Student ID: 700752115

Student Name: Rumana Shaik

Neural Networks & Deep Learning – ICP - 6

Github link: https://github.com/ShaikRumana301/Neural-Network-DL-ICP-6.git

- **Use Case Description:**
  Predicting the diabetes disease

- **Programming elements:**
  Keras Basics

- **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.

```
[1]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense

# Load dataset
dataset = pd.read_csv('diabetes.csv')
# Split dataset into features (X) and target variable (Y)
X = dataset.iloc[:, :-1]  # Features are all columns except the last one
Y = dataset.iloc[:, -1]   # Target variable is the last column

# Split dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

# Define the model
np.random.seed(155)
my_first_nn = Sequential()
# Add dense layers
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
# Additional dense layer
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3

# Output layer
my_first_nn.add(Dense(1, activation='sigmoid'))
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
print(my_first_nn.summary())
evaluation_result = my_first_nn.evaluate(X_test, Y_test)
print("Accuracy : ", evaluation_result[1]*100)
```

```
18/18 [==============================] - 0s 3ms/step - loss: 0.5880 - accuracy: 0.6696
Epoch 73/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5861 - accuracy: 0.6730
Epoch 74/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5830 - accuracy: 0.6765
Epoch 75/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5839 - accuracy: 0.6713
Epoch 76/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5823 - accuracy: 0.6748
Epoch 77/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5820 - accuracy: 0.6765
Epoch 78/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5812 - accuracy: 0.6748
Epoch 79/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5821 - accuracy: 0.6696
Epoch 80/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5795 - accuracy: 0.6730
Epoch 81/100
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                180

 dense_1 (Dense)             (None, 15)                315

 dense_2 (Dense)             (None, 10)                160

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 666 (2.60 KB)
Trainable params: 666 (2.60 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5790 - accuracy: 0.7031
Accuracy :  70.3125
```

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

```python
# Importing the libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Normalizing the data using StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

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

# Training the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/100
37/37 [==============================] - 2s 16ms/step - loss: 0.7077 - accuracy: 0.6429 - val_loss: 0.7151 - val_accuracy: 0.6044
Epoch 2/100
37/37 [==============================] - 0s 6ms/step - loss: 0.5785 - accuracy: 0.6456 - val_loss: 0.5977 - val_accuracy: 0.6044
Epoch 3/100
37/37 [==============================] - 0s 6ms/step - loss: 0.4875 - accuracy: 0.6429 - val_loss: 0.5139 - val_accuracy: 0.6044
Epoch 4/100
37/37 [==============================] - 0s 6ms/step - loss: 0.4275 - accuracy: 0.6484 - val_loss: 0.4618 - val_accuracy: 0.6264
Epoch 5/100
37/37 [==============================] - 0s 6ms/step - loss: 0.3896 - accuracy: 0.8242 - val_loss: 0.4238 - val_accuracy: 0.8462
Epoch 6/100
37/37 [==============================] - 0s 6ms/step - loss: 0.3634 - accuracy: 0.8654 - val_loss: 0.3927 - val_accuracy: 0.9011
Epoch 7/100
37/37 [==============================] - 0s 5ms/step - loss: 0.3375 - accuracy: 0.9121 - val_loss: 0.3536 - val_accuracy: 0.9780
Epoch 8/100
37/37 [==============================] - 0s 5ms/step - loss: 0.3022 - accuracy: 0.9423 - val_loss: 0.2980 - val_accuracy: 0.9780
Epoch 9/100
37/37 [==============================] - 0s 6ms/step - loss: 0.2523 - accuracy: 0.9505 - val_loss: 0.2381 - val_accuracy: 0.9890
```

```
Epoch 93/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0164 - accuracy: 0.9973 - val_loss: 0.0197 - val_accuracy: 0.9890
Epoch 94/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0161 - accuracy: 0.9973 - val_loss: 0.0195 - val_accuracy: 0.9890
Epoch 95/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0155 - accuracy: 0.9973 - val_loss: 0.0198 - val_accuracy: 0.9890
Epoch 96/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0152 - accuracy: 0.9973 - val_loss: 0.0195 - val_accuracy: 0.9890
Epoch 97/100
37/37 [==============================] - 0s 5ms/step - loss: 0.0150 - accuracy: 0.9973 - val_loss: 0.0184 - val_accuracy: 0.9890
Epoch 98/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0144 - accuracy: 0.9973 - val_loss: 0.0185 - val_accuracy: 0.9890
Epoch 99/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0140 - accuracy: 0.9973 - val_loss: 0.0188 - val_accuracy: 0.9890
Epoch 100/100
37/37 [==============================] - 0s 5ms/step - loss: 0.0137 - accuracy: 0.9973 - val_loss: 0.0178 - val_accuracy: 0.9890
4/4 [==============================] - 0s 4ms/step - loss: 0.1834 - accuracy: 0.9386
Accuracy: 93.86
```

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

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

# Load Breast Cancer dataset
data = load_breast_cancer()
X, Y = data.data, data.target

# Normalize the data
sc = StandardScaler()
X_normalized = sc.fit_transform(X)

# Split dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_normalized, Y, test_size=0.25, random_state=87)

# Define the model
np.random.seed(155)
my_first_nn = Sequential()

# Add dense layers
my_first_nn.add(Dense(20, input_dim=X_train.shape[1], activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
# Additional dense layer
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3

# Output layer
my_first_nn.add(Dense(1, activation='sigmoid'))
# Compile the model
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
print(my_first_nn.summary())
# Evaluate the model on test data
evaluation_result = my_first_nn.evaluate(X_test, Y_test)
print("Accuracy : ", evaluation_result[1]*100)  # Print accuracy value
```

```
Epoch 93/100
14/14 [==============================] - 0s 3ms/step - loss: 9.6155e-04 - accuracy: 1.0000
Epoch 94/100
14/14 [==============================] - 0s 3ms/step - loss: 8.7664e-04 - accuracy: 1.0000
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 8.4843e-04 - accuracy: 1.0000
Epoch 96/100
14/14 [==============================] - 0s 3ms/step - loss: 8.4337e-04 - accuracy: 1.0000
Epoch 97/100
14/14 [==============================] - 0s 3ms/step - loss: 8.1197e-04 - accuracy: 1.0000
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 7.7964e-04 - accuracy: 1.0000
Epoch 99/100
14/14 [==============================] - 0s 3ms/step - loss: 7.5516e-04 - accuracy: 1.0000
Epoch 100/100
14/14 [==============================] - 0s 4ms/step - loss: 7.3437e-04 - accuracy: 1.0000
Model: "sequential_2"
_________________________________________________________________
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 20)                620

 dense_8 (Dense)             (None, 15)                315

 dense_9 (Dense)             (None, 10)                160

 dense_10 (Dense)            (None, 1)                 11

=================================================================
Total params: 1106 (4.32 KB)
Trainable params: 1106 (4.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.3554 - accuracy: 0.9720
Accuracy :  97.20279574394226
```

In class programming:

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

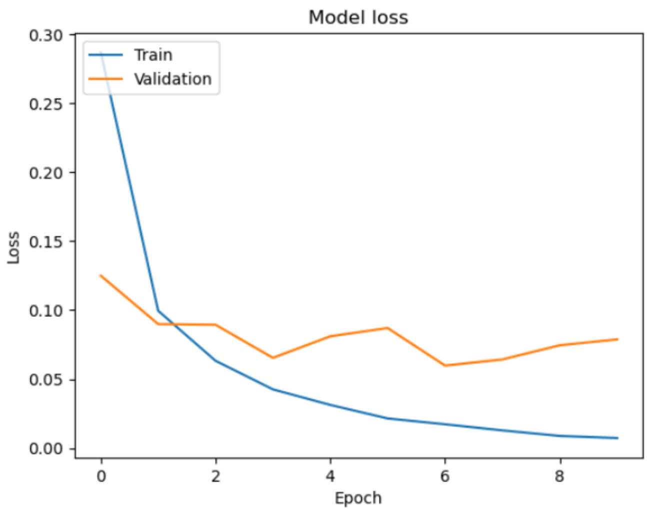
1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

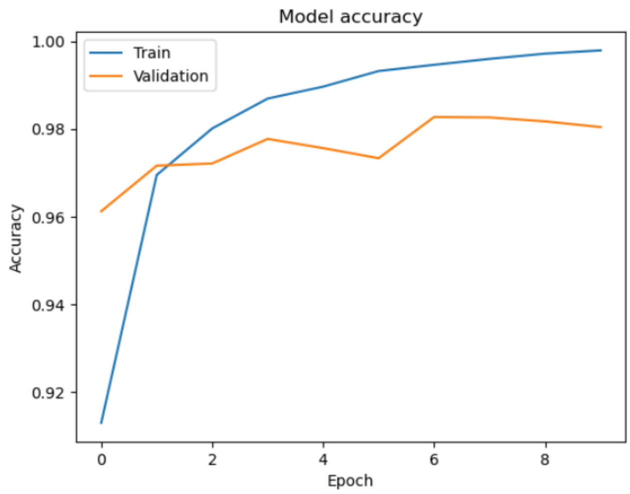
```
[4]: #Given image classification source code

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))
model.save('mnist_model.h5')
```

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 5s 18ms/step - loss: 0.2896 - accuracy: 0.9126 - val_loss: 0.1295 - val_accuracy: 0.9613
Epoch 2/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0991 - accuracy: 0.9696 - val_loss: 0.0901 - val_accuracy: 0.9729
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0625 - accuracy: 0.9804 - val_loss: 0.0911 - val_accuracy: 0.9721
Epoch 4/10
235/235 [==============================] - 3s 14ms/step - loss: 0.0421 - accuracy: 0.9864 - val_loss: 0.0840 - val_accuracy: 0.9752
Epoch 5/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0312 - accuracy: 0.9902 - val_loss: 0.0669 - val_accuracy: 0.9800
Epoch 6/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0227 - accuracy: 0.9930 - val_loss: 0.0853 - val_accuracy: 0.9775
Epoch 7/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0167 - accuracy: 0.9951 - val_loss: 0.0809 - val_accuracy: 0.9778
Epoch 8/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0121 - accuracy: 0.9963 - val_loss: 0.0778 - val_accuracy: 0.9803
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0095 - accuracy: 0.9970 - val_loss: 0.0737 - val_accuracy: 0.9825
Epoch 10/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0078 - accuracy: 0.9974 - val_loss: 0.0676 - val_accuracy: 0.9829
```

[5]:
```python
#Task 1
import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils import to_categorical
# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float') / 255.0
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float') / 255.0
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
# Define the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

# 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', 'Validation'], loc='upper left')
plt.show()
# 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', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 5s 18ms/step - loss: 0.2866 - accuracy: 0.9129 - val_loss: 0.1249 - val_accuracy: 0.9612
Epoch 2/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0994 - accuracy: 0.9695 - val_loss: 0.0898 - val_accuracy: 0.9716
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0632 - accuracy: 0.9801 - val_loss: 0.0893 - val_accuracy: 0.9721
Epoch 4/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0425 - accuracy: 0.9869 - val_loss: 0.0654 - val_accuracy: 0.9777
Epoch 5/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0313 - accuracy: 0.9896 - val_loss: 0.0810 - val_accuracy: 0.9756
Epoch 6/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0215 - accuracy: 0.9932 - val_loss: 0.0870 - val_accuracy: 0.9733
Epoch 7/10
235/235 [==============================] - 4s 18ms/step - loss: 0.0172 - accuracy: 0.9946 - val_loss: 0.0598 - val_accuracy: 0.9827
Epoch 8/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0128 - accuracy: 0.9959 - val_loss: 0.0642 - val_accuracy: 0.9826
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0088 - accuracy: 0.9972 - val_loss: 0.0745 - val_accuracy: 0.9817
Epoch 10/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0073 - accuracy: 0.9979 - val_loss: 0.0787 - val_accuracy: 0.9804
```

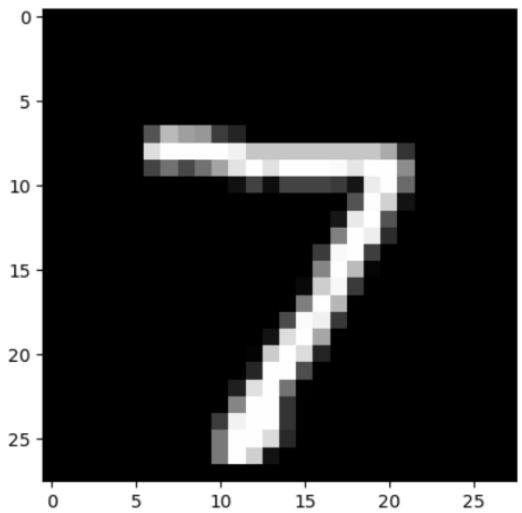




2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
[6]: import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils import to_categorical

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float') / 255.0
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float') / 255.0
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Define the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

# Fit the model
model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1)

# Plot one of the images in the test data
plt.imshow(test_images[0], cmap='gray')
plt.show()

# Make prediction on the single image
prediction = model.predict(test_data[0].reshape(1, 784))
print("Prediction:", np.argmax(prediction))
```

```
Epoch 1/10
235/235 [==============================] - 5s 16ms/step - loss: 0.2907 - accuracy: 0.9124
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0999 - accuracy: 0.9697
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0635 - accuracy: 0.9801
Epoch 4/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0445 - accuracy: 0.9859
Epoch 5/10
235/235 [==============================] - 3s 14ms/step - loss: 0.0318 - accuracy: 0.9896
Epoch 6/10
235/235 [==============================] - 3s 14ms/step - loss: 0.0222 - accuracy: 0.9929
Epoch 7/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0172 - accuracy: 0.9946
Epoch 8/10
235/235 [==============================] - 4s 15ms/step - loss: 0.0123 - accuracy: 0.9961
Epoch 9/10
235/235 [==============================] - 3s 15ms/step - loss: 0.0098 - accuracy: 0.9970
Epoch 10/10
235/235 [==============================] - 3s 15ms/step - loss: 0.0082 - accuracy: 0.9974
```



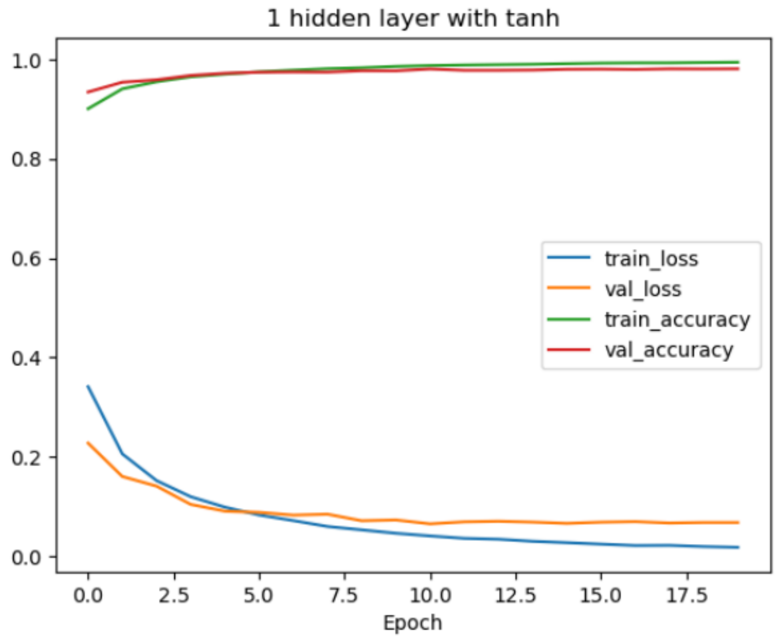
```
1/1 [==============================] - 0s 174ms/step
Prediction: 7
```

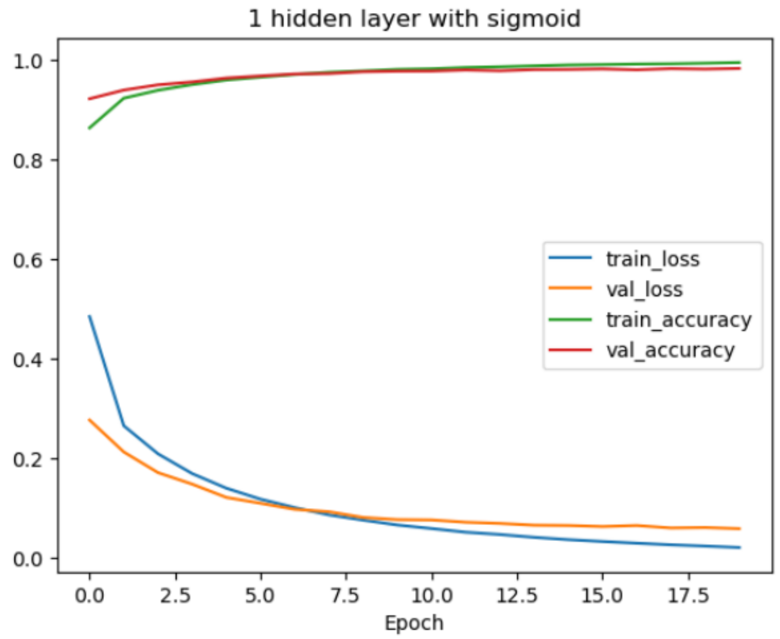
3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

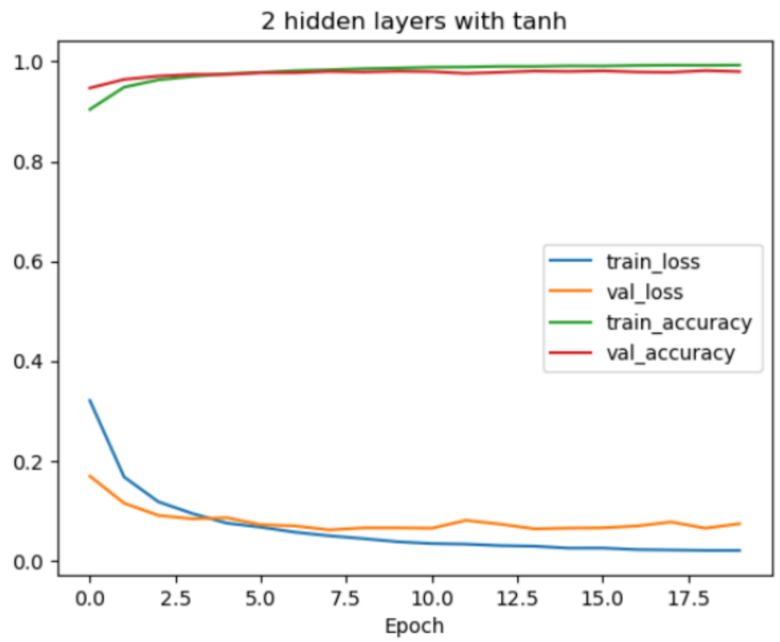
[9]:

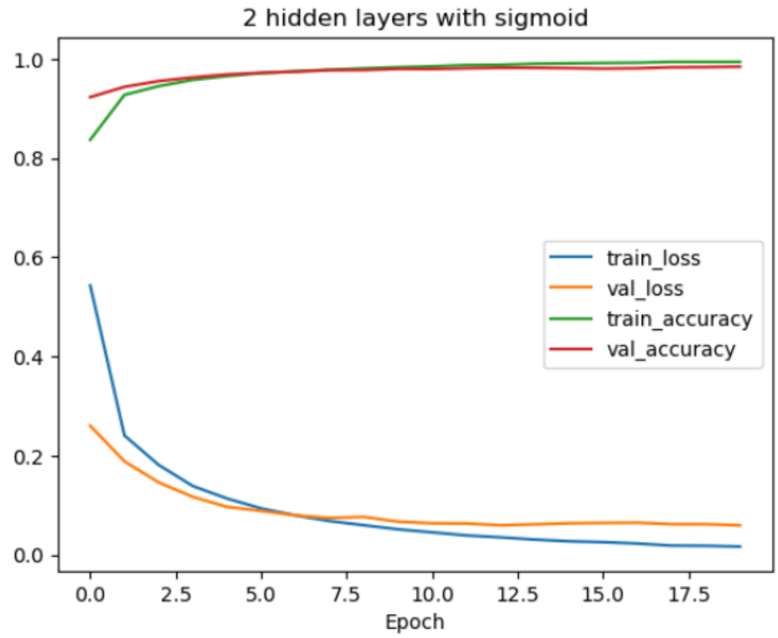
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

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

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# create a list of models to train
models = []

# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model))
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
```

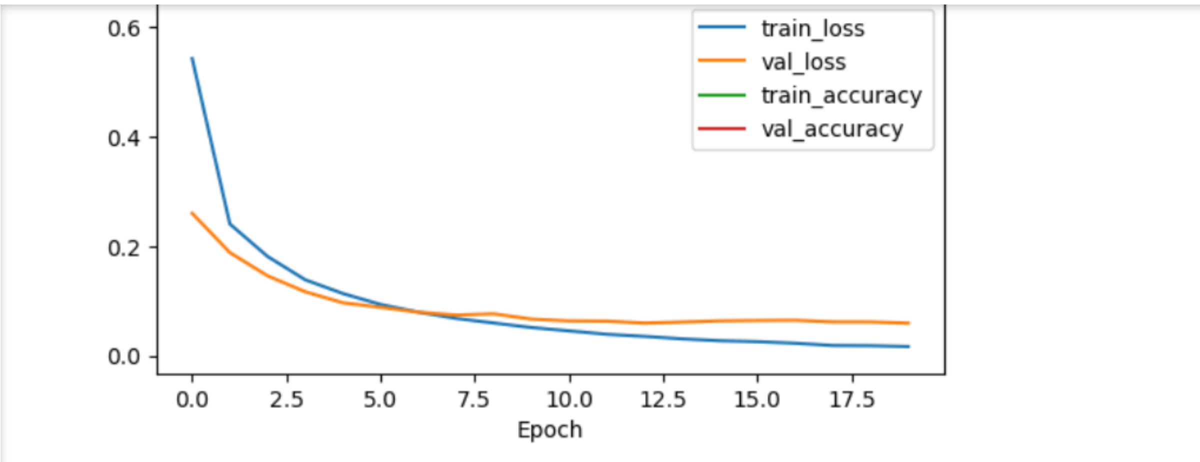
```
# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                        epochs=20, batch_size=128, verbose=0)
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()

    # evaluate the model on test data
    loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```




1 hidden layer with sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy


2 hidden layers with tanh
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy






```
2 hidden layers with sigmoid - Test loss: 0.0594, Test accuracy: 0.9843
```

4. Run the same code without scaling the images and check the performance?

[10]:
```python
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data
train_images = train_images.reshape((train_images.shape[0], -1)).astype('float32') / 255.0
test_images = test_images.reshape((test_images.shape[0], -1)).astype('float32') / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define a function to create and compile the model with specified parameters
def create_model(hidden_layers=2, activation='relu'):
    model = Sequential()
    model.add(Dense(512, activation=activation, input_shape=(784,)))
    for _ in range(hidden_layers - 1):
        model.add(Dense(512, activation=activation))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Create and train the model with 3 hidden layers and tanh activation
model_task3 = create_model(hidden_layers=3, activation='tanh')
history_task3 = model_task3.fit(train_images, train_labels, epochs=10, batch_size=128,
                                validation_data=(test_images, test_labels), verbose=1)

# Plot loss and accuracy for both training and validation data
plt.plot(history_task3.history['loss'], label='Training Loss')
plt.plot(history_task3.history['val_loss'], label='Validation Loss')
plt.plot(history_task3.history['accuracy'], label='Training Accuracy')
plt.plot(history_task3.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Loss/Accuracy')
plt.title('Training and Validation Metrics with 3 Hidden Layers and Tanh Activation')
plt.legend()
plt.show()
```

```
Epoch 1/10
469/469 [==============================] - 8s 14ms/step - loss: 0.3300 - accuracy: 0.8993 - val_loss: 0.2111 - val_accuracy: 0.9363
Epoch 2/10
469/469 [==============================] - 7s 15ms/step - loss: 0.1258 - accuracy: 0.9612 - val_loss: 0.2951 - val_accuracy: 0.9110
Epoch 3/10
469/469 [==============================] - 7s 14ms/step - loss: 0.0810 - accuracy: 0.9753 - val_loss: 0.0994 - val_accuracy: 0.9682
Epoch 4/10
469/469 [==============================] - 7s 15ms/step - loss: 0.0563 - accuracy: 0.9819 - val_loss: 0.0959 - val_accuracy: 0.9710
Epoch 5/10
469/469 [==============================] - 9s 20ms/step - loss: 0.0385 - accuracy: 0.9877 - val_loss: 0.0784 - val_accuracy: 0.9750
Epoch 6/10
469/469 [==============================] - 9s 20ms/step - loss: 0.0294 - accuracy: 0.9907 - val_loss: 0.0786 - val_accuracy: 0.9757
Epoch 7/10
469/469 [==============================] - 9s 18ms/step - loss: 0.0193 - accuracy: 0.9939 - val_loss: 0.0712 - val_accuracy: 0.9790
Epoch 8/10
469/469 [==============================] - 9s 19ms/step - loss: 0.0132 - accuracy: 0.9961 - val_loss: 0.0658 - val_accuracy: 0.9809
Epoch 9/10
469/469 [==============================] - 9s 18ms/step - loss: 0.0091 - accuracy: 0.9972 - val_loss: 0.0761 - val_accuracy: 0.9779
Epoch 10/10
469/469 [==============================] - 9s 18ms/step - loss: 0.0041 - accuracy: 0.9990 - val_loss: 0.0678 - val_accuracy: 0.9815
```