Spring 2024 CS5720

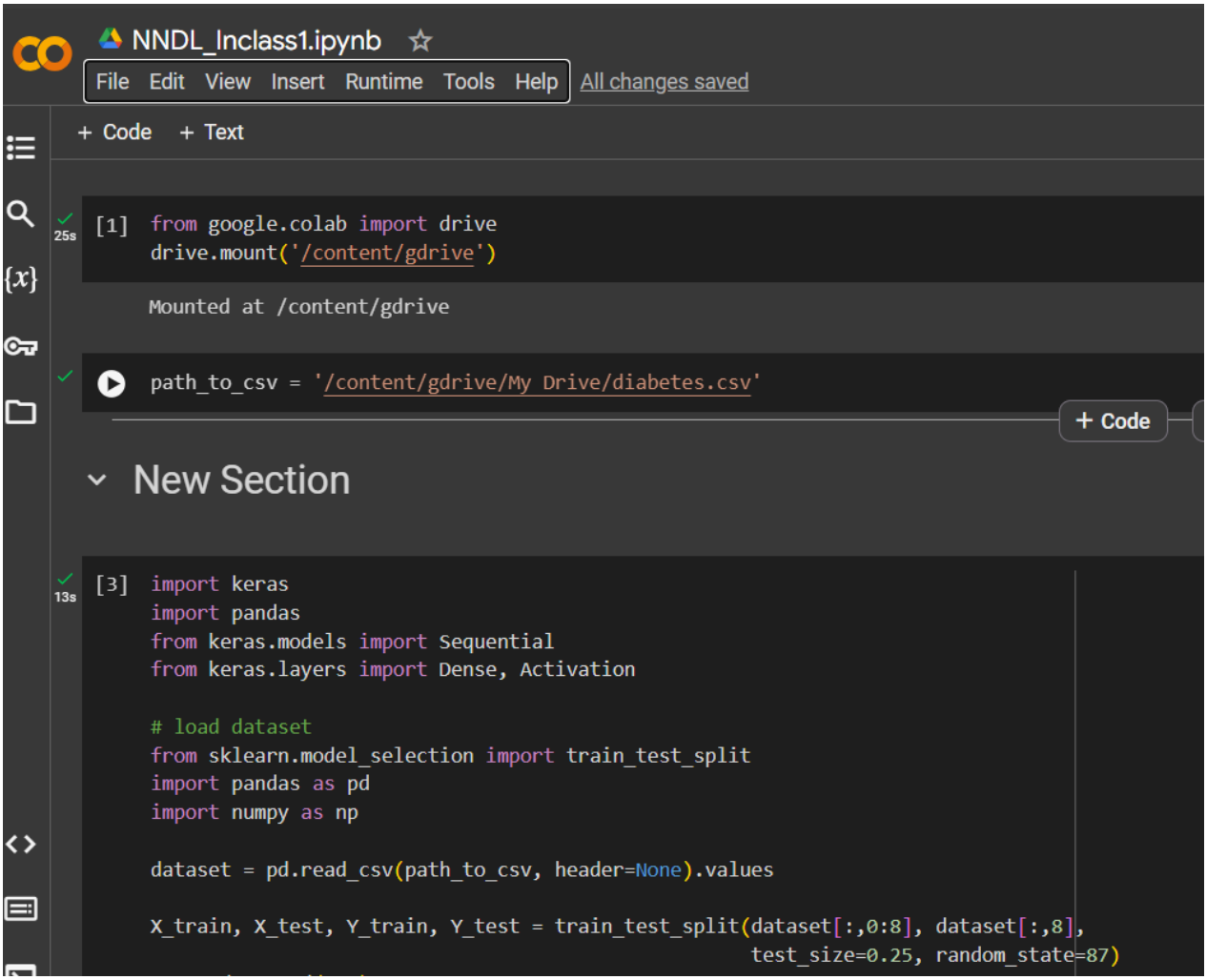
Neural Networks & Deep Learning - Assignment 6

Name: Sravya Reddy Pilli

Student Id: 700747154

Github link : https://github.com/09sravyareddy/NNDL-ICP6

In class programming: 1. Use the use case in the class: a. Add more Dense layers to the existing code and check how the accuracy changes. 2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model. 3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler() Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer


NNDL_Inclass1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[1] from google.colab import drive
drive.mount('/content/gdrive')
25s
Mounted at /content/gdrive
path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
+ Code
New Section
[3] import keras
import pandas
from keras.models import Sequential
from keras.layers import Dense, Activation
# load dataset
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
dataset = pd.read_csv(path_to_csv, header=None).values
X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
test_size=0.25, random_state=87)
13s

+ Code + Text

13s

```
X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, Y_test))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 19.6251 - acc: 0.6580
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 8.2646 - acc: 0.6615
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 2.0327 - acc: 0.5955
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 1.4780 - acc: 0.6580
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 1.3653 - acc: 0.6632
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 1.2616 - acc: 0.6771
Epoch 7/100
18/18 [==============================] - 0s 2ms/step - loss: 1.1901 - acc: 0.6806
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 1.1426 - acc: 0.6753
Epoch 9/100
18/18 [==============================] - 0s 2ms/step - loss: 1.0723 - acc: 0.6753
Epoch 10/100
```

```
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5655 - acc: 0.7188
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5507 - acc: 0.7257
Epoch 97/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5621 - acc: 0.7205
Epoch 98/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5533 - acc: 0.7153
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6016 - acc: 0.7014
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5942 - acc: 0.6892
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                180

 dense_1 (Dense)             (None, 1)                 21

=================================================================
Total params: 201 (804.00 Byte)
Trainable params: 201 (804.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.6415 - acc: 0.6771
[0.6415067911148071, 0.6770833134651184]
```

+ Code + Text

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
```

```
#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 6s 25ms/step - loss: 0.2889 - accuracy: 0.9113 - val_loss: 0.1801 - val_accuracy: 0.9398
Epoch 2/10
235/235 [==============================] - 11s 46ms/step - loss: 0.1006 - accuracy: 0.9684 - val_loss: 0.0900 - val_accuracy: 0.9718
Epoch 3/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0642 - accuracy: 0.9802 - val_loss: 0.0927 - val_accuracy: 0.9698
Epoch 4/10
235/235 [==============================] - 7s 32ms/step - loss: 0.0453 - accuracy: 0.9860 - val_loss: 0.0729 - val_accuracy: 0.9771
Epoch 5/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0320 - accuracy: 0.9902 - val_loss: 0.0918 - val_accuracy: 0.9720
Epoch 6/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0232 - accuracy: 0.9927 - val_loss: 0.0659 - val_accuracy: 0.9816
Epoch 7/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0168 - accuracy: 0.9947 - val_loss: 0.1069 - val_accuracy: 0.9720
Epoch 8/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0131 - accuracy: 0.9959 - val_loss: 0.0757 - val_accuracy: 0.9802
```

```
Epoch 9/10
235/235 [==============================] - 6s 27ms/step - loss: 0.0102 - accuracy: 0.9969 - val_loss: 0.0718 - va
Epoch 10/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0691 - va
```

```python
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split

# load dataset
path_to_csv = '/content/gdrive/MyDrive/diabetes.csv'
dataset = pd.read_csv(path_to_csv, header=None).values

# split dataset into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)

# define the model
np.random.seed(155)
my_second_nn = Sequential()
my_second_nn.add(Dense(20, input_dim=8, activation='relu'))
my_second_nn.add(Dense(20, input_dim=8,activation='relu'))
my_second_nn.add(Dense(20, input_dim=8,activation='relu'))
my_second_nn.add(Dense(1, activation='sigmoid'))
my_second_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# train the model
my_second_nn_fitted= my_second_nn.fit(X_train, Y_train, epochs=100,
                                      initial_epoch=0)


# evaluate the model on the test set
score = my_second_nn.evaluate(X_test, Y_test, batch_size=64)
print(my_second_nn.summary())
print("Test accuracy:", score[1])
```

```
Epoch 1/100
18/18 [==============================] - 2s 4ms/step - loss: 2.3255 - accuracy: 0.5208
Epoch 2/100
18/18 [==============================] - 0s 4ms/step - loss: 1.0137 - accuracy: 0.5729
Epoch 3/100
18/18 [==============================] - 0s 6ms/step - loss: 0.8428 - accuracy: 0.6424
Epoch 4/100
18/18 [==============================] - 0s 6ms/step - loss: 0.7392 - accuracy: 0.6458
Epoch 5/100
18/18 [==============================] - 0s 4ms/step - loss: 0.7389 - accuracy: 0.6493
Epoch 6/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6826 - accuracy: 0.6667
Epoch 7/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6552 - accuracy: 0.6719
Epoch 8/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6721 - accuracy: 0.6580
Epoch 9/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6329 - accuracy: 0.6649
Epoch 10/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6607 - accuracy: 0.6701
```

+ Code + Text

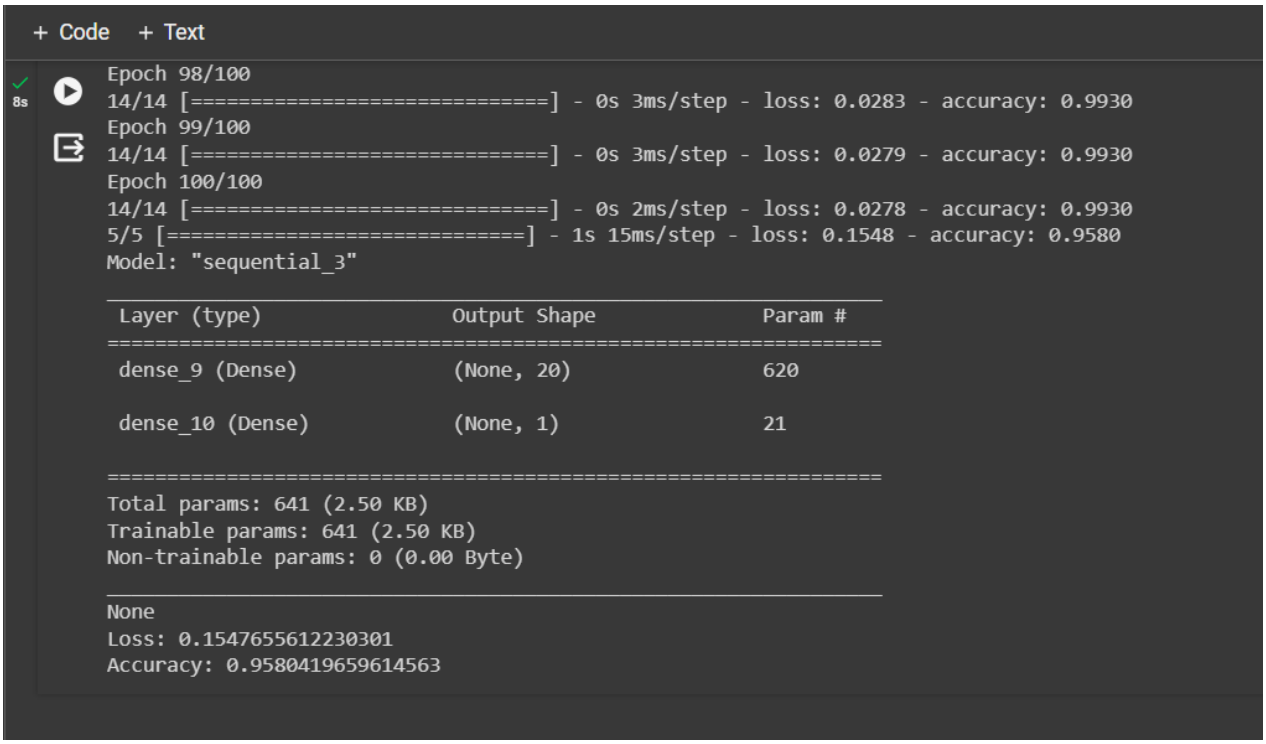
```
 dense_7 (Dense)             (None, 20)                420

 dense_8 (Dense)             (None, 1)                 21

=================================================================
Total params: 1041 (4.07 KB)
Trainable params: 1041 (4.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Test accuracy: 0.7083333134651184
```

[7]
```
path_to_csv = '/content/gdrive/MyDrive/breastcancer.csv'
```

[8]
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense

# Load dataset
data = load_breast_cancer()

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,
                                                    test_size=0.25, random_state=87)
```

```python
# Normalize data
sc = StandardScaler()
X_train_norm = sc.fit_transform(X_train)
X_test_norm = sc.transform(X_test)

# Create model
np.random.seed(155)
model = Sequential()
model.add(Dense(20, input_dim=30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
model.fit(X_train_norm, y_train, epochs=100, initial_epoch=0)

# Evaluate model on testing set
loss, accuracy = model.evaluate(X_test_norm, y_test)
print(model.summary())
print("Loss:", loss)
print("Accuracy:", accuracy)
```

```
Epoch 1/100
14/14 [==============================] - 1s 8ms/step - loss: 0.9442 - accuracy: 0.3662
Epoch 2/100
14/14 [==============================] - 0s 5ms/step - loss: 0.6279 - accuracy: 0.6925
Epoch 3/100
14/14 [==============================] - 0s 4ms/step - loss: 0.4447 - accuracy: 0.8286
Epoch 4/100
14/14 [==============================] - 0s 5ms/step - loss: 0.3392 - accuracy: 0.8897
Epoch 5/100
```



2. Use Image Classification on the hand written digits data set (mnist) 1. Plot the loss and accuracy for both training data and validation data using the history object in the source code. 2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image. 3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens. 4. Run the same code without scaling the images and check the performance?

NNDL_Inclass2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
```

+ Code + Text

```
#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 19s 71ms/step - loss: 0.2858 - accuracy: 0.9119 - val_loss: 0.1565 - val_accuracy: 0.9498
Epoch 2/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0990 - accuracy: 0.9695 - val_loss: 0.1182 - val_accuracy: 0.9627
Epoch 3/10
235/235 [==============================] - 8s 35ms/step - loss: 0.0627 - accuracy: 0.9803 - val_loss: 0.0810 - val_accuracy: 0.9735
Epoch 4/10
235/235 [==============================] - 8s 32ms/step - loss: 0.0437 - accuracy: 0.9861 - val_loss: 0.0635 - val_accuracy: 0.9798
Epoch 5/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0314 - accuracy: 0.9900 - val_loss: 0.0709 - val_accuracy: 0.9780
Epoch 6/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0229 - accuracy: 0.9926 - val_loss: 0.0829 - val_accuracy: 0.9761
Epoch 7/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0170 - accuracy: 0.9945 - val_loss: 0.0633 - val_accuracy: 0.9816
Epoch 8/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0132 - accuracy: 0.9958 - val_loss: 0.0748 - val_accuracy: 0.9792
Epoch 9/10
235/235 [==============================] - 7s 30ms/step - loss: 0.0099 - accuracy: 0.9969 - val_loss: 0.0668 - val_accuracy: 0.9821
```

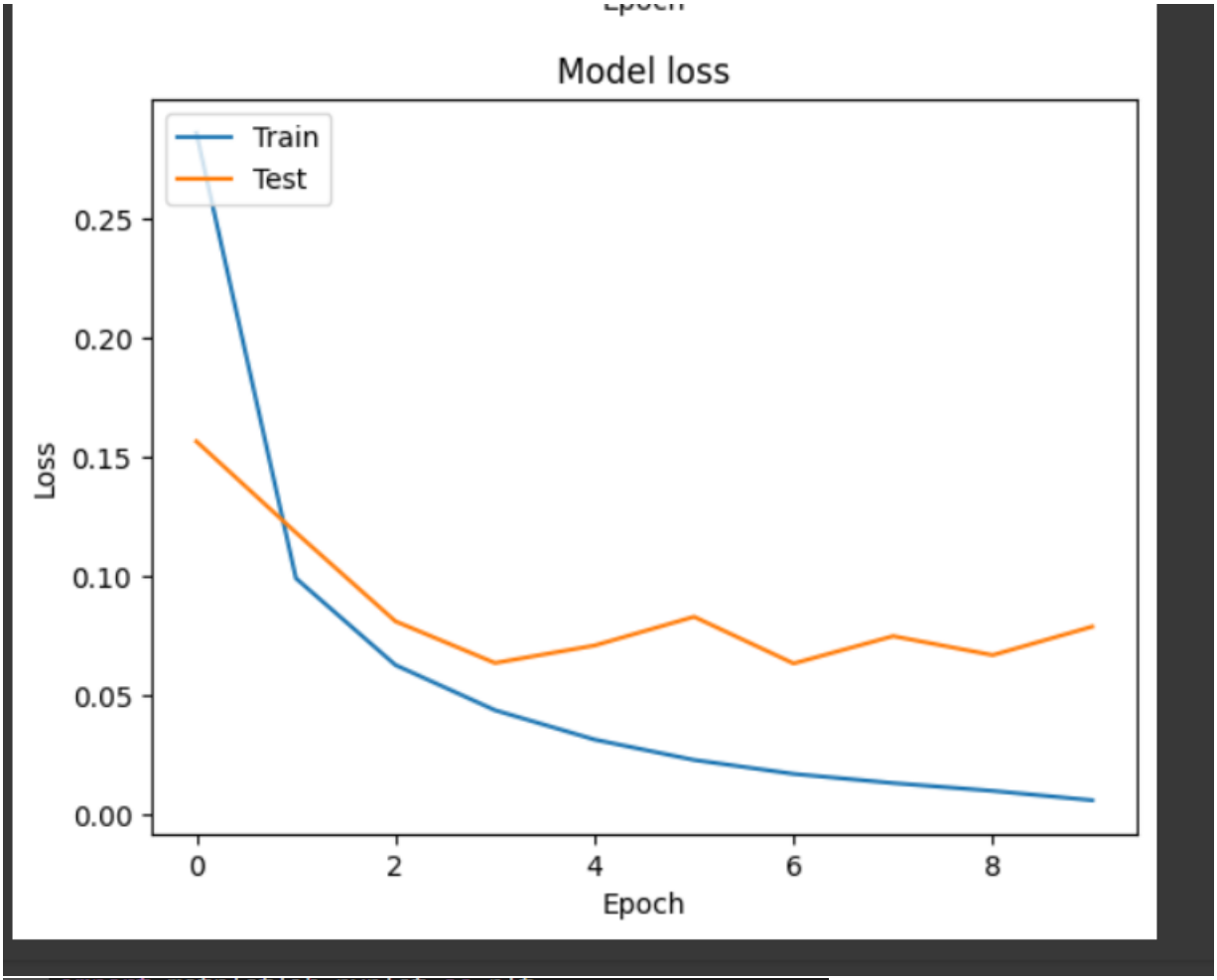
+ Code + Text

```
235/235 [==============================] - 6s 25ms/step - loss: 0.0132 - accuracy: 0.9958 -
Epoch 9/10
235/235 [==============================] - 7s 30ms/step - loss: 0.0099 - accuracy: 0.9969 -
Epoch 10/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0060 - accuracy: 0.9983 -
```

[2]

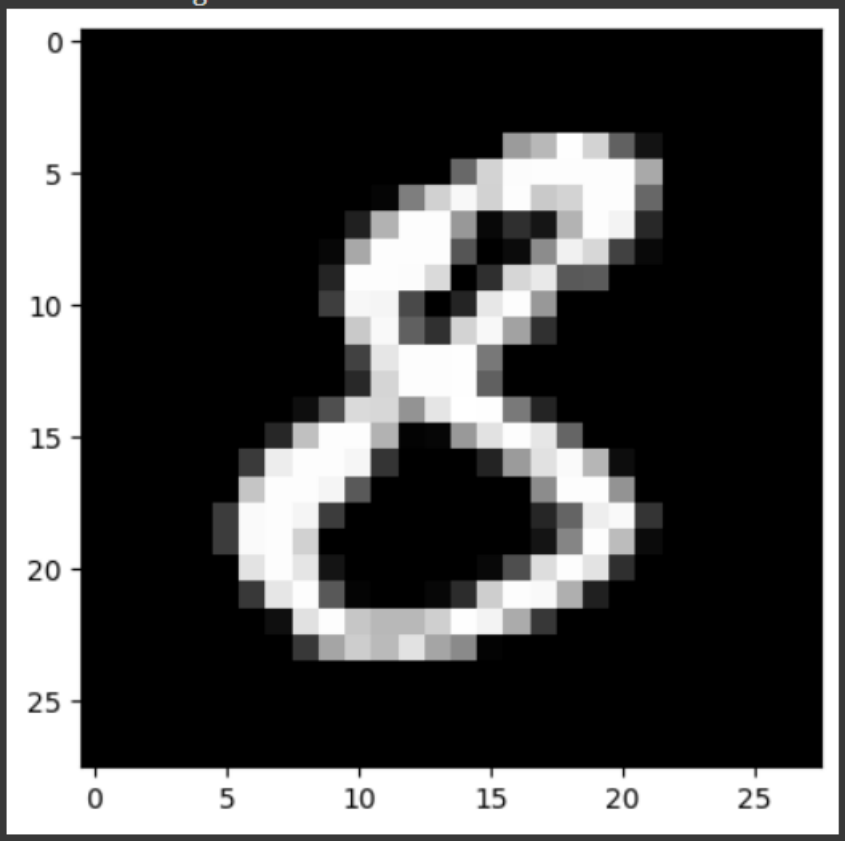
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```


Model accuracy
Train
Test
1.00
0.98
0.96
0.94
0.92
Accuracy
0
2
4
6
8




```
import matplotlib.pyplot as plt

# select a random image from test data
image_index = 1234
img = test_images[image_index]

# plot the image
plt.imshow(img, cmap='gray')

# reshape image to 1D vector
img = img.reshape((1, 784))

# normalize pixel values
img = img / 255.0

# predict class of image
result = model.predict(img)
print("Predicted digit:", np.argmax(result))
```

```
1/1 [==============================] - 0s 138ms/step
Predicted digit: 8
```



+ Code + Text

```python
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
```

+ Code + Text

```
#creating network
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model.add(Dense(256, activation='tanh'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 6s 22ms/step - loss: 0.3350 - accuracy: 0.8992 - val_loss: 0.2176 - val_accuracy: 0.9322
Epoch 2/10
235/235 [==============================] - 6s 27ms/step - loss: 0.1504 - accuracy: 0.9547 - val_loss: 0.1414 - val_accuracy: 0.9555
Epoch 3/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0997 - accuracy: 0.9704 - val_loss: 0.1293 - val_accuracy: 0.9575
Epoch 4/10
235/235 [==============================] - 6s 27ms/step - loss: 0.0736 - accuracy: 0.9785 - val_loss: 0.1185 - val_accuracy: 0.9603
Epoch 5/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0539 - accuracy: 0.9836 - val_loss: 0.0936 - val_accuracy: 0.9706
Epoch 6/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0423 - accuracy: 0.9872 - val_loss: 0.0747 - val_accuracy: 0.9752
Epoch 7/10
235/235 [==============================] - 6s 23ms/step - loss: 0.0312 - accuracy: 0.9902 - val_loss: 0.1228 - val_accuracy: 0.9614
Epoch 8/10
235/235 [==============================] - 5s 22ms/step - loss: 0.0241 - accuracy: 0.9929 - val_loss: 0.0809 - val_accuracy: 0.9742
Epoch 9/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0187 - accuracy: 0.9949 - val_loss: 0.0710 - val_accuracy: 0.9791
```

+ Code + Text

```
235/235 [==============================] - 5s 22ms/step - loss: 0.0126 - accuracy: 0.9963 - val_loss: 0.0875 - val_accuracy: 0.9755
```

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')

#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
```

```python
#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
test_loss, test_acc = model.evaluate(test_data, test_labels_one_hot, verbose=0)
print(f'Test loss: {test_loss:.3f}, Test accuracy: {test_acc:.3f}')
```

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 26ms/step - loss: 5.0982 - accuracy: 0.8769 - val_loss: 0.8423 - val_accuracy: 0.9043
Epoch 2/10
235/235 [==============================] - 7s 30ms/step - loss: 0.3880 - accuracy: 0.9446 - val_loss: 0.3313 - val_accuracy: 0.9507
Epoch 3/10
235/235 [==============================] - 6s 24ms/step - loss: 0.2474 - accuracy: 0.9591 - val_loss: 0.2967 - val_accuracy: 0.9513
Epoch 4/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1926 - accuracy: 0.9669 - val_loss: 0.2524 - val_accuracy: 0.9563
Epoch 5/10
235/235 [==============================] - 6s 25ms/step - loss: 0.1693 - accuracy: 0.9699 - val_loss: 0.4899 - val_accuracy: 0.9487
Epoch 6/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1486 - accuracy: 0.9758 - val_loss: 0.5468 - val_accuracy: 0.9502
Epoch 7/10
235/235 [==============================] - 6s 26ms/step - loss: 0.1354 - accuracy: 0.9777 - val_loss: 0.3012 - val_accuracy: 0.9676
Epoch 8/10
235/235 [==============================] - 7s 29ms/step - loss: 0.1280 - accuracy: 0.9809 - val_loss: 0.3431 - val_accuracy: 0.9623
Epoch 9/10
```


```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 26ms/step - loss: 5.0982 - accuracy: 0.8769 - val_loss: 0.8423 - val_accuracy: 0.9043
Epoch 2/10
235/235 [==============================] - 7s 30ms/step - loss: 0.3880 - accuracy: 0.9446 - val_loss: 0.3313 - val_accuracy: 0.9507
Epoch 3/10
235/235 [==============================] - 6s 24ms/step - loss: 0.2474 - accuracy: 0.9591 - val_loss: 0.2967 - val_accuracy: 0.9513
Epoch 4/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1926 - accuracy: 0.9669 - val_loss: 0.2524 - val_accuracy: 0.9563
Epoch 5/10
235/235 [==============================] - 6s 25ms/step - loss: 0.1693 - accuracy: 0.9699 - val_loss: 0.4899 - val_accuracy: 0.9487
Epoch 6/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1486 - accuracy: 0.9758 - val_loss: 0.5468 - val_accuracy: 0.9502
Epoch 7/10
235/235 [==============================] - 6s 26ms/step - loss: 0.1354 - accuracy: 0.9777 - val_loss: 0.3012 - val_accuracy: 0.9676
Epoch 8/10
235/235 [==============================] - 7s 29ms/step - loss: 0.1280 - accuracy: 0.9809 - val_loss: 0.3431 - val_accuracy: 0.9623
Epoch 9/10
```


```
235/235 [==============================] - 6s 26ms/step - loss: 0.1219 - accuracy: 0.9821 - val_loss: 0.4305 - val_accuracy: 0.9605
Epoch 10/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1165 - accuracy: 0.9843 - val_loss: 0.3652 - val_accuracy: 0.9681
Test loss: 0.365, Test accuracy: 0.968
```