# CS 5720 Neural Networks & Deep Learning


## Madhavi Mamidala

## 700744319


Video Recording :
https://drive.google.com/file/d/119qzpRjlxdTJg0Ztzlt5eLpa95H7EQyM/view?usp=sharing

GitHub Link: https://github.com/MadhaviMamidala/CS5720-NN-Assignment-2

1.

ICP_Basics in Keras

```
[279] import keras
      import pandas
      from keras.models import Sequential
      from keras.layers.core import Dense, Activation

      # load dataset
      from sklearn.model_selection import train_test_split
      import pandas as pd
      import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

2.

```
path = '/content/drive/MyDrive/Colab Notebooks/Data/diabetes.csv'

dataset = pd.read_csv(path, 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)
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 80/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5917 - acc: 0.7066
Epoch 81/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6112 - acc: 0.6823
Epoch 82/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6007 - acc: 0.6962
Epoch 83/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5510 - acc: 0.7135
Epoch 84/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5770 - acc: 0.7014
Epoch 85/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5624 - acc: 0.7170
Epoch 86/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5445 - acc: 0.7205
```

```
Epoch 92/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5570 - acc: 0.7135
Epoch 93/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5488 - acc: 0.7431
Epoch 94/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5456 - acc: 0.7205
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5429 - acc: 0.7309
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5545 - acc: 0.7309
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5644 - acc: 0.7222
Epoch 98/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5438 - acc: 0.7240
Epoch 99/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5423 - acc: 0.7483
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5439 - acc: 0.7153
Model: "sequential_66"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_149 (Dense)           (None, 20)                180

 dense_150 (Dense)           (None, 1)                 21

=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 4ms/step - loss: 0.7264 - acc: 0.6354
[0.7263796329498291, 0.6354166865348816]
```

```python
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(20, input_dim=4, activation='relu')) # hidden layer
my_first_nn.add(Dense(20, input_dim=2, 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 82/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4794 - acc: 0.7535
Epoch 83/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4950 - acc: 0.7535
Epoch 84/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4857 - acc: 0.7691
Epoch 85/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4750 - acc: 0.7847
Epoch 86/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4892 - acc: 0.7535
Epoch 87/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4710 - acc: 0.7604
Epoch 88/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4801 - acc: 0.7604
Epoch 89/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4873 - acc: 0.7674
Epoch 90/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5088 - acc: 0.7604
Epoch 91/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4871 - acc: 0.7708
Epoch 92/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4806 - acc: 0.7674
Epoch 93/100
18/18 [==============================] - 0s 3ms/step - loss: 0.4675 - acc: 0.7569
```

```
Epoch 91/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4871 - acc: 0.7708
Epoch 92/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4806 - acc: 0.7674
Epoch 93/100
18/18 [==============================] - 0s 3ms/step - loss: 0.4675 - acc: 0.7569
Epoch 94/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4651 - acc: 0.7812
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4712 - acc: 0.7760
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4880 - acc: 0.7535
Epoch 97/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5049 - acc: 0.7396
Epoch 98/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4684 - acc: 0.7778
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4724 - acc: 0.7708
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4614 - acc: 0.7760
Model: "sequential_67"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_151 (Dense)           (None, 20)                180

 dense_152 (Dense)           (None, 20)                420

 dense_153 (Dense)           (None, 20)                420

 dense_154 (Dense)           (None, 1)                 21

=================================================================
Total params: 1,041
Trainable params: 1,041
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5917 - acc: 0.6979
[0.5917186141014099, 0.6979166865348816]
```

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

```
path = '/content/drive/MyDrive/Colab Notebooks/Data/breastcancer.csv'

dataset = pd.read_csv(path, header=None).values
dataset[0]
```

```
array(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean',
       'concavity_mean', 'concave points_mean', 'symmetry_mean',
       'fractal_dimension_mean', 'radius_se', 'texture_se',
       'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', nan], dtype=object)
```

```python
train_data = dataset[1:,2:32]
test_data = dataset[1:,1]


test_data[(test_data == 'M')]=1
test_data[(test_data == 'B')]=0
train_data  = train_data.astype(float)
train_data[0]
```

```
array([1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
       3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
       8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
       3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
       1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01])
```

+ Code + Text

[285]
```python
X_train, X_test, Y_train, Y_test = train_test_split(train_data, test_data, test_size=0.25, random_state=87)

X_train = np.array([np.array(val) for val in X_train])  # reconstruct
Y_train = np.array([np.array(val) for val in Y_train])  # reconstruct
X_test = np.array([np.array(val) for val in X_test])  # reconstruct
Y_test = np.array([np.array(val) for val in Y_test])  # reconstruct

np.random.seed(155)

my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
#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 94/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3083 - acc: 0.8826
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2260 - acc: 0.9178
Epoch 96/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2248 - acc: 0.9225
Epoch 97/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2438 - acc: 0.9085
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2410 - acc: 0.9155
Epoch 99/100
14/14 [==============================] - 0s 4ms/step - loss: 0.2815 - acc: 0.9061
Epoch 100/100
14/14 [==============================] - 0s 3ms/step - loss: 0.3041 - acc: 0.8944
Model: "sequential_68"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_155 (Dense)           (None, 20)                620

 dense_156 (Dense)           (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 6ms/step - loss: 0.4599 - acc: 0.8811
[0.45993831753730774, 0.881118893623352]
```

```
[286] from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      scaled_train = sc.fit_transform(train_data)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(scaled_train, test_data, test_size=0.25, random_state=87)

X_train = np.array([np.array(val) for val in X_train])  # reconstruct
Y_train = np.array([np.array(val) for val in Y_train])  # reconstruct
X_test = np.array([np.array(val) for val in X_test])  # reconstruct
Y_test = np.array([np.array(val) for val in Y_test])  # reconstruct

np.random.seed(155)

my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
#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'])
```

```
[288] np.random.seed(155)
      my_nn = Sequential() # create model
      my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
      my_nn.add(Dense(1, activation='sigmoid')) # output layer
      my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
      my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                               initial_epoch=0)
      print(my_nn.summary())
      print(my_nn.evaluate(X_test, Y_test))
```

+ Code + Text

```
Epoch 93/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0255 - acc: 0.9930
Epoch 94/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0255 - acc: 0.9930
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0250 - acc: 0.9930
Epoch 96/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0248 - acc: 0.9930
Epoch 97/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0245 - acc: 0.9930
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0242 - acc: 0.9930
Epoch 99/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0239 - acc: 0.9930
Epoch 100/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0237 - acc: 0.9930
Model: "sequential_70"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_159 (Dense)           (None, 20)                620

 dense_160 (Dense)           (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: 0.1759 - acc: 0.9650
[0.17592251300811768, 0.9650349617004395]
```

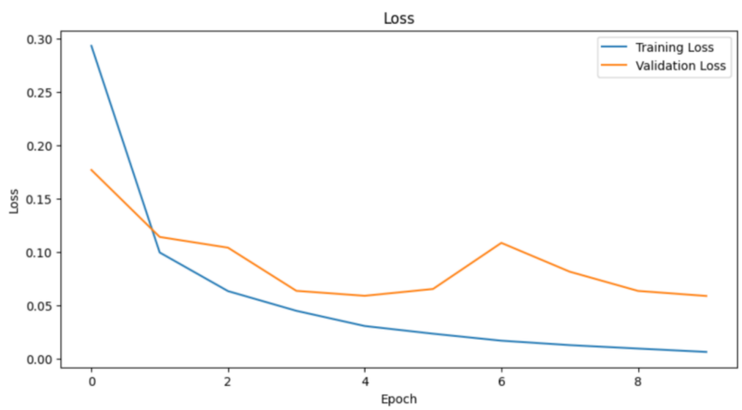
```python
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(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)

#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))
history.history['accuracy']
```

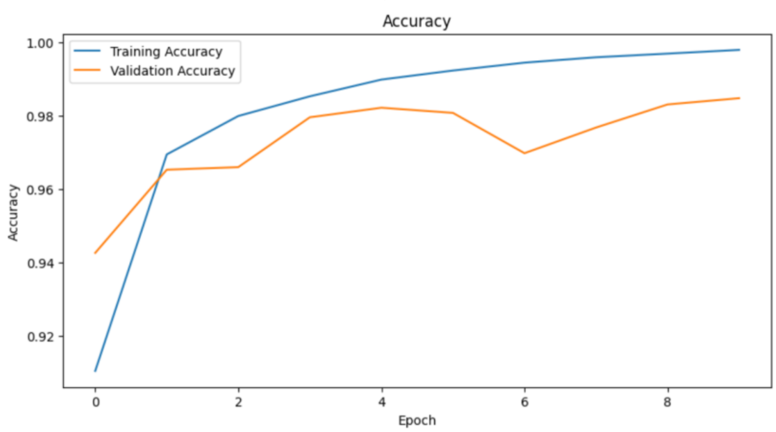
```
[289]
Epoch 3/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0636 - accuracy: 0.9800 - val_loss: 0.1042 - val_accuracy: 0.9660
Epoch 4/10
235/235 [==============================] - 11s 45ms/step - loss: 0.0451 - accuracy: 0.9853 - val_loss: 0.0638 - val_accuracy: 0.9796
Epoch 5/10
235/235 [==============================] - 9s 40ms/step - loss: 0.0309 - accuracy: 0.9899 - val_loss: 0.0592 - val_accuracy: 0.9822
Epoch 6/10
235/235 [==============================] - 9s 38ms/step - loss: 0.0237 - accuracy: 0.9923 - val_loss: 0.0656 - val_accuracy: 0.9808
Epoch 7/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0172 - accuracy: 0.9945 - val_loss: 0.1087 - val_accuracy: 0.9698
Epoch 8/10
235/235 [==============================] - 12s 49ms/step - loss: 0.0130 - accuracy: 0.9960 - val_loss: 0.0818 - val_accuracy: 0.9768
Epoch 9/10
235/235 [==============================] - 8s 35ms/step - loss: 0.0098 - accuracy: 0.9970 - val_loss: 0.0638 - val_accuracy: 0.9831
Epoch 10/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0590 - val_accuracy: 0.9848
[0.9103999733924866,
 0.9694666862487793,
 0.9799500107765198,
 0.9853166937828064,
 0.989883303642273,
 0.9923499822616577,
 0.9944999814033508,
 0.9959666728973389,
 0.9969666600227356,
 0.9979833364486694]
```

Use Image Classification on the hand written digits data set (mnist)

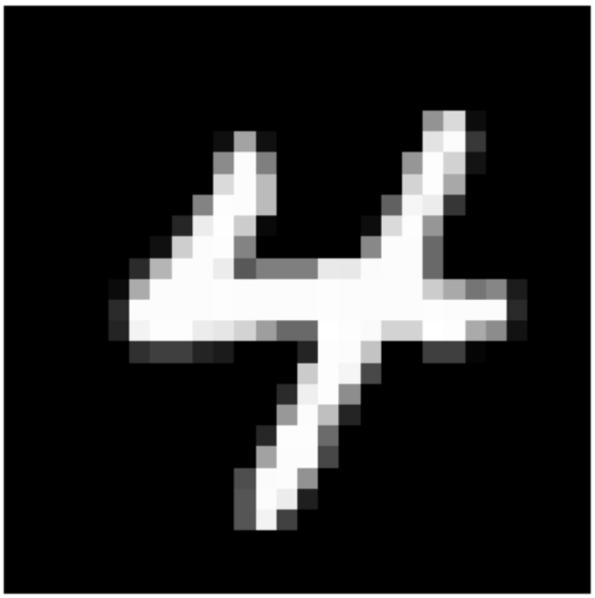
1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

```
import matplotlib.pyplot as plt

# Accessing loss and accuracy values from the history object
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Plotting the loss
plt.figure(figsize=(10, 5))
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plotting the accuracy
plt.figure(figsize=(10, 5))
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```python
# Select a random image from the test data
image_index = np.random.randint(0, len(test_data))
image = test_images[image_index]

# Plot the image
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.show()
```



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

```
Epoch 1/10
235/235 [==============================] - 17s 68ms/step - loss: 0.3911 - accuracy: 0.8782 - val_loss: 0.1407 - val_accuracy: 0.9542
Epoch 2/10
235/235 [==============================] - 10s 44ms/step - loss: 0.1222 - accuracy: 0.9626 - val_loss: 0.0946 - val_accuracy: 0.9700
Epoch 3/10
235/235 [==============================] - 13s 54ms/step - loss: 0.0776 - accuracy: 0.9756 - val_loss: 0.0964 - val_accuracy: 0.9699
Epoch 4/10
235/235 [==============================] - 13s 55ms/step - loss: 0.0536 - accuracy: 0.9833 - val_loss: 0.1014 - val_accuracy: 0.9670
Epoch 5/10
235/235 [==============================] - 11s 46ms/step - loss: 0.0380 - accuracy: 0.9877 - val_loss: 0.0817 - val_accuracy: 0.9724
Epoch 6/10
235/235 [==============================] - 17s 71ms/step - loss: 0.0272 - accuracy: 0.9913 - val_loss: 0.0720 - val_accuracy: 0.9780
Epoch 7/10
235/235 [==============================] - 9s 38ms/step - loss: 0.0214 - accuracy: 0.9930 - val_loss: 0.0635 - val_accuracy: 0.9812
Epoch 8/10
235/235 [==============================] - 13s 54ms/step - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.1050 - val_accuracy: 0.9705
Epoch 9/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0129 - accuracy: 0.9957 - val_loss: 0.0757 - val_accuracy: 0.9791
Epoch 10/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0092 - accuracy: 0.9973 - val_loss: 0.0778 - val_accuracy: 0.9812
```

```
[293] history.history['accuracy']
```

```
[0.8782166838645935,
 0.9625833630561829,
 0.9756166934967041,
 0.9833166599273682,
 0.9877499938011169,
 0.991349995136261,
 0.9930333495140076,
 0.994783341884613,
 0.9956833124160767,
 0.9972500205039978]
```

```
(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)

#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))
```

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 26ms/step - loss: 5.0882 - accuracy: 0.8827 - val_loss: 0.5847 - val_accuracy: 0.9360
Epoch 2/10
235/235 [==============================] - 6s 25ms/step - loss: 0.4122 - accuracy: 0.9484 - val_loss: 0.3153 - val_accuracy: 0.9487
Epoch 3/10
235/235 [==============================] - 8s 32ms/step - loss: 0.2498 - accuracy: 0.9589 - val_loss: 0.3796 - val_accuracy: 0.9494
Epoch 4/10
235/235 [==============================] - 5s 23ms/step - loss: 0.1804 - accuracy: 0.9684 - val_loss: 0.3223 - val_accuracy: 0.9550
Epoch 5/10
235/235 [==============================] - 8s 33ms/step - loss: 0.1633 - accuracy: 0.9720 - val_loss: 0.3169 - val_accuracy: 0.9589
Epoch 6/10
235/235 [==============================] - 6s 25ms/step - loss: 0.1419 - accuracy: 0.9763 - val_loss: 0.3005 - val_accuracy: 0.9664
Epoch 7/10
235/235 [==============================] - 6s 24ms/step - loss: 0.1377 - accuracy: 0.9793 - val_loss: 0.3342 - val_accuracy: 0.9587
Epoch 8/10
235/235 [==============================] - 8s 33ms/step - loss: 0.1238 - accuracy: 0.9806 - val_loss: 0.3285 - val_accuracy: 0.9737
Epoch 9/10
235/235 [==============================] - 5s 23ms/step - loss: 0.1232 - accuracy: 0.9836 - val_loss: 0.3089 - val_accuracy: 0.9712
Epoch 10/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1143 - accuracy: 0.9841 - val_loss: 0.4161 - val_accuracy: 0.9643
```

```
[295] history.history['accuracy']
```

```
[0.8827000260353088,
 0.9484000205993652,
 0.9588833451271057,
 0.9684333205223083,
 0.9720333218574524,
 0.9763000011444092,
 0.9793000221252441,
 0.9806166887283325,
 0.9836000204086304,
 0.9840999841690063]
```

ICP_Basics in Keras

```
[279] import keras
      import pandas
      from keras.models import Sequential
      from keras.layers.core import Dense, Activation

      # load dataset
      from sklearn.model_selection import train_test_split
      import pandas as pd
      import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
path = '/content/drive/MyDrive/Colab Notebooks/Data/diabetes.csv'

dataset = pd.read_csv(path, 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)
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 80/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5917 - acc: 0.7066
Epoch 81/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6112 - acc: 0.6823
Epoch 82/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6007 - acc: 0.6962
Epoch 83/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5510 - acc: 0.7135
Epoch 84/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5770 - acc: 0.7014
Epoch 85/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5624 - acc: 0.7170
Epoch 86/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5445 - acc: 0.7205
```

```
Epoch 92/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5570 - acc: 0.7135
Epoch 93/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5488 - acc: 0.7431
Epoch 94/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5456 - acc: 0.7205
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5429 - acc: 0.7309
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5545 - acc: 0.7309
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5644 - acc: 0.7222
Epoch 98/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5438 - acc: 0.7240
Epoch 99/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5423 - acc: 0.7483
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5439 - acc: 0.7153
Model: "sequential_66"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_149 (Dense)           (None, 20)                180

 dense_150 (Dense)           (None, 1)                 21

=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 4ms/step - loss: 0.7264 - acc: 0.6354
[0.7263796329498291, 0.6354166865348816]
```

```python
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(20, input_dim=4, activation='relu')) # hidden layer
my_first_nn.add(Dense(20, input_dim=2, 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 82/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4794 - acc: 0.7535
Epoch 83/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4950 - acc: 0.7535
Epoch 84/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4857 - acc: 0.7691
Epoch 85/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4750 - acc: 0.7847
Epoch 86/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4892 - acc: 0.7535
Epoch 87/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4710 - acc: 0.7604
Epoch 88/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4801 - acc: 0.7604
Epoch 89/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4873 - acc: 0.7674
Epoch 90/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5088 - acc: 0.7604
Epoch 91/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4871 - acc: 0.7708
Epoch 92/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4806 - acc: 0.7674
Epoch 93/100
18/18 [==============================] - 0s 3ms/step - loss: 0.4675 - acc: 0.7569
```


Epoch 91/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4871 - acc: 0.7708
Epoch 92/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4806 - acc: 0.7674
Epoch 93/100
18/18 [==============================] - 0s 3ms/step - loss: 0.4675 - acc: 0.7569


```
Epoch 94/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4651 - acc: 0.7812
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4712 - acc: 0.7760
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4880 - acc: 0.7535
Epoch 97/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5049 - acc: 0.7396
Epoch 98/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4684 - acc: 0.7778
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4724 - acc: 0.7708
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.4614 - acc: 0.7760
Model: "sequential_67"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_151 (Dense)           (None, 20)                180

 dense_152 (Dense)           (None, 20)                420

 dense_153 (Dense)           (None, 20)                420

 dense_154 (Dense)           (None, 1)                 21

=================================================================
Total params: 1,041
Trainable params: 1,041
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5917 - acc: 0.6979
[0.5917186141014099, 0.6979166865348816]
```

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

```
path = '/content/drive/MyDrive/Colab Notebooks/Data/breastcancer.csv'

dataset = pd.read_csv(path, header=None).values
dataset[0]
```

```
array(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean',
       'concavity_mean', 'concave points_mean', 'symmetry_mean',
       'fractal_dimension_mean', 'radius_se', 'texture_se',
       'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', nan], dtype=object)
```

```
train_data = dataset[1:,2:32]
test_data = dataset[1:,1]


test_data[(test_data == 'M')]=1
test_data[(test_data == 'B')]=0
train_data  = train_data.astype(float)
train_data[0]
```

```
array([1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
       3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
       8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
       3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
       1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01])
```

+ Code + Text

```
[285] X_train, X_test, Y_train, Y_test = train_test_split(train_data, test_data, test_size=0.25, random_state=87)

X_train = np.array([np.array(val) for val in X_train])  # reconstruct
Y_train = np.array([np.array(val) for val in Y_train])  # reconstruct
X_test = np.array([np.array(val) for val in X_test])  # reconstruct
Y_test = np.array([np.array(val) for val in Y_test])  # reconstruct

np.random.seed(155)

my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
#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 94/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3083 - acc: 0.8826
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2260 - acc: 0.9178
Epoch 96/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2248 - acc: 0.9225
Epoch 97/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2438 - acc: 0.9085
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2410 - acc: 0.9155
Epoch 99/100
14/14 [==============================] - 0s 4ms/step - loss: 0.2815 - acc: 0.9061
Epoch 100/100
14/14 [==============================] - 0s 3ms/step - loss: 0.3041 - acc: 0.8944
Model: "sequential_68"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_155 (Dense)           (None, 20)                620

 dense_156 (Dense)           (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 6ms/step - loss: 0.4599 - acc: 0.8811
[0.45993831753730774, 0.881118893623352]
```

[286]
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
scaled_train = sc.fit_transform(train_data)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(scaled_train, test_data, test_size=0.25, random_state=87)

X_train = np.array([np.array(val) for val in X_train])  # reconstruct
Y_train = np.array([np.array(val) for val in Y_train])  # reconstruct
X_test = np.array([np.array(val) for val in X_test])  # reconstruct
Y_test = np.array([np.array(val) for val in Y_test])  # reconstruct

np.random.seed(155)

my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
#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'])
```

[288]
```
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

\+ Code + Text

```
Epoch 93/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0255 - acc: 0.9930
Epoch 94/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0255 - acc: 0.9930
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0250 - acc: 0.9930
Epoch 96/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0248 - acc: 0.9930
Epoch 97/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0245 - acc: 0.9930
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0242 - acc: 0.9930
Epoch 99/100
14/14 [==============================] - 0s 3ms/step - loss: 0.0239 - acc: 0.9930
Epoch 100/100
14/14 [==============================] - 0s 2ms/step - loss: 0.0237 - acc: 0.9930
Model: "sequential_70"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_159 (Dense)           (None, 20)                620

 dense_160 (Dense)           (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: 0.1759 - acc: 0.9650
[0.17592251300811768, 0.9650349617004395]
```

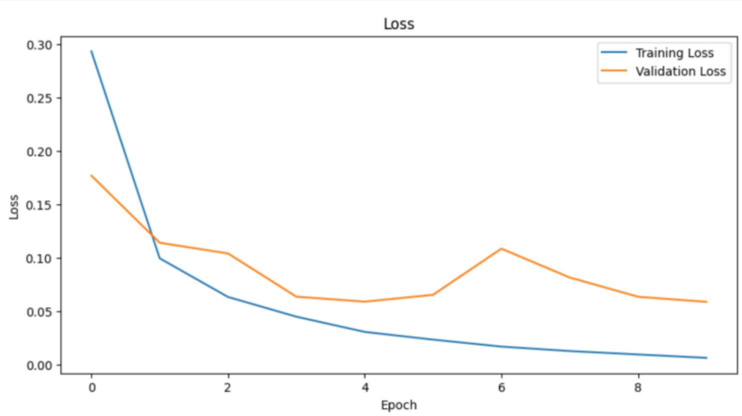
```python
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(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)

#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))
history.history['accuracy']
```

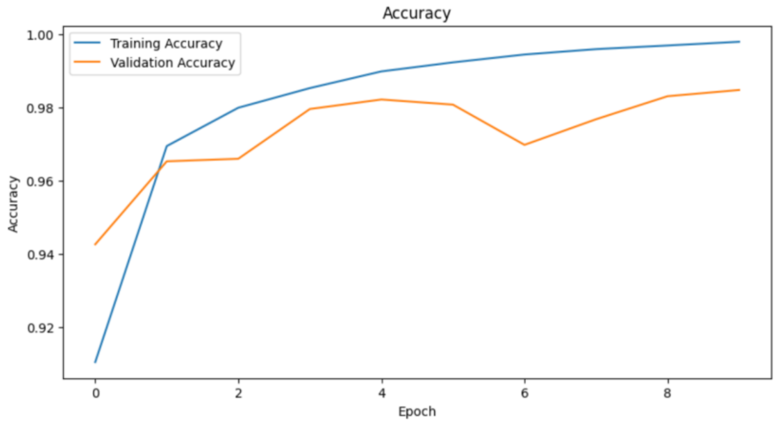
```
[289] Epoch 3/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0636 - accuracy: 0.9800 - val_loss: 0.1042 - val_accuracy: 0.9660
Epoch 4/10
235/235 [==============================] - 11s 45ms/step - loss: 0.0451 - accuracy: 0.9853 - val_loss: 0.0638 - val_accuracy: 0.9796
Epoch 5/10
235/235 [==============================] - 9s 40ms/step - loss: 0.0309 - accuracy: 0.9899 - val_loss: 0.0592 - val_accuracy: 0.9822
Epoch 6/10
235/235 [==============================] - 9s 38ms/step - loss: 0.0237 - accuracy: 0.9923 - val_loss: 0.0656 - val_accuracy: 0.9808
Epoch 7/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0172 - accuracy: 0.9945 - val_loss: 0.1087 - val_accuracy: 0.9698
Epoch 8/10
235/235 [==============================] - 12s 49ms/step - loss: 0.0130 - accuracy: 0.9960 - val_loss: 0.0818 - val_accuracy: 0.9768
Epoch 9/10
235/235 [==============================] - 8s 35ms/step - loss: 0.0098 - accuracy: 0.9970 - val_loss: 0.0638 - val_accuracy: 0.9831
Epoch 10/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0590 - val_accuracy: 0.9848
[0.9103999733924866,
 0.9694666862487793,
 0.9799500107765198,
 0.9853166937828064,
 0.989883303642273,
 0.9923499822616577,
 0.9944999814033508,
 0.9959666728973389,
 0.9969666600227356,
 0.9979833364486694]
```

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.

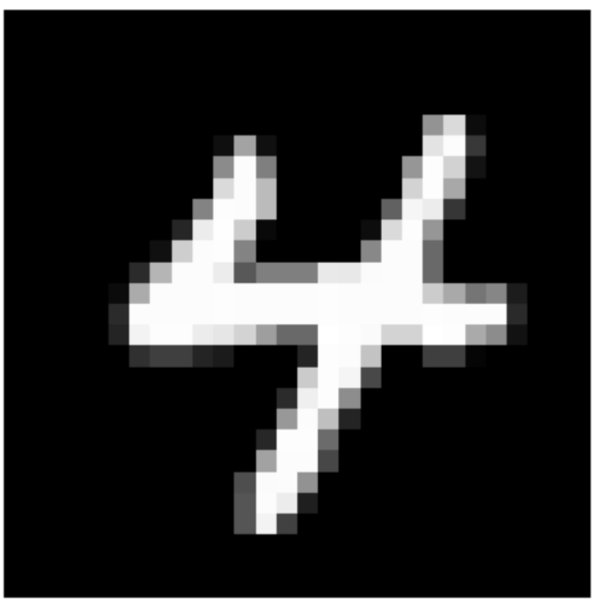
```python
import matplotlib.pyplot as plt

# Accessing loss and accuracy values from the history object
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

# Plotting the loss
plt.figure(figsize=(10, 5))
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plotting the accuracy
plt.figure(figsize=(10, 5))
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```






0s

```
# Select a random image from the test data
image_index = np.random.randint(0, len(test_data))
image = test_images[image_index]

# Plot the image
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.show()
```



2m

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

```
Epoch 1/10
235/235 [==============================] - 17s 68ms/step - loss: 0.3911 - accuracy: 0.8782 - val_loss: 0.1407 - val_accuracy: 0.9542
Epoch 2/10
235/235 [==============================] - 10s 44ms/step - loss: 0.1222 - accuracy: 0.9626 - val_loss: 0.0946 - val_accuracy: 0.9700
Epoch 3/10
235/235 [==============================] - 13s 54ms/step - loss: 0.0776 - accuracy: 0.9756 - val_loss: 0.0964 - val_accuracy: 0.9699
Epoch 4/10
235/235 [==============================] - 13s 55ms/step - loss: 0.0536 - accuracy: 0.9833 - val_loss: 0.1014 - val_accuracy: 0.9670
Epoch 5/10
235/235 [==============================] - 11s 46ms/step - loss: 0.0380 - accuracy: 0.9877 - val_loss: 0.0817 - val_accuracy: 0.9724
Epoch 6/10
235/235 [==============================] - 17s 71ms/step - loss: 0.0272 - accuracy: 0.9913 - val_loss: 0.0720 - val_accuracy: 0.9780
Epoch 7/10
235/235 [==============================] - 9s 38ms/step - loss: 0.0214 - accuracy: 0.9930 - val_loss: 0.0635 - val_accuracy: 0.9812
Epoch 8/10
235/235 [==============================] - 13s 54ms/step - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.1050 - val_accuracy: 0.9705
Epoch 9/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0129 - accuracy: 0.9957 - val_loss: 0.0757 - val_accuracy: 0.9791
Epoch 10/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0092 - accuracy: 0.9973 - val_loss: 0.0778 - val_accuracy: 0.9812
```

0s

```python
[293] history.history['accuracy']
```

```
[0.8782166838645935,
 0.9625833630561829,
 0.9756166934967041,
 0.9833166599273682,
 0.9877499938011169,
 0.991349995136261,
 0.9930333495140076,
 0.994783341884613,
 0.9956833124160767,
 0.9972500205039978]
```

```
(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)

#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))
```


```
validation_data=(test_data, test_labels_one_hot))
```


```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 26ms/step - loss: 5.0882 - accuracy: 0.8827 - val_loss: 0.5847 - val_accuracy: 0.9360
Epoch 2/10
235/235 [==============================] - 6s 25ms/step - loss: 0.4122 - accuracy: 0.9484 - val_loss: 0.3153 - val_accuracy: 0.9487
Epoch 3/10
235/235 [==============================] - 8s 32ms/step - loss: 0.2498 - accuracy: 0.9589 - val_loss: 0.3796 - val_accuracy: 0.9494
Epoch 4/10
235/235 [==============================] - 5s 23ms/step - loss: 0.1804 - accuracy: 0.9684 - val_loss: 0.3223 - val_accuracy: 0.9550
Epoch 5/10
235/235 [==============================] - 8s 33ms/step - loss: 0.1633 - accuracy: 0.9720 - val_loss: 0.3169 - val_accuracy: 0.9589
Epoch 6/10
235/235 [==============================] - 6s 25ms/step - loss: 0.1419 - accuracy: 0.9763 - val_loss: 0.3005 - val_accuracy: 0.9664
Epoch 7/10
235/235 [==============================] - 6s 24ms/step - loss: 0.1377 - accuracy: 0.9793 - val_loss: 0.3342 - val_accuracy: 0.9587
Epoch 8/10
235/235 [==============================] - 8s 33ms/step - loss: 0.1238 - accuracy: 0.9806 - val_loss: 0.3285 - val_accuracy: 0.9737
Epoch 9/10
235/235 [==============================] - 5s 23ms/step - loss: 0.1232 - accuracy: 0.9836 - val_loss: 0.3089 - val_accuracy: 0.9712
Epoch 10/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1143 - accuracy: 0.9841 - val_loss: 0.4161 - val_accuracy: 0.9643
```

```
[295] history.history['accuracy']
```

```
[0.8827000260353088,
 0.9484000205993652,
 0.9588833451271057,
 0.9684333205223083,
 0.9720333218574524,
 0.9763000011444092,
 0.9793000221252441,
 0.9806166887283325,
 0.9836000204086304,
 0.9840999841690063]
```