Name: konyala sai vardhan reddy

Student id:700745733

Github: https://github.com/vardhan141/icp2deep_learning

video link: https://drive.google.com/file/d/122R7ETZWyWMGeKTyZwE26FB-pvDuAPJU/view?usp=sharing

```
[1] # code to mount drive
    from google.colab import drive
    drive.mount('/content/gdrive')

    Mounted at /content/gdrive
```

```
[3] # csv file path for diabetes
    path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
```

```
[4] # importing pandas
    import pandas as pd
    # reading csv file
    dataset = pd.read_csv(path_to_csv, header=None)
    dataset.shape

    (768, 9)
```

Here we first mount the drive and we read the diabetes csv file and print the shape which is (768,9)

```
# importing keras for deep learning
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

dataset = pd.read_csv(path_to_csv, header=None).values
# splitting test and train data
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'))
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 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5656 - acc: 0.7066
Epoch 96/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5850 - acc: 0.7257
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5531 - acc: 0.7292
Epoch 98/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5635 - acc: 0.7205
Epoch 99/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5608 - acc: 0.7205
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5644 - acc: 0.7274
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                180

 dense_1 (Dense)             (None, 1)                 21

=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.6841 - acc: 0.6458
[0.6840513348579407, 0.6458333134651184]
```

here we used the sample code given for creating model and got 72 percent accuracy and test accuracy is 0.6458

1st question

```
1. Use the use case in the class:
a. Add more Dense layers to the existing code and check how the accuracy changes
```

```
[6]
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

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)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer1
my_first_nn.add(Dense(40, input_dim=8, activation='relu')) # hidden layer2
my_first_nn.add(Dense(40, input_dim=8, activation='relu')) # hidden layer 3
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))
```

Here I added two more layers and ran the model

```
Epoch 96/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5296 - acc: 0.7344
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5104 - acc: 0.7500
Epoch 98/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5485 - acc: 0.7344
Epoch 99/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5049 - acc: 0.7413
Epoch 100/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5095 - acc: 0.7483
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 20)                180

 dense_3 (Dense)             (None, 40)                840

 dense_4 (Dense)             (None, 40)                1640

 dense_5 (Dense)             (None, 1)                 41

=================================================================
Total params: 2,701
Trainable params: 2,701
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 8ms/step - loss: 0.5449 - acc: 0.7292
[0.5449329614639282, 0.7291666865348816]
```

We got the train accuracy as 74% and test accuracy as 72 which better due to adding layers

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

```
# 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)

dataset['diagnosis'] = dataset['diagnosis'].map({'M':1,'B':0})
test=dataset['diagnosis']
# dropping diagnosis because it is target variable and id and unnamed because they are not useful
dataset=dataset.drop(['diagnosis','id','Unnamed: 32'],axis=1)
X_train, X_test, Y_train, Y_test = train_test_split(dataset, test, stratify=test, test_size=0.2,shuffle=True, random_state=5)

#training the model
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(30, input_dim=30, 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))
```

here we use the breast cancer dataset where we map the diagnosis column to 1 and 0 and drop diagnosis and id and unnamed as they are not useful.

Here the train dataset has 30 columns because we dropped 3 columns and used Adam optimizer

```
15/15 [==============================] - 0s 5ms/step - loss: 0.1147 - acc: 0.9516
Epoch 96/100
15/15 [==============================] - 0s 4ms/step - loss: 0.3208 - acc: 0.9011
Epoch 97/100
15/15 [==============================] - 0s 4ms/step - loss: 0.1332 - acc: 0.9495
Epoch 98/100
15/15 [==============================] - 0s 5ms/step - loss: 0.1589 - acc: 0.9363
Epoch 99/100
15/15 [==============================] - 0s 5ms/step - loss: 0.1341 - acc: 0.9451
Epoch 100/100
15/15 [==============================] - 0s 5ms/step - loss: 0.1185 - acc: 0.9560
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 30)                930

 dense_7 (Dense)             (None, 1)                 31

=================================================================
Total params: 961
Trainable params: 961
Non-trainable params: 0
_________________________________________________________________
None
4/4 [==============================] - 0s 7ms/step - loss: 0.1403 - acc: 0.9298
[0.14031684398651123, 0.9298245906829834]
```

The train accuracy is 96 and test accuracy is 92%

Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

[9]
```
#normalize the data using StandardScaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(dataset)
dataset_scaled = sc.transform(dataset)
```

```
)] X_train, X_test, Y_train, Y_test = train_test_split(dataset, test, stratify=test, test_size=0.2,shuffle=True, random_state=5)

   #training the model
   np.random.seed(155)
   my_first_nn = Sequential() # create model
   my_first_nn.add(Dense(30, input_dim=30, 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 99/100
15/15 [==============================] - 0s 3ms/step - loss: 0.1250 - acc: 0.9429
Epoch 100/100
15/15 [==============================] - 0s 3ms/step - loss: 0.1725 - acc: 0.9275
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 30)                930

 dense_9 (Dense)             (None, 1)                 31

=================================================================
Total params: 961
Trainable params: 961
Non-trainable params: 0
_________________________________________________________________
None
4/4 [==============================] - 0s 4ms/step - loss: 0.2329 - acc: 0.9123
[0.2329448163509369, 0.9122806787490845]
```

The train accuracy and test accuracy are given by 92 and 91 after scaling

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

```
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)
```

```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))
```

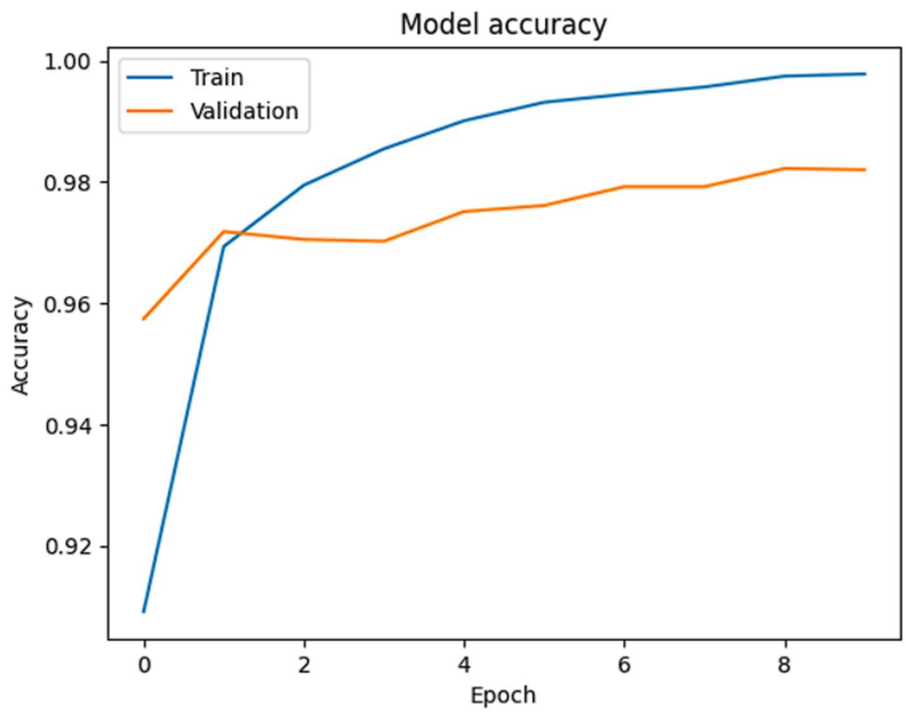
```
235/235 [==============================] - 1s 5ms/step - loss: 0.0653 - accuracy: 0.9794 - val_loss: 0.0980 - val_accuracy: 0.9705
Epoch 4/10
235/235 [==============================] - 1s 6ms/step - loss: 0.0454 - accuracy: 0.9855 - val_loss: 0.0950 - val_accuracy: 0.9702
Epoch 5/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0317 - accuracy: 0.9901 - val_loss: 0.0843 - val_accuracy: 0.9751
Epoch 6/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0220 - accuracy: 0.9931 - val_loss: 0.0852 - val_accuracy: 0.9761
Epoch 7/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0174 - accuracy: 0.9945 - val_loss: 0.0696 - val_accuracy: 0.9792
Epoch 8/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0133 - accuracy: 0.9956 - val_loss: 0.0776 - val_accuracy: 0.9792
Epoch 9/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0090 - accuracy: 0.9974 - val_loss: 0.0673 - val_accuracy: 0.9822
Epoch 10/10
235/235 [==============================] - 1s 4ms/step - loss: 0.0074 - accuracy: 0.9978 - val_loss: 0.0686 - val_accuracy: 0.9820
```

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

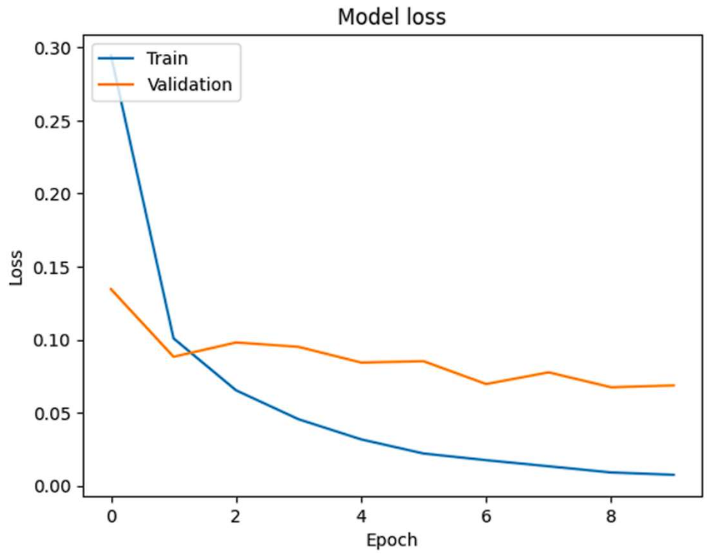
The train accuracy and test accuracy for mnist dataset are 99 and 98 percent

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
    # Plot the training and validation accuracy over epochs
    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()
```




```
# Plot the training and validation loss over epochs
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()
```



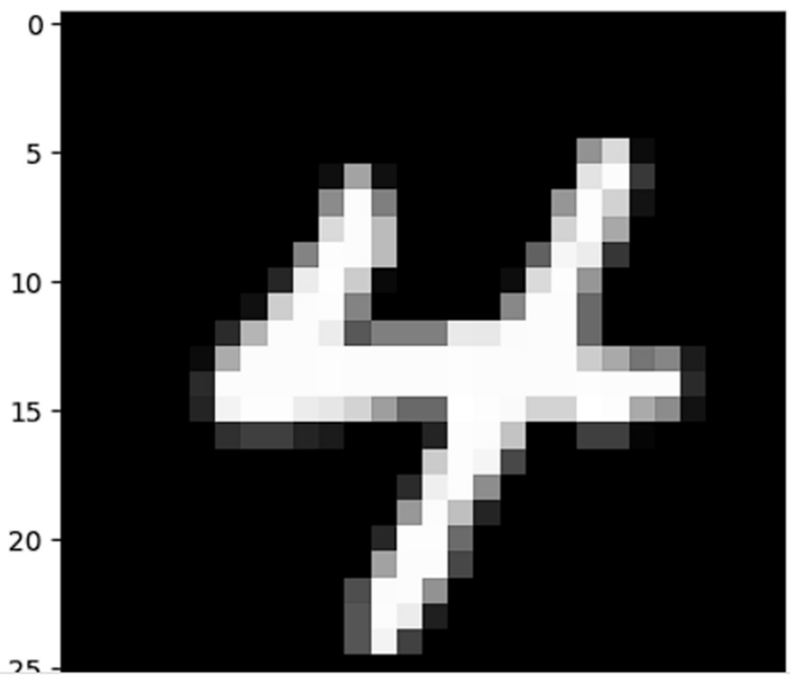
Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
# Select a random image from the test data
idx = np.random.randint(0, test_images.shape[0])
img = test_images[idx]

# Plot the selected image
plt.imshow(img, cmap='gray')
plt.show()

input_image = img.reshape(1, 784).astype('float32') / 255.0

prediction = model.predict(input_image)
print('The image is predicted as:', np.argmax(prediction))
```

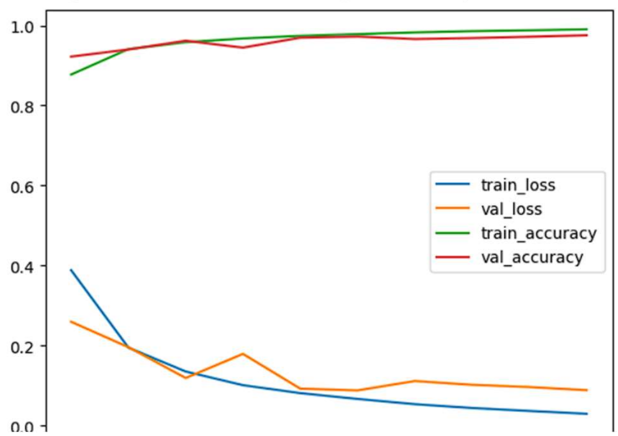


We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

```
[ ] #creating network
    model = Sequential()
    model.add(Dense(128, activation='tanh', input_shape=(dimData,)))
    model.add(Dense(256, activation='tanh'))
    model.add(Dense(128, activation='tanh'))
    model.add(Dense(512, 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))
     # 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.xlabel('Epoch')
    plt.legend()
    plt.show()
```

```
] 235/235 [==============================] - 1s 4ms/step - loss: 0.0816 - accuracy: 0.9739 - val_loss: 0.0928 - val_accuracy: 0.9694
  Epoch 6/10
  235/235 [==============================] - 1s 4ms/step - loss: 0.0671 - accuracy: 0.9784 - val_loss: 0.0884 - val_accuracy: 0.9722
  Epoch 7/10
  235/235 [==============================] - 1s 6ms/step - loss: 0.0540 - accuracy: 0.9825 - val_loss: 0.1118 - val_accuracy: 0.9660
  Epoch 8/10
  235/235 [==============================] - 1s 6ms/step - loss: 0.0444 - accuracy: 0.9857 - val_loss: 0.1023 - val_accuracy: 0.9683
  Epoch 9/10
  235/235 [==============================] - 1s 4ms/step - loss: 0.0371 - accuracy: 0.9878 - val_loss: 0.0968 - val_accuracy: 0.9717
  Epoch 10/10
  235/235 [==============================] - 1s 5ms/step - loss: 0.0300 - accuracy: 0.9903 - val_loss: 0.0890 - val_accuracy: 0.9754
```



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

```
[ ] 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')

    #Commenting the scale data part
    #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(128, activation='tanh', input_shape=(dimData,)))
    model.add(Dense(256, activation='tanh'))
    model.add(Dense(128, activation='tanh'))
    model.add(Dense(512, activation='tanh'))
    model.add(Dense(10, activation='softmax'))

    #Feeding the unscaled data to the network
    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))
     # 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.xlabel('Epoch')
    plt.legend()
    plt.show()
```

```
[ ] 235/235 [==============================] - 1s 5ms/step - loss: 0.1403 - accuracy: 0.9552 - val_loss: 0.1687 - val_accuracy: 0.9469
    Epoch 10/10
    235/235 [==============================] - 2s 7ms/step - loss: 0.1295 - accuracy: 0.9592 - val_loss: 0.1448 - val_accuracy: 0.9564
```