Machine Learning Project

Domain: Face Recognition

I. Initial Release

Tasks:

1. Classify photos as "George Bush" and "Not George Bush".

2. Classify photos as "Serena Williams" "Not Serena Williams".

Dataset Description: A total of 13,233 photos which 64X64 gray-scale photos and are csv format files

- X.csv: Data for all where each row corresponds to a photo and columns correspond to pixels.
- y_bush_vs_others.csv: Labels for task 1
- y_williams_vs_others.csv: Labels for task 2

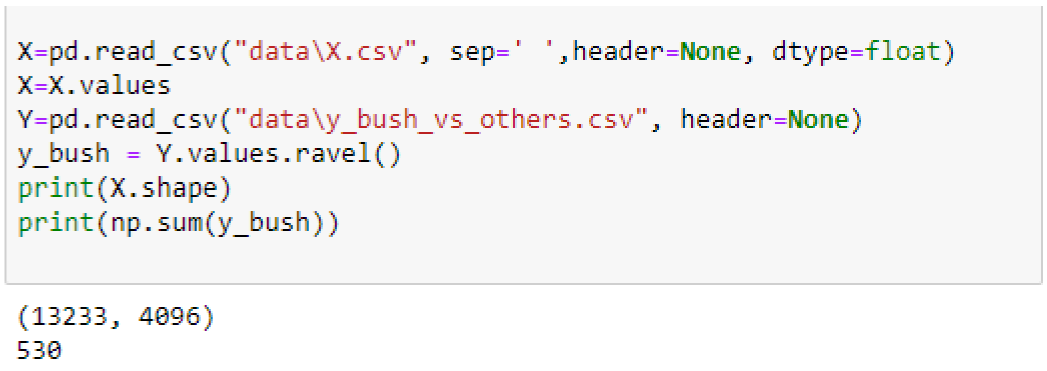
Performance Metrics: For this project, we have mainly precision, recall and F1 for class 1 as performance metric that is to understand how well results, we obtain through execution of classifiers and models

II. Project Phase 1

Description: Here we first load the csv files, choose a classifier for it and then perform three-fold cross validation.

- The datasets were loaded in the following way

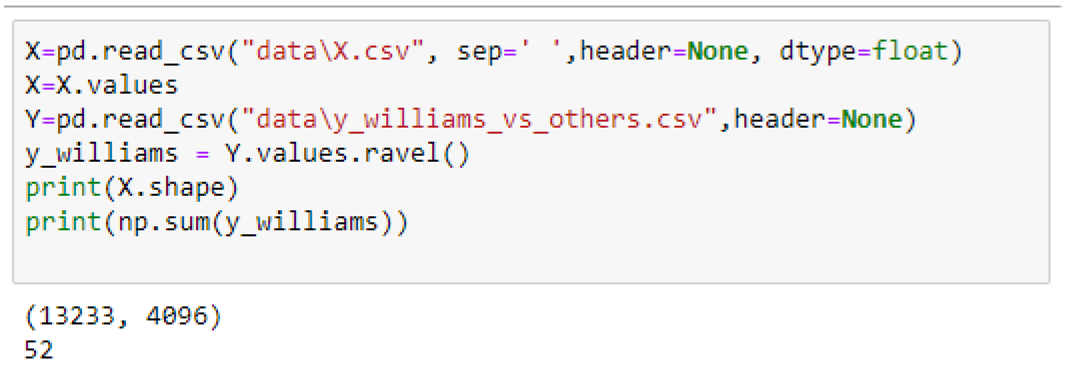
i. For Bush



```
X=pd.read_csv("data\X.csv", sep=' ',header=None, dtype=float)
X=X.values
Y=pd.read_csv("data\y_bush_vs_others.csv", header=None)
y_bush = Y.values.ravel()
print(X.shape)
print(np.sum(y_bush))
```

```
(13233, 4096)
530
```

ii. For Williams



```
X=pd.read_csv("data\X.csv", sep=' ',header=None, dtype=float)
X=X.values
Y=pd.read_csv("data\y_williams_vs_others.csv",header=None)
y_williams = Y.values.ravel()
print(X.shape)
print(np.sum(y_williams))
```

```
(13233, 4096)
52
```

- The classifier which were used for this project for both Bush and Williams datasets are

i. KNeighborsClassifier having n_neighbors=1,3 and 5

```
clf=KNeighborsClassifier(n_neighbors=1)
```

```
clf=KNeighborsClassifier(n_neighbors=3)
```

```
clf=KNeighborsClassifier(n_neighbors=5)
```

ii. SVC or Support Vector Classification with various parameters setting to obtain the best results in F1 Parameters for the classifier: Various C values; Kernel as rbf, linear and poly; gamma for rbf kernel; degree values for poly kernel

```
svc=svm.SVC(C=1,kernel='linear')
```

```
clf=svm.SVC(C=1,kernel='rbf',gamma=value)
```

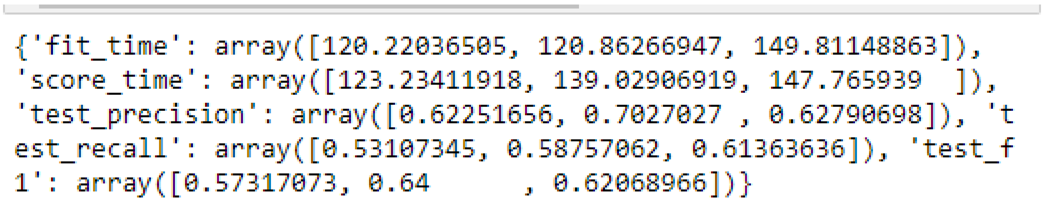
Where gamma value was computed as value=(1/(X.shape[1]*X.std()))

```
clf=svm.SVC(C=10,kernel='poly',degree=3,coef0=1)
```

- Now using three-fold cross validation, we can validate our classifier for data by

stratified_cv_results = cross_validate(clf, X, y, cv=StratifiedKFold(n_splits = 3, shuffle=True, random_state = 8401), scoring=('precision', 'recall', 'f1'), return_train_score=False)

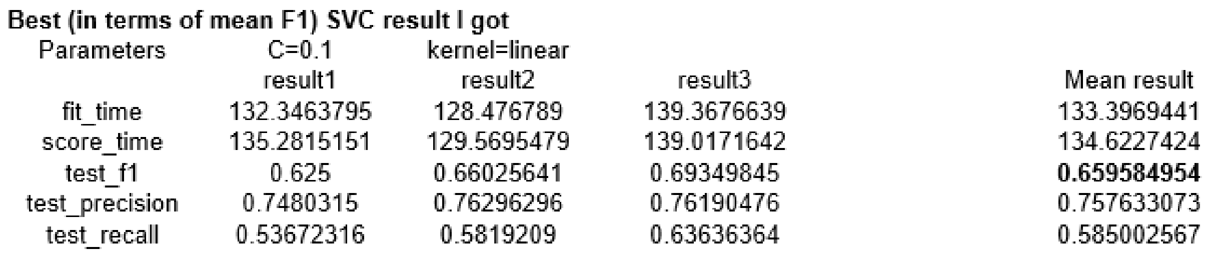
This returns Precision, Recall and F1 values after executing validation classifier with various settings for data which can be shown as below



```
{'fit_time': array([120.22036505, 120.86266947, 149.81148863]),
'score_time': array([123.23411918, 139.02906919, 147.765939  ]),
'test_precision': array([0.62251656, 0.7027027 , 0.62790698]), 't
est_recall': array([0.53107345, 0.58757062, 0.61363636]), 'test_f
1': array([0.57317073, 0.64       , 0.62068966])}
```

- Now we do the mean of F1 score obtained and record the mean F1 score for n-neighbors=1,3 and 5. For SVC classifier, we choose the best SVC parameter tuning which helped to get better mean F1 scores. We also, record the SVC classifier parameter tuning giving mean F1 as zero in excel sheet
- The SVC classifier parameter tuning which gave better mean F1 scores comparatively

i. For Bush: Parameters: C=0.1, Kernel = Linear



Best (in terms of mean F1) SVC result I got

Parameters	C=0.1	kernel=linear		
	result1	result2	result3	Mean result
fit_time	132.3463795	128.476789	139.3676639	133.3969441
score_time	135.2815151	129.5695479	139.0171642	134.6227424
test_f1	0.625	0.66025641	0.69349845	**0.659584954**
test_precision	0.7480315	0.76296296	0.76190476	0.757633073
test_recall	0.53672316	0.5819209	0.63636364	0.585002567

ii. **For Williams: Parameters: C=1, Kernel = Linear**

Best (in terms of mean F1) SVC result I got

Parameters	C=1	kernel=linear		
	result1	result2	result3	Mean result
fit_time	25.80657268	24.64823556	26.29670453	25.58383759
score_time	26.84471703	25.68049312	27.13464499	26.55328505
test_f1	0.42857143	0.44444444	0.57142857	**0.481481481**
test_precision	0.6	0.6	0.72727273	0.642424242
test_recall	0.33333333	0.35294118	0.47058824	0.385620915

- So, the final results obtained for KNN and SVC classifier in this phase

i. For Bush

Classifier	Parameters	Mean F1
KNeighborsClassifier	n_neighbors=1	0.132655799
KNeighborsClassifier	n_neighbors=3	0.048719958
KNeighborsClassifier	n_neighbors=5	0.018018653
SVC (Best result)	C=0.1,kernel=linear	0.659584954

ii. For Williams

Classifier	Parameters	Mean F1
KNeighborsClassifier	n_neighbors=1	0.211904762
KNeighborsClassifier	n_neighbors=3	0
KNeighborsClassifier	n_neighbors=5	0
SVC (Best result)	C=1,kernel= 'linear'	0.481481481

III. Project Phase 2

Description: For this phase, we need to perform Principle Component Analysis(PCA) to reduce the dimensionality of dataset. PCA basically transforms a number of correlated variables in data and reduces the size of dataset. Our aim was to tune various parameters such as n_components for PCA and get lowest number of components for which we get good mean F1 score comparatively

- Load the dataset similar to phase 1. Define PCA and then perform fit_transform function on data

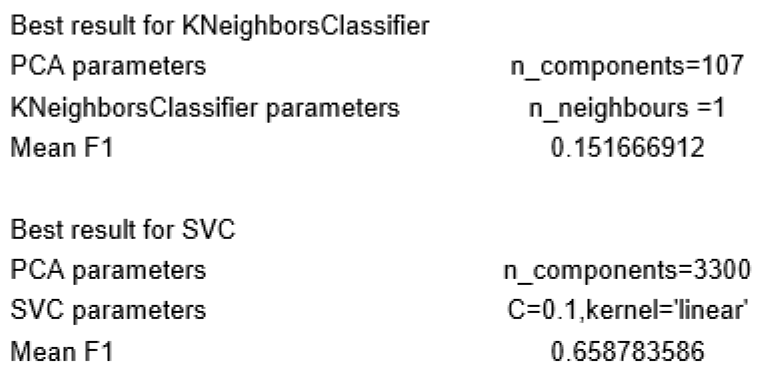
```
pca = PCA(n_components=2000)
X_pca=pca.fit_transform(X)
```

- Next similar to phase 1, we choose a classifier KNN with n_neighbors 1,3 and 5 or SVC with various parameters tuning. Following that, validation is performed using

  stratified_cv_results=cross_validate(clf, X_pca, y, cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=8401), scoring=('precision','recall','f1'), return_train_score=False)

- Thus, we record the data obtained this way and following results are the final results obtained in this phase of project

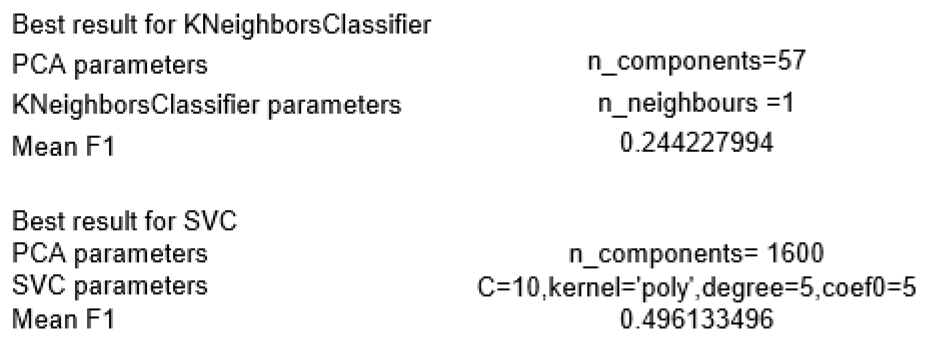
i. For Bush: KNN: n_components=107; n_neighbors=1
SVC: n_components=3300; C=0.1; kernel= linear



Best result for KNeighborsClassifier	
PCA parameters	n_components=107
KNeighborsClassifier parameters	n_neighbours =1
Mean F1	0.151666912

Best result for SVC	
PCA parameters	n_components=3300
SVC parameters	C=0.1,kernel='linear'
Mean F1	0.658783586

ii. **For Williams: KNN: n_components=57; n_neighbors=1**
SVC: n_components=1600; C=10; kernel= poly; degree=5; coef0=5



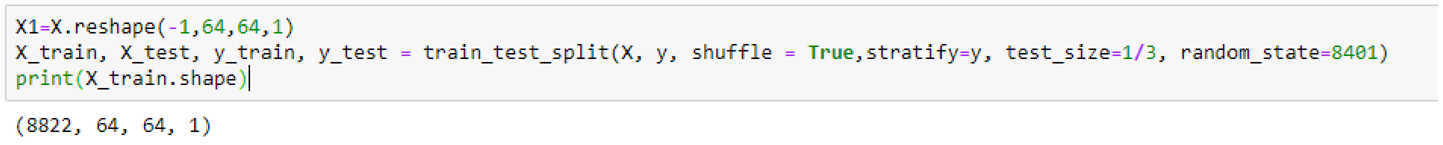
Best result for KNeighborsClassifier	
PCA parameters	n_components=57
KNeighborsClassifier parameters	n_neighbours =1
Mean F1	0.244227994

Best result for SVC	
PCA parameters	n_components= 1600
SVC parameters	C=10,kernel='poly',degree=5,coef0=5
Mean F1	0.496133496

IV. Project Phase 3

Description: In this phase, we developed a deep learning model for image classification using Convolutional Neural Networks and Maxpooling. Usage of keras model was done in order to make an efficient deep learning model and perform it on data which is split using train_test_split.

- Load the dataset and then splitting it using train_test_split and randomn state as 8401.Also, we need to perform reshaping and change the size to 64x64 image. This can be given as below



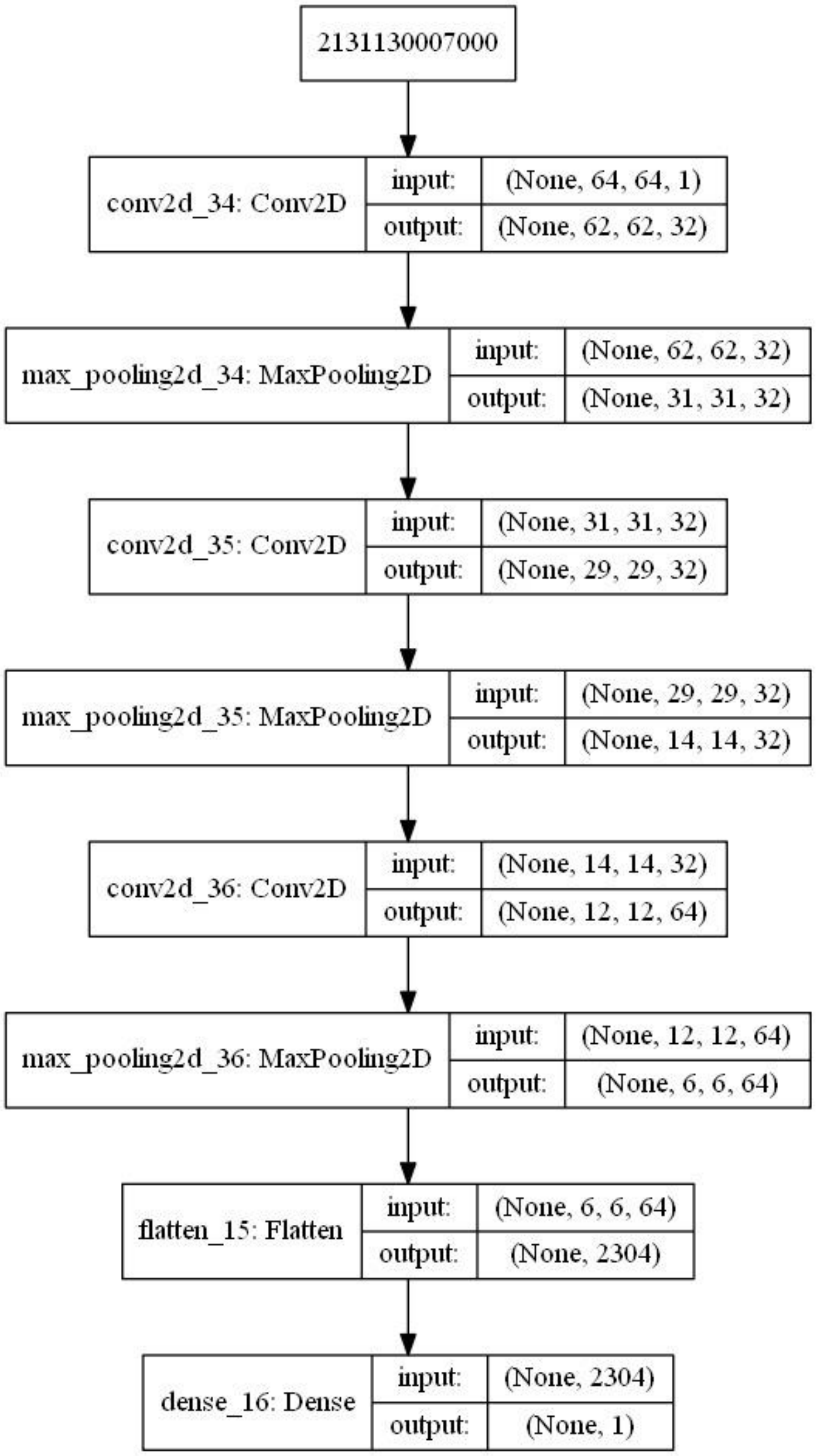
```
X1=X.reshape(-1,64,64,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle = True,stratify=y, test_size=1/3, random_state=8401)
print(X_train.shape)
```

```
(8822, 64, 64, 1)
```

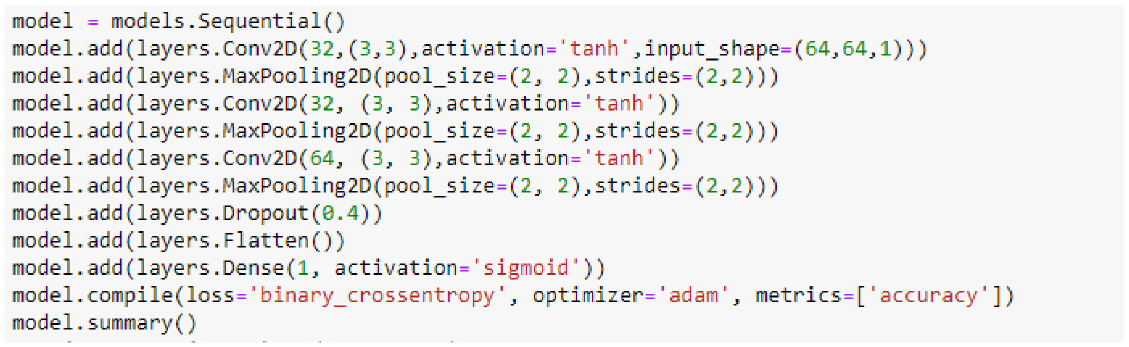
- Now we use keras to develop deep learning model using Convolutional Neural Network (Conv2D) and Maxpooling. The output layer is single node with a sigmoid activation. The architecture for both Bush and Williams can be given as

i. **For Bush**
As seen in model below, there are 3 layers of 2D-CNN followed by Maxpooling

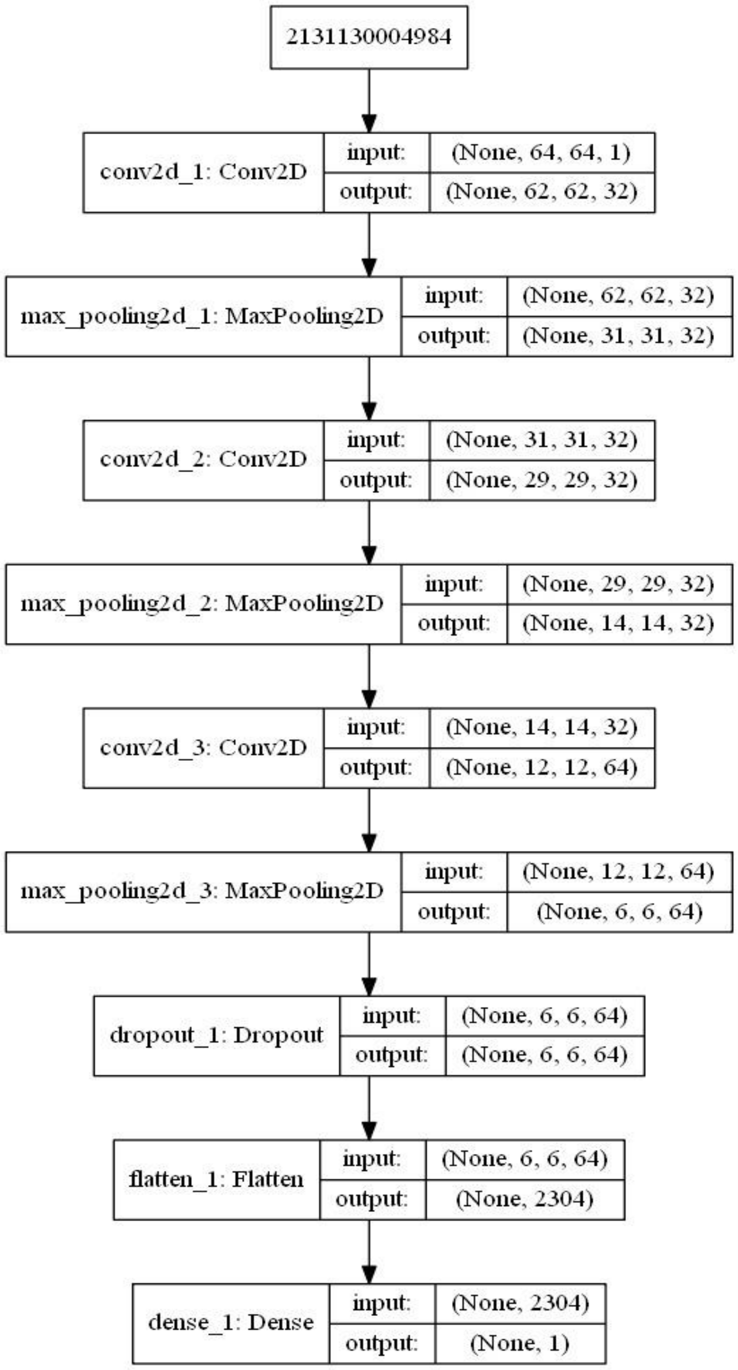
```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(64,64,1)))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Conv2D(32, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Conv2D(64, (3, 3),activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```


2131130007000
conv2d_34: Conv2D
input: (None, 64, 64, 1)
output: (None, 62, 62, 32)
max_pooling2d_34: MaxPooling2D
input: (None, 62, 62, 32)
output: (None, 31, 31, 32)
conv2d_35: Conv2D
input: (None, 31, 31, 32)
output: (None, 29, 29, 32)
max_pooling2d_35: MaxPooling2D
input: (None, 29, 29, 32)
output: (None, 14, 14, 32)
conv2d_36: Conv2D
input: (None, 14, 14, 32)
output: (None, 12, 12, 64)
max_pooling2d_36: MaxPooling2D
input: (None, 12, 12, 64)
output: (None, 6, 6, 64)
flatten_15: Flatten
input: (None, 6, 6, 64)
output: (None, 2304)
dense_16: Dense
input: (None, 2304)
output: (None, 1)

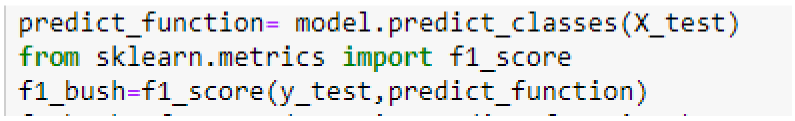
ii. For Williams



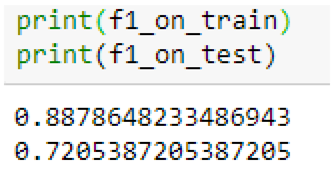
```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='tanh',input_shape=(64,64,1)))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Conv2D(32, (3, 3),activation='tanh'))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Conv2D(64, (3, 3),activation='tanh'))
model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
model.add(layers.Dropout(0.4))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```


2131130004984
conv2d_1: Conv2D | input: (None, 64, 64, 1) | output: (None, 62, 62, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 62, 62, 32) | output: (None, 31, 31, 32)
conv2d_2: Conv2D | input: (None, 31, 31, 32) | output: (None, 29, 29, 32)
max_pooling2d_2: MaxPooling2D | input: (None, 29, 29, 32) | output: (None, 14, 14, 32)
conv2d_3: Conv2D | input: (None, 14, 14, 32) | output: (None, 12, 12, 64)
max_pooling2d_3: MaxPooling2D | input: (None, 12, 12, 64) | output: (None, 6, 6, 64)
dropout_1: Dropout | input: (None, 6, 6, 64) | output: (None, 6, 6, 64)
flatten_1: Flatten | input: (None, 6, 6, 64) | output: (None, 2304)
dense_1: Dense | input: (None, 2304) | output: (None, 1)

- For the next step, we compute F1 score from sklearn for both Bush and Williams for train and test data

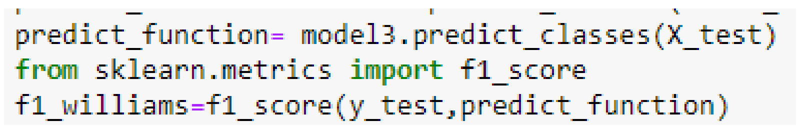


```
predict_function= model.predict_classes(X_test)
from sklearn.metrics import f1_score
f1_bush=f1_score(y_test,predict_function)
```

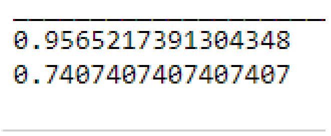


```
print(f1_on_train)
print(f1_on_test)

0.8878648233486943
0.7205387205387205
```



```
predict_function= model3.predict_classes(X_test)
from sklearn.metrics import f1_score
f1_williams=f1_score(y_test,predict_function)
```



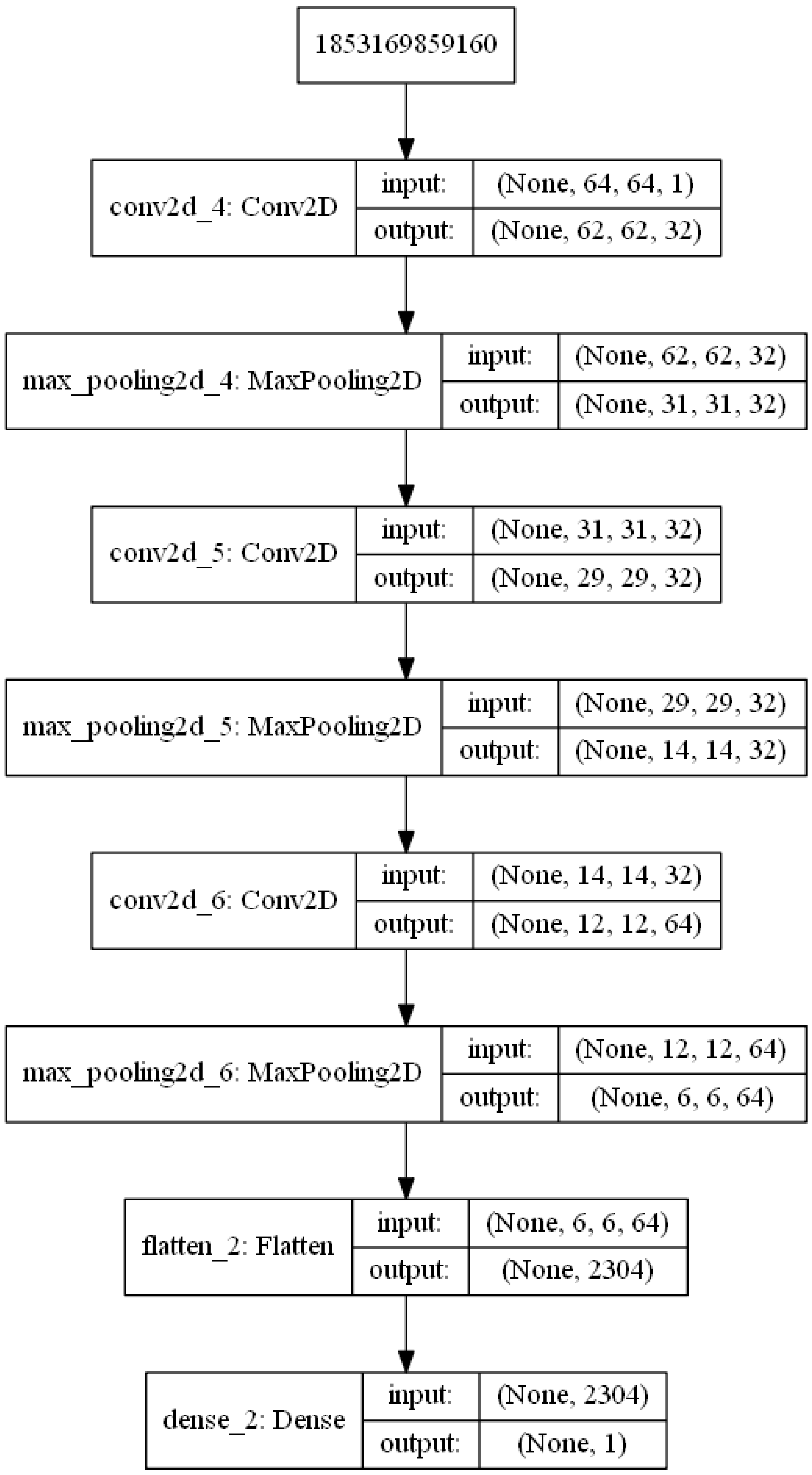
```
0.9565217391304348
0.7407407407407407
```

V. Project Phase 4

Description: Here we use Transfer Learning aspect to implement for this phase on image classification dataset and preprocessed it to be 64x64 size, gray-scale and have binary labels. It is similar to phase 3 but instead for starting with a randomly initialized model, we initialize the model on new dataset and load it and perform it again for Bush and Williams to get better results for F1

- The new dataset has specifications
  Name of dataset: Dogs vs Cats (https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data)
  Train folder: 25,000 images
  Test folder: 12,500 images
  Binary class labels (1=cat, 0=dog)
  It classifies that the image is of cat or not
- The data is preprocessed to 64x64 scale images to gray scale images and then deep learning model is developed for this data and then f1 score is computed
- The model created for new dataset can be given as below using the same method as phase3

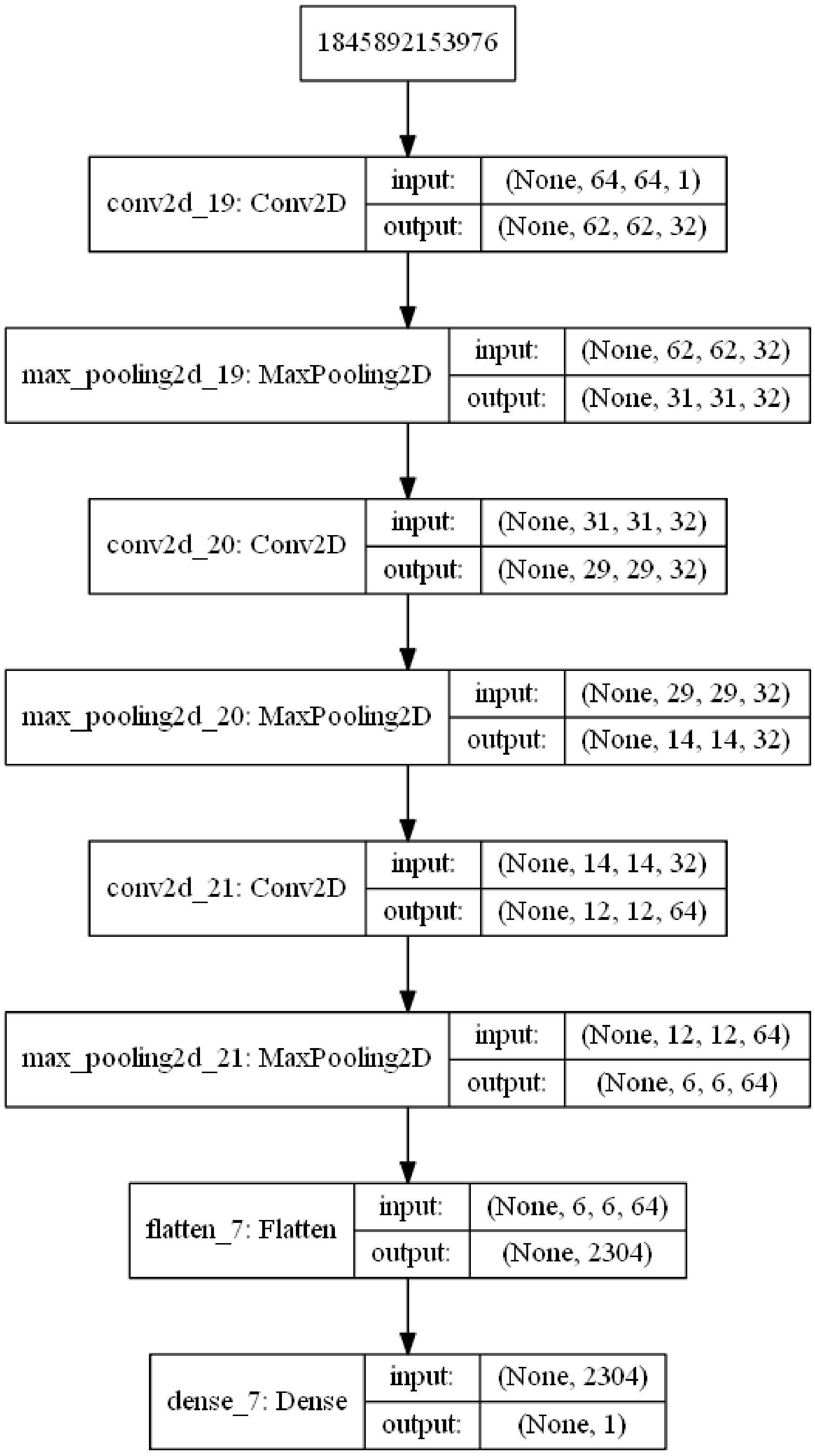
i. For original dataset



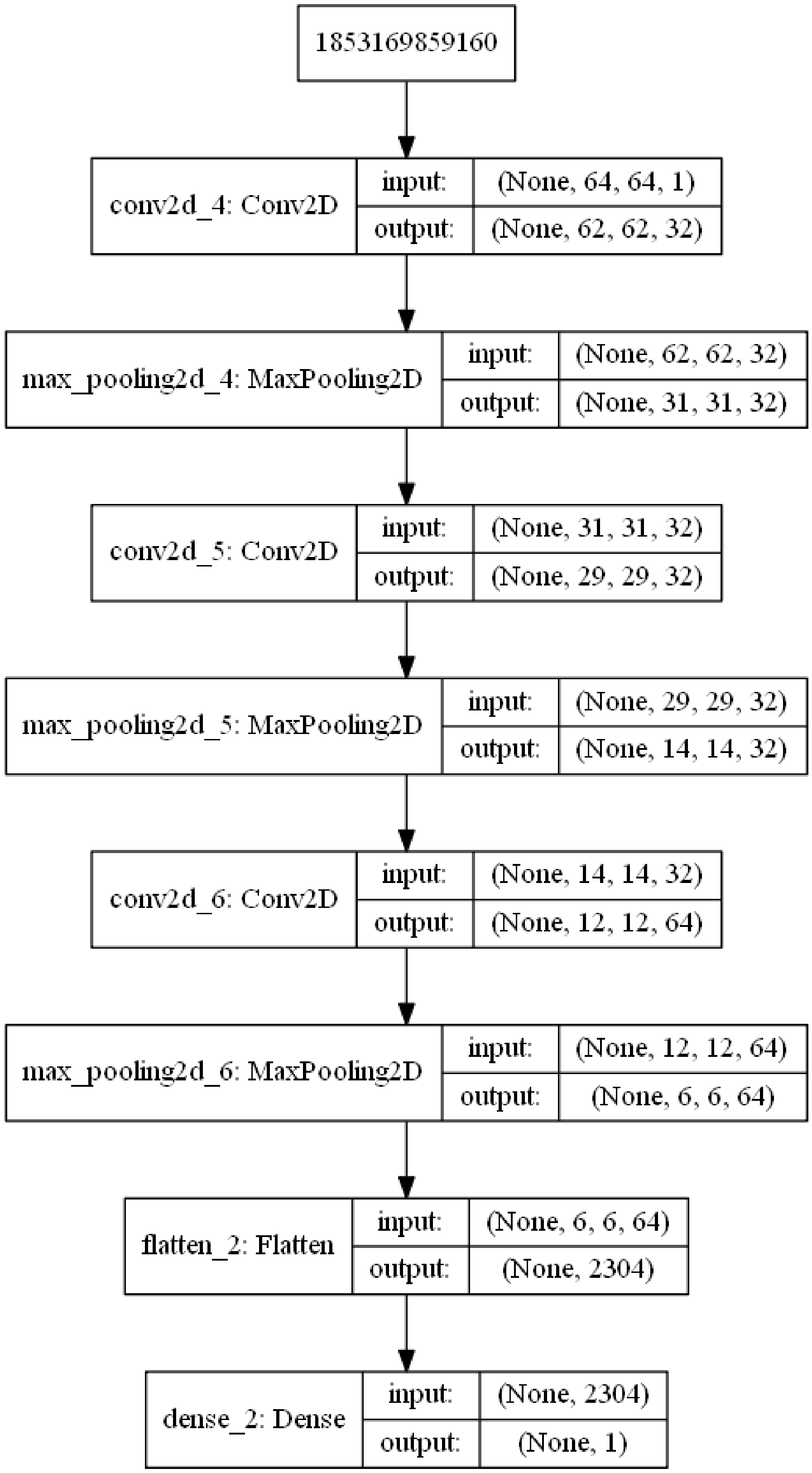
1853169859160
conv2d_4: Conv2D | input: (None, 64, 64, 1) | output: (None, 62, 62, 32)
max_pooling2d_4: MaxPooling2D | input: (None, 62, 62, 32) | output: (None, 31, 31, 32)
conv2d_5: Conv2D | input: (None, 31, 31, 32) | output: (None, 29, 29, 32)
max_pooling2d_5: MaxPooling2D | input: (None, 29, 29, 32) | output: (None, 14, 14, 32)
conv2d_6: Conv2D | input: (None, 14, 14, 32) | output: (None, 12, 12, 64)
max_pooling2d_6: MaxPooling2D | input: (None, 12, 12, 64) | output: (None, 6, 6, 64)
flatten_2: Flatten | input: (None, 6, 6, 64) | output: (None, 2304)
dense_2: Dense | input: (None, 2304) | output: (None, 1)

- This model is saved as “initialized-model.keras” and loaded again to train on either Bush or Williams to get F1 score for these datasets. We use the same train_test_split to split the dataset with random state as 8401. We could see that model architecture remains the same for Bush and Williams as for the original dataset. Thus, we transfer a model compiled, trained and fitted on new dataset and load to fit on of our dataset without compiling the model again

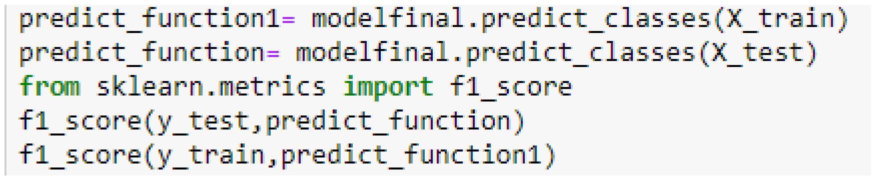
ii. For Bush



iii. **For Williams:**


1853169859160
conv2d_4: Conv2D
input: (None, 64, 64, 1)
output: (None, 62, 62, 32)
max_pooling2d_4: MaxPooling2D
input: (None, 62, 62, 32)
output: (None, 31, 31, 32)
conv2d_5: Conv2D
input: (None, 31, 31, 32)
output: (None, 29, 29, 32)
max_pooling2d_5: MaxPooling2D
input: (None, 29, 29, 32)
output: (None, 14, 14, 32)
conv2d_6: Conv2D
input: (None, 14, 14, 32)
output: (None, 12, 12, 64)
max_pooling2d_6: MaxPooling2D
input: (None, 12, 12, 64)
output: (None, 6, 6, 64)
flatten_2: Flatten
input: (None, 6, 6, 64)
output: (None, 2304)
dense_2: Dense
input: (None, 2304)
output: (None, 1)

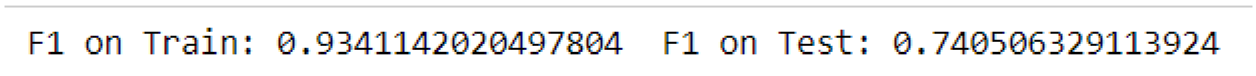
- We then perform the same steps as phase 3 that we fit the transformed model on dataset and predict classes for train and test data to compute the f1 score



```
predict_function1= modelfinal.predict_classes(X_train)
predict_function= modelfinal.predict_classes(X_test)
from sklearn.metrics import f1_score
f1_score(y_test,predict_function)
f1_score(y_train,predict_function1)
```

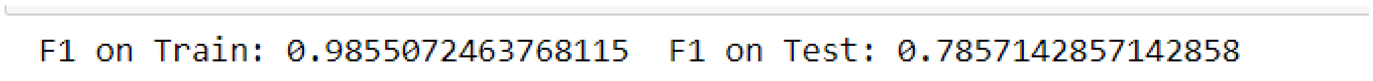
- We get better results for F1 in this phase:

i. **For Bush**



```
F1 on Train: 0.9341142020497804  F1 on Test: 0.740506329113924
```

ii. **For Williams**



```
F1 on Train: 0.9855072463768115  F1 on Test: 0.7857142857142858
```

Conclusion:

In this project we have performed KNN and SVC for original dataset and got results for F1 in phase 1 and for phase 2 we transformed the data using PCA and then got better F1 results. Also, the values obtained for KNN is less than values for the SVC for both phase which is because as the neighbours in KNN increase the distance increases and F1 thus obtained in less. For SVC, we can tune the parameters to obtain better results in mean F1 score.

In next two phases we perform, deep learning model execution where in phase 3 we train and fit on the out original dataset and obtain result F1 score but for phase 4 we train, compile and fit on new dataset and transfer it to our original dataset and it gives better F1 results for both Bush and williams. This is because the new data has higher number of data and thus a complex model is created to perform deep learning on the new data and get good results and so we get better results if we perform Transfer Learning