line to load your model
  - model = tf.keras.models.load_model(sys.argv[2]) (provided you are importing tensorflow as tf)
- Line to get array of percent confidence
  - prediction = model.predict(img)[0]
- Line to decide the index of the highest prediction
  - predicted_label = prediction.argmax(axis=-1)

However, the model identified **incorrectly** the following image of a letter of “F” as an “I” with 51% accuracy.

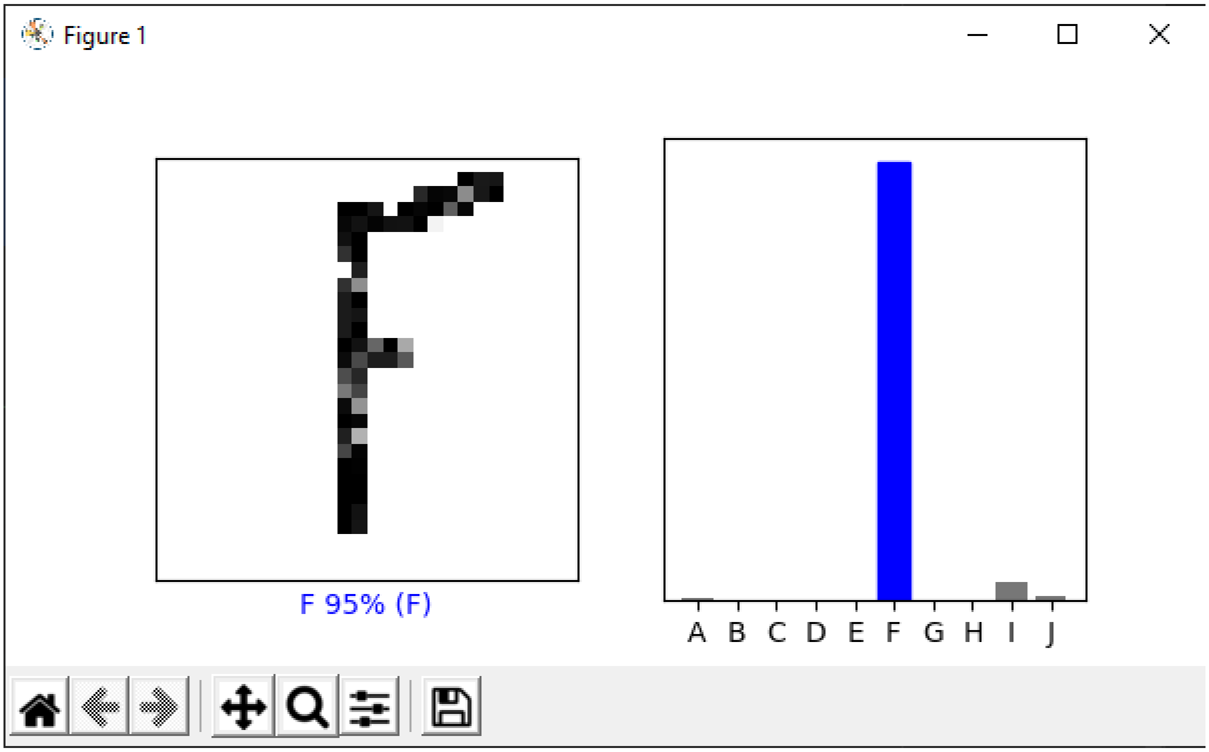


After change, the model correctly identified the letter as F, with 95% accuracy.

The model used was as follows:

```
--Evaluate model--
313/313 - 0s - loss: 0.2973 - accuracy: 0.9393
Model Loss:     0.30
Model Accuracy: 93.9%
```

- tf.keras.layers.Flatten(input_shape=(28, 28))
- tf.keras.layers.Dense(**512**, activation='relu')
- tf.keras.layers.Dropout(0.1)
- tf.keras.layers.Dense(10, activation='softmax')
- changed epochs **to equal 30**
- batch size stays the same as the original



To run: python notMNISTModel.py

CHDModel.py

BEFORE:

I implemented a function to split my heart.csv data randomly 90% into heart_train.csv and 10% into heart_test.csv.

I, then, followed this tutorial in order to load the csv components into the program so that they can be used with tensorflow (apply data normalization).

The original model was showing signs of **overfit**. The train datasets were evaluated at 97.8% whereas the test datasets evaluated at 65% accuracy.

The original model was as follows:

- tf.keras.layers.DenseFeatures(categoricalColumns + numericColumns)
- tf.keras.layers.Dense(256, activation='relu')
- tf.keras.layers.Dense(256, activation='relu')
- tf.keras.layers.Dense(1, activation='sigmoid')

The compilation of the model was done as follows:

- model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
- 20 epochs at a size of 128 each to fit model

Results:

```
Epoch 20/20
128/128 [==============================] - 0s 623us/step - loss: 0.1070 - accuracy: 0.9781
--Evaluate model--
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor, but we
Consider rewriting this model with the Functional API.
128/128 [==============================] - 0s 475us/step - loss: 1.4412 - accuracy: 0.6500
Model Loss:     1.44
Model Accuracy: 65.0%
```

AFTER:

After some tweaking of the model, I have managed to decrease its loss rate by about 3 times, reaching an accuracy of 78.2% on test data and 72.2% on train data.

I followed this tutorial.

The modified model is as follows:

- tf.keras.layers.DenseFeatures(categoricalColumns + numericColumns)
- **m.add(tf.keras.layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01)))**
- **tf.keras.layers.Dropout(0.5) tf.keras.layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01), activation='relu')**
- **tf.keras.layers.Dropout(0.5)**
- tf.keras.layers.Dense(1, activation='sigmoid')

The compilation of the model is done as follows:

- model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
- 20 epochs at a size **of 512 each** to fit model

Results:

```
Epoch 20/20
512/512 [==============================] - 0s 522us/step - loss: 0.5471 - accuracy: 0.7297
--Evaluate model--
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor, but w
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Callbacks method `on_test_batch_end` is slow compared to the batch time (b
512/512 [==============================] - 0s 446us/step - loss: 0.5116 - accuracy: 0.7395
Model Loss:     0.51
Model Accuracy: 73.9%
```

```
Epoch 20/20
512/512 [==============================] - 0s 559us/step - loss: 0.5630 - accuracy: 0.7219
--Evaluate model--
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor, but we
Consider rewriting this model with the Functional API.
512/512 [==============================] - 0s 464us/step - loss: 0.4455 - accuracy: 0.7824
Model Loss:     0.45
Model Accuracy: 78.2%
```

These changes made the training data accuracy to drop for their original 97.8% to 72.2%. This outcome did positively impact the test model accuracy, raising it from 65% to 78.2%.

To run: python CHDModel.py