Project Description (House Loan Data Analysis):

“””

Facial recognition is a biometric alternative that measures unique characteristics of a human face. Applications available today include flight check in, tagging friends and family members in photos, and “tailored” advertising. You are a computer vision engineer who needs to develop a face recognition programme with deep convolutional neural networks.

Objective: Use a deep convolutional neural network to perform facial recognition using Keras.

Dataset Details:

ORL face database composed of 400 images of size 112 x 92. There are 40 people, 10 images per person. The images were taken at different times, lighting and facial expressions. The faces are in an upright position in frontal view, with a slight left-right rotation.

Link to the Dataset: https://www.dropbox.com/s/i7uzp5yxk7wruva/ORL_faces.npz?dl=0

Steps to be done:

- Input the required libraries
- Load the dataset after loading the dataset, you have to normalize every image.
- Split the dataset
- Transform the images to equal sizes to feed in CNN
- Build a CNN model that has 3 main layers:
    i. Convolutional Layer
    ii. Pooling Layer
    iii. Fully Connected Layer
- Train the model
- Plot the result
- Iterate the model until the accuracy is above 90%

“””

Source Code:

```
#importing Libraries


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import itertools


#Scikit-learn libraries

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import roc_curve,auc


#Keras API Tensorflow 2 libraries

import tensorflow as tf

import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,
Activation, LeakyReLU
```

```
from keras.layers.noise import AlphaDropout

from tensorflow.keras.optimizers import Adam


from keras.utils.generic_utils import get_custom_objects

from keras import backend as K

from keras.callbacks import TensorBoard

from keras.utils.np_utils import to_categorical



#loading Dataset

print("loading Dataset")

print("-------------------------------------")

file_path=input("enter path for the loan data file to load:")

df_path=file_path.replace("\\",'/')

df=np.load(df_path)


# Loading train and test dataset (data is already split into)

print("-------------------------------------\n")

print("splitting the dataset:\n")

x_train = df['trainX']

y_train = df['trainY']
```

```
x_test = df['testX']

y_test = df['testY']



# Normalizing each image as each image is between 0-255 pixels

print("------------------------------------------------------------\n")

print("transforming images into equal sizes to feed in CNN")

print("------------------------------------------------------------")

x_train = x_train.astype(np.float32) / 255.0

x_test = x_test.astype(np.float32) / 255.0

print('Training dataset shape: ',x_train.shape)

print('Testing dataset shape: ',x_test.shape)



x_train, x_valid, y_train, y_valid =
train_test_split(x_train,y_train,test_size=0.1,random_state=42)



# Shape of image definition

rows = 112

columns = 92

image_shape = (rows,columns,1)
```

```
# Reshape function

x_train = x_train.reshape(x_train.shape[0],*image_shape)

x_test = x_test.reshape(x_test.shape[0],*image_shape)

x_valid = x_valid.reshape(x_valid.shape[0],*image_shape)



print('Training dataset modified shape: ',x_train.shape)

print('Testing dataset modified shape: ',x_test.shape)

print('Validating dataset modified shape: ',x_valid.shape)


get_custom_objects().update({'leaky-relu': Activation(LeakyReLU(alpha=0.2))})

#Building a CNN model

print("-----------------------------------------\n")

print("Building CNN Model")

print("-----------------------------------------\n")

model = Sequential()


model.add(Conv2D(32, kernel_size=3,activation='leaky-
relu',input_shape=image_shape)) #32 filter with kernel size of 3 x 3 with input
shape

model.add(MaxPooling2D(pool_size=2))
```

```
model.add(Conv2D(64,3, activation='leaky-relu')) #64 filter with kernel size of 3 x 3

model.add(MaxPooling2D(pool_size=2)) #Max pool with size of 2


model.add(Flatten())


model.add(Dense(2024, activation='leaky-relu'))

model.add(Dropout(0.5))

model.add(Dense(1024, activation='leaky-relu'))

model.add(Dropout(0.5))

model.add(Dense(512, activation='leaky-relu'))

model.add(Dropout(0.5))


model.add(Dense(20, activation='softmax')) #Output layer


model.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(clipvalue=0.5),metrics=['accuracy'])


#Training the model

print("-----------------------------------\n")

print("Training the Model")
```

```
print("-----------------------------------\n")
history = model.fit(np.array(x_train), np.array(y_train),
                    batch_size=512,
                    epochs=75,
                    verbose=2,
                    validation_data=(np.array(x_valid),np.array(y_valid)))

result_score = model.evaluate(np.array(x_test),np.array(y_test),verbose=0)
print("------------------------------")
print("Model Results:\n")

print('Test Loss {:.4f}'.format(result_score[0]))
print('Test Accuracy {:.4f}'.format(result_score[1]))



#Plotting Accuracy for the model
print("-----------------------------------\n")
print("plotting model Accuracy")
print("-----------------------------------\n")
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel(' No. of Epochs')
plt.legend(['Train', 'Valid'])
plt.grid()
plt.show()

#Plotting Loss for the model
print("-----------------------------------\n")
print("plotting Model loss")
print("-----------------------------------\n")
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('No. of Epochs')
plt.legend(['Train', 'Valid'])
plt.grid()
plt.show()
```

Screenshot of the output:

Loading Dataset:

```
loading Dataset
-------------------------------------
enter path for the loan data file to load:C:\Users\sweth\Downloads\ORL_faces.npz
-------------------------------------
```

Transforming Images into equal size to build CNN:

```
transforming images into equal sizes to feed in CNN
-------------------------------------------------------------
Training dataset shape:  (240, 10304)
Testing dataset shape:  (160, 10304)
Training dataset modified shape:  (216, 112, 92, 1)
Testing dataset modified shape:  (160, 112, 92, 1)
Validating dataset modified shape:  (24, 112, 92, 1)
-----------------------------------------------
```

Building CNN Model:

```
Building CNN Model
-------------------------------------------

2021-11-08 12:52:24.281669: W tensorflow/stream_executor/platform/default/dso_loader.cc:6
2021-11-08 12:52:24.281791: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed
2021-11-08 12:52:24.286996: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] re
2021-11-08 12:52:24.287269: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] ho
2021-11-08 12:52:24.287579: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Ten
ormance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flag
-------------------------------------

Training the Model
-------------------------------------
```

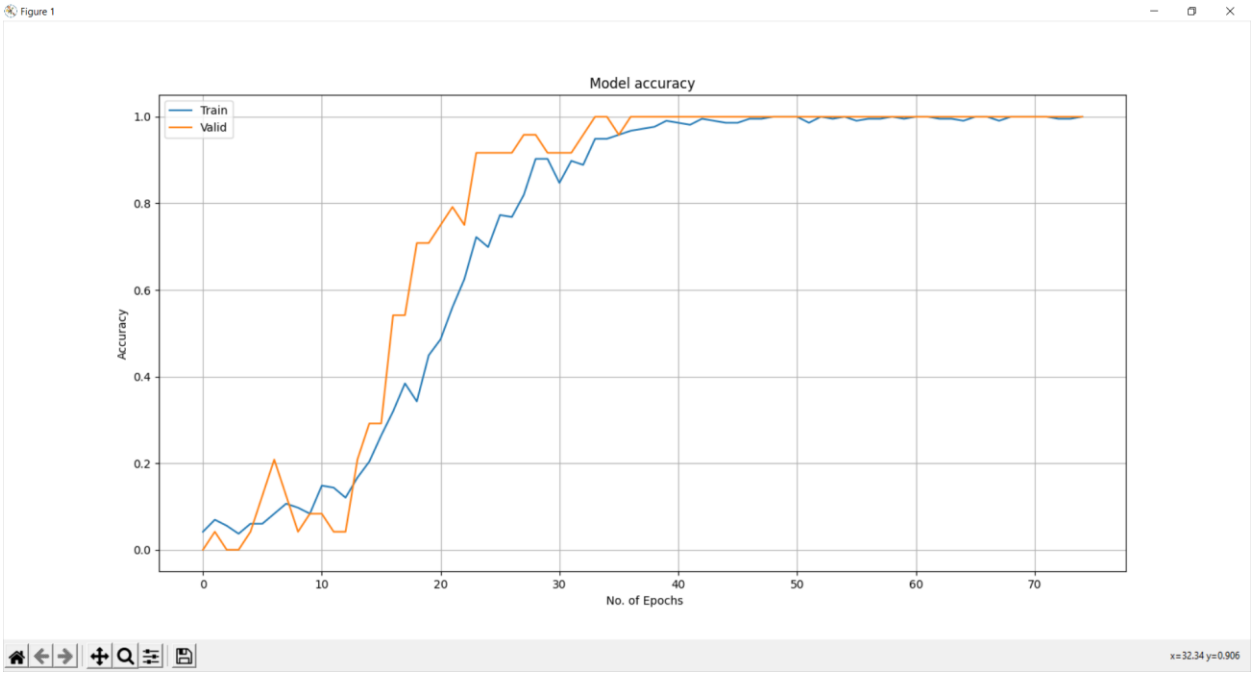
Training Model:

```
Training the Model
--------------------------------------

2021-11-08 12:52:24.750685: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/75
1/1 - 2s - loss: 3.0102 - accuracy: 0.0417 - val_loss: 3.7108 - val_accuracy: 0.0000e+00
Epoch 2/75
1/1 - 1s - loss: 4.5562 - accuracy: 0.0694 - val_loss: 3.3109 - val_accuracy: 0.0417
Epoch 3/75
1/1 - 2s - loss: 4.4479 - accuracy: 0.0556 - val_loss: 2.9826 - val_accuracy: 0.0000e+00
Epoch 4/75
1/1 - 1s - loss: 3.5687 - accuracy: 0.0370 - val_loss: 2.9919 - val_accuracy: 0.0000e+00
Epoch 5/75
1/1 - 1s - loss: 3.0230 - accuracy: 0.0602 - val_loss: 2.9865 - val_accuracy: 0.0417
Epoch 6/75
1/1 - 1s - loss: 2.9738 - accuracy: 0.0602 - val_loss: 3.1550 - val_accuracy: 0.1250
Epoch 7/75
1/1 - 2s - loss: 3.7688 - accuracy: 0.0833 - val_loss: 2.9520 - val_accuracy: 0.2083
Epoch 8/75
1/1 - 2s - loss: 3.0781 - accuracy: 0.1065 - val_loss: 2.9286 - val_accuracy: 0.1250
Epoch 9/75
1/1 - 2s - loss: 2.9515 - accuracy: 0.0972 - val_loss: 2.9505 - val_accuracy: 0.0417
Epoch 10/75
1/1 - 2s - loss: 2.9323 - accuracy: 0.0833 - val_loss: 2.9372 - val_accuracy: 0.0833
Epoch 11/75
1/1 - 2s - loss: 2.8608 - accuracy: 0.1481 - val_loss: 2.8961 - val_accuracy: 0.0833
Epoch 12/75
1/1 - 2s - loss: 2.8296 - accuracy: 0.1435 - val_loss: 2.8345 - val_accuracy: 0.0417
Epoch 13/75
1/1 - 2s - loss: 2.8039 - accuracy: 0.1204 - val_loss: 2.7884 - val_accuracy: 0.0417
Epoch 14/75
1/1 - 2s - loss: 2.7422 - accuracy: 0.1667 - val_loss: 2.7387 - val_accuracy: 0.2083
Epoch 15/75
1/1 - 2s - loss: 2.6491 - accuracy: 0.2037 - val_loss: 2.6672 - val_accuracy: 0.2917
Epoch 16/75
1/1 - 2s - loss: 2.5590 - accuracy: 0.2639 - val_loss: 2.5769 - val_accuracy: 0.2917
Epoch 17/75
1/1 - 2s - loss: 2.4543 - accuracy: 0.3194 - val_loss: 2.4749 - val_accuracy: 0.5417
Epoch 18/75
1/1 - 2s - loss: 2.3155 - accuracy: 0.3843 - val_loss: 2.3214 - val_accuracy: 0.5417
Epoch 19/75
1/1 - 2s - loss: 2.1975 - accuracy: 0.3426 - val_loss: 2.1026 - val_accuracy: 0.7083
Epoch 20/75
1/1 - 2s - loss: 2.0047 - accuracy: 0.4491 - val_loss: 1.8807 - val_accuracy: 0.7083
Epoch 21/75
1/1 - 2s - loss: 1.7903 - accuracy: 0.4861 - val_loss: 1.6122 - val_accuracy: 0.7500
Epoch 22/75
1/1 - 2s - loss: 1.6104 - accuracy: 0.5602 - val_loss: 1.3776 - val_accuracy: 0.7917
Epoch 23/75
1/1 - 2s - loss: 1.4027 - accuracy: 0.6250 - val_loss: 1.1831 - val_accuracy: 0.7500
```

Model Result:

```
Model Results:

Test Loss 0.3284
Test Accuracy 0.9438
--------------------------------------
```

Plot of Model Accuracy:



Plot of Model Loss: