Ian Pope 700717419 Big Data Analytics ICP4

Ian Pope 700717419 ICP4

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data: normalize images and one-hot encode labels
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build a Sequential model
model = Sequential()

# Flatten the input (28x28 images) into a vector of size 784
model.add(Flatten(input_shape=(28, 28)))

# Add a hidden layer with 256 neurons and Sigmoid activation
model.add(Dense(256, activation='sigmoid'))

# Add a hidden layer with 128 neurons and Sigmoid activation
model.add(Dense(128, activation='sigmoid'))

# Add a hidden layer with 128 neurons and Sigmoid activation
model.add(Dense(128, activation='sigmoid'))

# Add a hidden layer with 64 neurons and Sigmoid activation
model.add(Dense(64, activation='sigmoid'))

# Add a hidden layer with 32 neurons and Sigmoid activation
model.add(Dense(32, activation='sigmoid'))
```

```python
# Add the output layer with 10 neurons (one for each class) and softmax activation
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=32, validation_split=0.2)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`,
  super().__init__(**kwargs)
Epoch 1/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.5326 - loss: 1.4191 - val_accuracy: 0.9215 - val_loss: 0.3234
Epoch 2/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9267 - loss: 0.2909 - val_accuracy: 0.9426 - val_loss: 0.2083
Epoch 3/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9534 - loss: 0.1761 - val_accuracy: 0.9577 - val_loss: 0.1496
Epoch 4/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9664 - loss: 0.1248 - val_accuracy: 0.9641 - val_loss: 0.1301
Epoch 5/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9750 - loss: 0.0933 - val_accuracy: 0.9697 - val_loss: 0.1126
Epoch 6/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9794 - loss: 0.0748 - val_accuracy: 0.9701 - val_loss: 0.1082
Epoch 7/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9839 - loss: 0.0588 - val_accuracy: 0.9713 - val_loss: 0.1093
Epoch 8/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9864 - loss: 0.0497 - val_accuracy: 0.9711 - val_loss: 0.1162
Epoch 9/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9884 - loss: 0.0410 - val_accuracy: 0.9711 - val_loss: 0.1127
Epoch 10/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9909 - loss: 0.0336 - val_accuracy: 0.9778 - val_loss: 0.1368
Epoch 89/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9999 - loss: 2.9076e-04 - val_accuracy: 0.9772 - val_loss: 0.1686
Epoch 90/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9984 - loss: 0.0052 - val_accuracy: 0.9768 - val_loss: 0.1643
Epoch 91/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9989 - loss: 0.0031 - val_accuracy: 0.9773 - val_loss: 0.1602
Epoch 92/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9994 - loss: 0.0022 - val_accuracy: 0.9796 - val_loss: 0.1544
Epoch 93/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9999 - loss: 4.2554e-04 - val_accuracy: 0.9788 - val_loss: 0.1640
Epoch 94/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9996 - loss: 0.0025 - val_accuracy: 0.9758 - val_loss: 0.1722
Epoch 95/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9995 - loss: 0.0018 - val_accuracy: 0.9768 - val_loss: 0.1558
Epoch 96/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9992 - loss: 0.0028 - val_accuracy: 0.9768 - val_loss: 0.1627
Epoch 97/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9993 - loss: 0.0016 - val_accuracy: 0.9757 - val_loss: 0.1749
Epoch 98/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9995 - loss: 0.0018 - val_accuracy: 0.9777 - val_loss: 0.1642
Epoch 99/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9993 - loss: 0.0024 - val_accuracy: 0.9792 - val_loss: 0.1580
Epoch 100/100
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9998 - loss: 7.9323e-04 - val_accuracy: 0.9750 - val_loss: 0.1973
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9720 - loss: 0.2233
Test accuracy: 0.9745000004768372
```