Import Libraries

In [1]:

```
#Importing libraries
from PIL import Image
import numpy as np
import sys
import os
import csv
from keras.models import Sequential
from sklearn import preprocessing
from keras.utils import np_utils
from keras.layers import Dense, Dropout, GaussianNoise, Conv1D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
```

Data Pre-Processing

In [2]:

```
# Converting all the images from JPG to CSV
filelist = []
for path in [x[0] for x in os.walk('D:/thesis_dataset/')][1:]:
    label = os.path.basename(path)
    for root, dirs, files in os.walk(path, topdown=False):
        for name in files:

            if name.endswith('.jpg'):
                fullName = os.path.join(root, name)
                filelist.append([fullName,label])
row = []
for file,label in filelist:

    print(file)
    img_file = Image.open(file)


    # get original image parameters...
    width, height = img_file.size
    format = img_file.format
    mode = img_file.mode

    # Make image Greyscale
    img_grey = img_file.convert('L')

    # Save Greyscale values
    value = np.asarray(img_grey.getdata(), dtype=np.int64).reshape((img_grey.size[1], img_g
    value = value.flatten()
    row.append([value,label])
```

```
D:/thesis_dataset/Abdoulaye_Wade\Abdoulaye_Wade_0001.jpg
D:/thesis_dataset/Abdoulaye_Wade\Abdoulaye_Wade_0002.jpg
D:/thesis_dataset/Abdoulaye_Wade\Abdoulaye_Wade_0003.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0001.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0002.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0003.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0004.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0005.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0006.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0007.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0009.jpg
D:/thesis_dataset/Adrien_Brody\Adrien_Brody_0011.jpg
D:/thesis_dataset/John_McCain\John_McCain_0001.jpg
D:/thesis_dataset/John_McCain\John_McCain_0002.jpg
D:/thesis_dataset/John_McCain\John_McCain_0005.jpg
D:/thesis_dataset/John_McCain\John_McCain_0006.jpg
D:/thesis_dataset/John_McCain\John_McCain_0007.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0001.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0003.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0004.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0005.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0006.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0007.jpg
D:/thesis_dataset/Paradorn_Srichaphan\Paradorn_Srichaphan_0008.jpg
D:/thesis_dataset/Unknown\Abdoulaye_Wade_0004.jpg
D:/thesis_dataset/Unknown\Adrien_Brody_0008.jpg
D:/thesis_dataset/Unknown\Adrien_Brody_0010.jpg
D:/thesis_dataset/Unknown\Adrien_Brody_0012.jpg
D:/thesis_dataset/Unknown\John_McCain_0003.jpg
```

```
D:/thesis_dataset/Unknown\John_McCain_0004.jpg
D:/thesis_dataset/Unknown\Paradorn_Srichaphan_0002.jpg
D:/thesis_dataset/Unknown\Paradorn_Srichaphan_0009.jpg
D:/thesis_dataset/Unknown\Paradorn_Srichaphan_0010.jpg
```

In [3]:

```
# Display the images in the dataframe
labels = []
arrays2 = []
for arr in row:
    labels.append(arr[1])
    arrays2.append(arr[0])
df = pd.DataFrame(arrays2)
df['label'] = labels
df.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	62491	62492	62493	62494	62495
0	214	214	213	213	213	214	215	215	218	218	...	53	59	58	54	6
1	45	47	48	48	49	50	51	51	52	55	...	34	34	34	34	34
2	1	1	1	1	1	0	0	0	0	0	...	45	1	0	0	1
3	0	0	0	0	0	0	0	0	7	7	...	50	53	53	52	52
4	108	105	99	93	87	79	69	61	54	50	...	47	49	51	56	60

5 rows × 62501 columns

In [4]:

```
df.tail(5)
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	62491	62492	62493	62494	62495	62496	62497	6249
28	0	0	0	0	0	0	0	0	0	0	...	6	6	5	5	6	6	4	
29	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
30	4	6	8	6	5	6	9	11	8	9	...	1	1	0	1	2	1	0	
31	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
32	0	0	0	0	0	3	6	6	7	7	...	0	0	0	0	0	0	0	

5 rows × 62501 columns

In [5]:

```
#Save the dataframe
df.to_csv('D:/thesis_dataset/ultimate_test.csv')
```

Exploratory Data Analysis

In [6]:

```
#Encode the label column
pixels = df.drop(["label"],axis=1)
label = df["label"]
le = preprocessing.LabelEncoder()
encoded_label = le.fit_transform(label)
```

In [7]:

```
le.classes_
```

Out[7]:

```
array(['Abdoulaye_Wade', 'Adrien_Brody', 'John_McCain',
       'Paradorn_Srichaphan', 'Unknown'], dtype=object)
```

In [8]:

```
print (pixels)
```

```
     0      1      2      3      4      5      6      7      8      9
...   \
0      214    214    213    213    213    214    215    215    218    218
...
1       45     47     48     48     49     50     51     51     52     55
...
2        1      1      1      1      1      0      0      0      0      0
...
3        0      0      0      0      0      0      0      0      7      7
...
4      108    105     99     93     87     79     69     61     54     50
...
5        0      0      0      0      0      0      1      1      0      0
...
6        1      1      0      0      0      0      1      1      1      1
...
7        1      1      1      0      0      0      0      0      0      0
...
8        0      0      0      0      0      0      0      0      0      0
...
9        0      0      0      0      0      0      0      0      0      3
...
10     242    242    242    242    243    243    243    243    242    242
...
11       0      0      0      0      0      0      0      0      1      0
...
12       1      1      1      1      1      1      1      1      0      0
...
13       0      0      0      0      0      0      0      0      0      0
...
14       0      0      0      0      0      0      0      0      0      3
...
15      36     36     34     32     30     28     25     23     25     28
...
16      89     88     87     87     88     88     88     89     89     89
...
17      70     72     74     74     73     72     71     69     65     62
...
18      43     42     42     42     42     41     42     43     44     43
...
19     137    136    136    136    136    136    136    136    136    136
...
20       2      0      0      1      0      0      0      0      0      2
...
21       1      1      2      1      1      1      1      1      1      1
...
22       1      1      1      1      1      0      0      0      0      1
...
23       1      1      1      1      1      1      1      1      1      1
...
24       1      0      1      0      1      1      1      1      1      1
...
25       0      0      0      0      0      0      0      0      0      0
...
26     222    222    222    222    221    221    221    221    221    221
...
27       6      5      5      4      4      4      7      8      8     11
```

```
...
28       0      0      0      0      0      0      0      0      0      0
...
29       0      0      0      0      0      0      0      0      0      0
...
30       4      6      8      6      5      6      9     11      8      9
...
31       0      0      0      0      0      0      0      0      0      0
...
32       0      0      0      0      0      3      6      6      7      7
...

    62490  62491  62492  62493  62494  62495  62496  62497  62498  62499
0      53     53     59     58     54      6      2      1      1      1
1      33     34     34     34     34     34      0      1      0      0
2      44     45      1      0      0      1      0      0      0      1
3      56     50     53     53     52     52      1      0      0      1
4      43     47     49     51     56     60     65     69     79     76
5      25     24     24     23     23     24     26     28     28     27
6       0      0      0      0      0      0      0      0      0      0
7      30     28      5      1      0      1      1      1      2      1
8     107    109    111    115    123    130    139    146    149    153
9       1      1      1      1      1      1      1      1      1      1
10     10     11     11     11     11     11     11     11     11     11
11     28     27     28     32     36     40     44     47     47     47
12     56     56     58     61     59     57     56     49     50     58
13     32      2      0      0      0      0      0      0      0      0
14      0      0      1      1      0      0      0      0      0      0
15     56     52     51     52     51     48     43     41     41     41
16     42     41     41     41     42     44     45     46     45     45
17     63     71     73     79     88     98    102     90     73     62
18    219    221    224    226    226    223    225    231    235    234
19      7      3      2      3      0      2      0      3      4      3
20      0      0      0      0      0      0      0      0      0      0
21      1      1      1      1      1      0      0      0      0      1
22     22     27     37     48     57     65     76     95    130    156
23    195    188    183    186    192    199    215    233    241    238
24     25     22     19     16     14     13     14     15     16     17
25     24     24     24     24     24     24     24     24     24     24
26     32     33     34     34     34     33     35      2      1      1
27    141    141    142    142    143    143    143    142    140    139
28      5      6      6      5      5      6      6      4      1      1
29      0      0      0      0      0      0      0      0      0      0
30      0      1      1      0      1      2      1      0      0      0
31      0      0      0      0      0      0      0      0      0      0
32      1      0      0      0      0      0      0      0      0      0

[33 rows x 62500 columns]
```

In [9]:

```
print (encoded_label)
```

```
[0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4]
```

In [10]:

```
#Function to display the images using the points in each column
def show_images(pixels):
    fig, axes = plt.subplots(5, 6, figsize=(11, 7), subplot_kw={'xticks':[], 'yticks':[]})
    for i, ax in enumerate(axes.flat):
        ax.imshow(np.array(pixels)[i].reshape(250, 250), cmap='gray')
    plt.show()
```

In [11]:

```
show_images(pixels)
```

Splitting the dataset

In [12]:

```
x_train, x_test, y_train, y_test = train_test_split(pixels, encoded_label)
```

In [13]:

```
x_train.shape
```

Out[13]:

```
(24, 62500)
```

In [14]:

```
x_test.shape
```

Out[14]:

```
(9, 62500)
```

In [15]:

```
y_train.shape
```

Out[15]:

```
(24,)
```

In [16]:

```
y_test.shape
```

Out[16]:

```
(9,)
```

Feature Extraction

PCA

In [17]:

```
#Standard Scaling
scaler = StandardScaler()
scaler.fit(x_train)
X_sc_train = scaler.transform(x_train)
X_sc_test = scaler.transform(x_test)
```

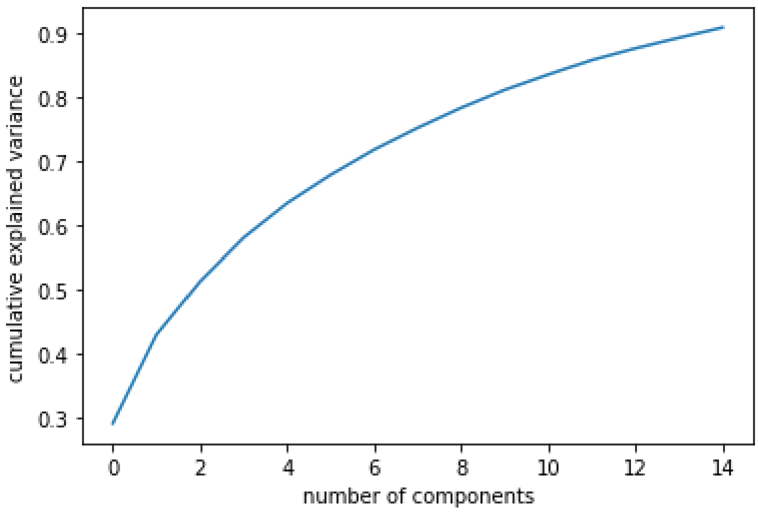
In [18]:

```
# Extracting feature using PCA
NCOMPONENTS = 15

pca = PCA(n_components=NCOMPONENTS)
X_pca_train = pca.fit_transform(X_sc_train)
X_pca_test = pca.transform(X_sc_test)
pca_std = np.std(X_pca_train)

print(X_sc_train.shape)
print(X_pca_train.shape)
print(y_train.shape)
Y_train = y_train.astype('int32')
Y_train = np_utils.to_categorical(Y_train)
print(Y_train)
```

```
(24, 62500)
(24, 15)
(24,)
[[0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]]
```

In [19]:

```
#pca = PCA(n_components=NCOMPONENTS, svd_solver='full').fit(x_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.show()
```

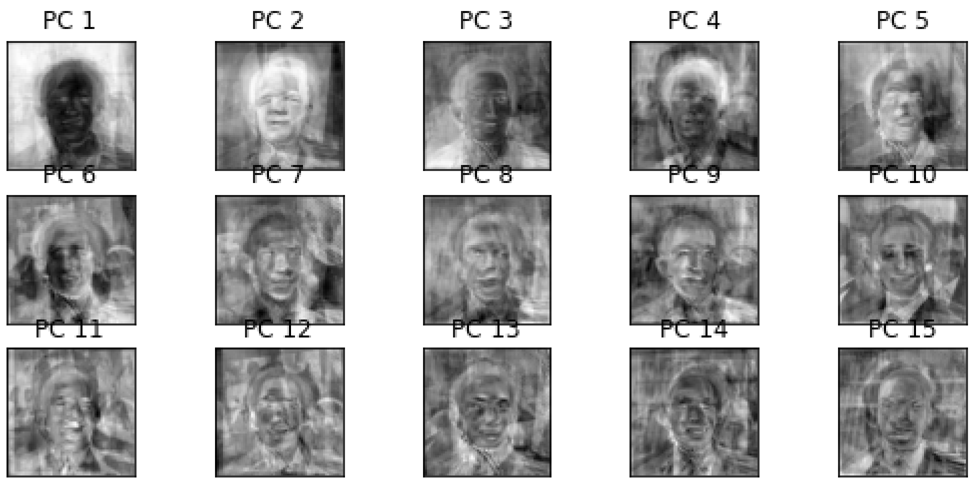


In [20]:

```
def show_eigenfaces(pca):
    fig, axes = plt.subplots(3, 5, figsize=(9, 4),subplot_kw={'xticks':[], 'yticks':[]})
    for i, ax in enumerate(axes.flat):
        ax.imshow(pca.components_[i].reshape(250, 250), cmap='gray')
        ax.set_title("PC " + str(i+1))
    plt.show()
```

In [21]:

```
show_eigenfaces(pca)
```



Fisherface

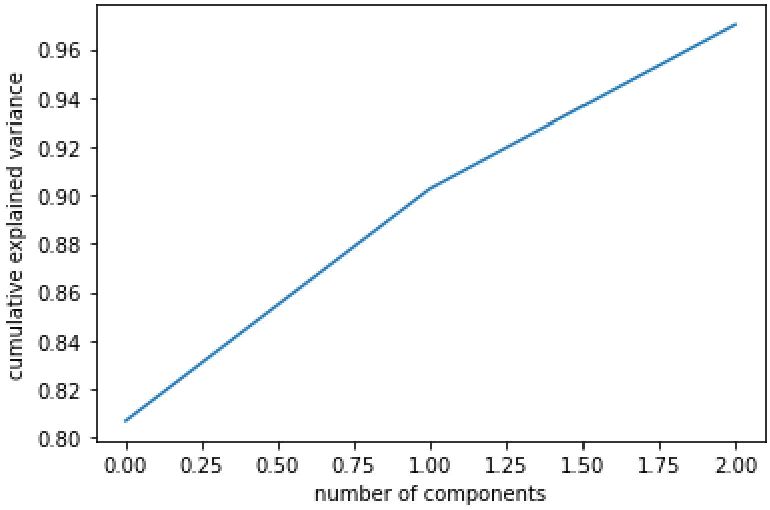
In [22]:

```
#Extracting feature using LDA
lda = LDA(n_components=3)

X_lda_train = lda.fit_transform(X_pca_train,y_train)
X_lda_test = lda.transform(X_pca_test)
lda_std = np.std(X_lda_train)
print(X_sc_train.shape)
print(X_lda_train.shape)
print(y_train.shape)
Y_train = y_train.astype('int32')
Y_train = np_utils.to_categorical(Y_train)
print(Y_train)
```

```
(24, 62500)
(24, 3)
(24,)
[[0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]]
```

In [23]:

```
plt.plot(np.cumsum(lda.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.show()
```



Model Training Using CNN

Training for PCA

In [24]:

```
model = Sequential()
layers = 1
units = 128
pca_std = np.std(X_pca_train)
#Dense layer with 128 neurons
model.add(Dense(units, input_dim=NCOMPONENTS, activation='relu'))
#regularisation layer
model.add(GaussianNoise(pca_std))

#We are adding only 1 set of layer with 128Neurons
for i in range(layers):
    #Dense layer
    model.add(Dense(units, activation='relu'))
    #Both below for regularisation
    model.add(GaussianNoise(pca_std))
    model.add(Dropout(0.1))

#Output layer ; 5 because there are only 5 labels, "softmax" is used because it has multipl
model.add(Dense(5, activation='softmax'))
model.summary()


model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['categorical_a

history = model.fit(X_pca_train, Y_train, epochs=100, batch_size=256, validation_split=0.15
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               2048
_________________________________________________________________
gaussian_noise (GaussianNois (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
=================================================================
Total params: 19,205
Trainable params: 19,205
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
1/1 - 0s - loss: 132.2403 - categorical_accuracy: 0.1500 - val_loss: 10.24
94 - val_categorical_accuracy: 0.5000
Epoch 2/100
1/1 - 0s - loss: 180.4118 - categorical_accuracy: 0.1000 - val_loss: 8.591
3 - val_categorical_accuracy: 0.5000
Epoch 3/100
1/1 - 0s - loss: 94.3574 - categorical_accuracy: 0.2000 - val_loss: 6.8288
- val_categorical_accuracy: 0.2500
Epoch 4/100
1/1 - 0s - loss: 86.2352 - categorical_accuracy: 0.2500 - val_loss: 7.0677
- val_categorical_accuracy: 0.2500
```

```
Epoch 5/100
1/1 - 0s - loss: 118.8483 - categorical_accuracy: 0.1500 - val_loss: 7.384
8 - val_categorical_accuracy: 0.0000e+00
Epoch 6/100
1/1 - 0s - loss: 102.1127 - categorical_accuracy: 0.1500 - val_loss: 7.584
4 - val_categorical_accuracy: 0.2500
Epoch 7/100
1/1 - 0s - loss: 96.1455 - categorical_accuracy: 0.3500 - val_loss: 8.6372
- val_categorical_accuracy: 0.2500
Epoch 8/100
1/1 - 0s - loss: 137.5462 - categorical_accuracy: 0.3000 - val_loss: 9.341
0 - val_categorical_accuracy: 0.2500
Epoch 9/100
1/1 - 0s - loss: 88.5713 - categorical_accuracy: 0.2500 - val_loss: 8.9367
- val_categorical_accuracy: 0.2500
Epoch 10/100
1/1 - 0s - loss: 94.1316 - categorical_accuracy: 0.3500 - val_loss: 10.419
2 - val_categorical_accuracy: 0.2500
Epoch 11/100
1/1 - 0s - loss: 83.3545 - categorical_accuracy: 0.3500 - val_loss: 11.817
5 - val_categorical_accuracy: 0.2500
Epoch 12/100
1/1 - 0s - loss: 103.4551 - categorical_accuracy: 0.2500 - val_loss: 12.69
79 - val_categorical_accuracy: 0.2500
Epoch 13/100
1/1 - 0s - loss: 120.5581 - categorical_accuracy: 0.0500 - val_loss: 12.53
40 - val_categorical_accuracy: 0.2500
Epoch 14/100
1/1 - 0s - loss: 84.3679 - categorical_accuracy: 0.1000 - val_loss: 12.827
7 - val_categorical_accuracy: 0.2500
Epoch 15/100
1/1 - 0s - loss: 81.3334 - categorical_accuracy: 0.4000 - val_loss: 13.556
7 - val_categorical_accuracy: 0.2500
Epoch 16/100
1/1 - 0s - loss: 87.7171 - categorical_accuracy: 0.3000 - val_loss: 14.197
7 - val_categorical_accuracy: 0.2500
Epoch 17/100
1/1 - 0s - loss: 94.2719 - categorical_accuracy: 0.4000 - val_loss: 14.004
9 - val_categorical_accuracy: 0.2500
Epoch 18/100
1/1 - 0s - loss: 78.3606 - categorical_accuracy: 0.3500 - val_loss: 14.372
2 - val_categorical_accuracy: 0.2500
Epoch 19/100
1/1 - 0s - loss: 55.0513 - categorical_accuracy: 0.2500 - val_loss: 13.390
4 - val_categorical_accuracy: 0.2500
Epoch 20/100
1/1 - 0s - loss: 79.1103 - categorical_accuracy: 0.2500 - val_loss: 14.672
5 - val_categorical_accuracy: 0.2500
Epoch 21/100
1/1 - 0s - loss: 97.6447 - categorical_accuracy: 0.4000 - val_loss: 15.328
4 - val_categorical_accuracy: 0.2500
Epoch 22/100
1/1 - 0s - loss: 62.1739 - categorical_accuracy: 0.4000 - val_loss: 14.463
1 - val_categorical_accuracy: 0.2500
Epoch 23/100
1/1 - 0s - loss: 101.4405 - categorical_accuracy: 0.4000 - val_loss: 14.93
14 - val_categorical_accuracy: 0.2500
Epoch 24/100
1/1 - 0s - loss: 84.7096 - categorical_accuracy: 0.4000 - val_loss: 14.993
5 - val_categorical_accuracy: 0.2500
Epoch 25/100
```

```
1/1 - 0s - loss: 74.6480 - categorical_accuracy: 0.3500 - val_loss: 14.493
6 - val_categorical_accuracy: 0.2500
Epoch 26/100
1/1 - 0s - loss: 97.6903 - categorical_accuracy: 0.3000 - val_loss: 14.515
1 - val_categorical_accuracy: 0.2500
Epoch 27/100
1/1 - 0s - loss: 66.4778 - categorical_accuracy: 0.2000 - val_loss: 14.573
8 - val_categorical_accuracy: 0.2500
Epoch 28/100
1/1 - 0s - loss: 95.8761 - categorical_accuracy: 0.3000 - val_loss: 14.160
9 - val_categorical_accuracy: 0.2500
Epoch 29/100
1/1 - 0s - loss: 81.0167 - categorical_accuracy: 0.2500 - val_loss: 15.326
0 - val_categorical_accuracy: 0.2500
Epoch 30/100
1/1 - 0s - loss: 67.0089 - categorical_accuracy: 0.4500 - val_loss: 15.173
9 - val_categorical_accuracy: 0.2500
Epoch 31/100
1/1 - 0s - loss: 97.7576 - categorical_accuracy: 0.2000 - val_loss: 15.733
8 - val_categorical_accuracy: 0.2500
Epoch 32/100
1/1 - 0s - loss: 58.3440 - categorical_accuracy: 0.4500 - val_loss: 16.051
5 - val_categorical_accuracy: 0.2500
Epoch 33/100
1/1 - 0s - loss: 82.9267 - categorical_accuracy: 0.2500 - val_loss: 15.649
8 - val_categorical_accuracy: 0.2500
Epoch 34/100
1/1 - 0s - loss: 77.4843 - categorical_accuracy: 0.2000 - val_loss: 15.363
2 - val_categorical_accuracy: 0.2500
Epoch 35/100
1/1 - 0s - loss: 78.9386 - categorical_accuracy: 0.4500 - val_loss: 16.136
6 - val_categorical_accuracy: 0.2500
Epoch 36/100
1/1 - 0s - loss: 72.6359 - categorical_accuracy: 0.3500 - val_loss: 17.237
8 - val_categorical_accuracy: 0.2500
Epoch 37/100
1/1 - 0s - loss: 49.4388 - categorical_accuracy: 0.4000 - val_loss: 17.716
2 - val_categorical_accuracy: 0.2500
Epoch 38/100
1/1 - 0s - loss: 76.7558 - categorical_accuracy: 0.3500 - val_loss: 18.796
6 - val_categorical_accuracy: 0.2500
Epoch 39/100
1/1 - 0s - loss: 106.1629 - categorical_accuracy: 0.3000 - val_loss: 17.06
52 - val_categorical_accuracy: 0.2500
Epoch 40/100
1/1 - 0s - loss: 62.4759 - categorical_accuracy: 0.5500 - val_loss: 17.723
6 - val_categorical_accuracy: 0.2500
Epoch 41/100
1/1 - 0s - loss: 82.6934 - categorical_accuracy: 0.3500 - val_loss: 18.108
0 - val_categorical_accuracy: 0.2500
Epoch 42/100
1/1 - 0s - loss: 85.4566 - categorical_accuracy: 0.3500 - val_loss: 17.509
9 - val_categorical_accuracy: 0.2500
Epoch 43/100
1/1 - 0s - loss: 48.1003 - categorical_accuracy: 0.5000 - val_loss: 18.426
7 - val_categorical_accuracy: 0.2500
Epoch 44/100
1/1 - 0s - loss: 56.2207 - categorical_accuracy: 0.5500 - val_loss: 18.173
0 - val_categorical_accuracy: 0.2500
Epoch 45/100
1/1 - 0s - loss: 86.6100 - categorical_accuracy: 0.5000 - val_loss: 18.655
```

```
6 - val_categorical_accuracy: 0.2500
Epoch 46/100
1/1 - 0s - loss: 55.5133 - categorical_accuracy: 0.3000 - val_loss: 18.797
8 - val_categorical_accuracy: 0.2500
Epoch 47/100
1/1 - 0s - loss: 86.3379 - categorical_accuracy: 0.2500 - val_loss: 19.224
3 - val_categorical_accuracy: 0.2500
Epoch 48/100
1/1 - 0s - loss: 67.8287 - categorical_accuracy: 0.3000 - val_loss: 18.653
3 - val_categorical_accuracy: 0.2500
Epoch 49/100
1/1 - 0s - loss: 61.9151 - categorical_accuracy: 0.4500 - val_loss: 19.994
7 - val_categorical_accuracy: 0.2500
Epoch 50/100
1/1 - 0s - loss: 89.0276 - categorical_accuracy: 0.3500 - val_loss: 20.389
5 - val_categorical_accuracy: 0.2500
Epoch 51/100
1/1 - 0s - loss: 47.9069 - categorical_accuracy: 0.5000 - val_loss: 20.113
9 - val_categorical_accuracy: 0.2500
Epoch 52/100
1/1 - 0s - loss: 85.7421 - categorical_accuracy: 0.3000 - val_loss: 20.896
5 - val_categorical_accuracy: 0.2500
Epoch 53/100
1/1 - 0s - loss: 81.5644 - categorical_accuracy: 0.4000 - val_loss: 22.154
1 - val_categorical_accuracy: 0.2500
Epoch 54/100
1/1 - 0s - loss: 103.8905 - categorical_accuracy: 0.3000 - val_loss: 21.57
57 - val_categorical_accuracy: 0.2500
Epoch 55/100
1/1 - 0s - loss: 63.2328 - categorical_accuracy: 0.5000 - val_loss: 21.824
7 - val_categorical_accuracy: 0.2500
Epoch 56/100
1/1 - 0s - loss: 60.9317 - categorical_accuracy: 0.4000 - val_loss: 21.948
6 - val_categorical_accuracy: 0.2500
Epoch 57/100
1/1 - 0s - loss: 59.8430 - categorical_accuracy: 0.3000 - val_loss: 22.212
0 - val_categorical_accuracy: 0.2500

Epoch 58/100
1/1 - 0s - loss: 50.7430 - categorical_accuracy: 0.5000 - val_loss: 22.428
2 - val_categorical_accuracy: 0.2500
Epoch 59/100
1/1 - 0s - loss: 44.2622 - categorical_accuracy: 0.5000 - val_loss: 22.758
9 - val_categorical_accuracy: 0.2500
Epoch 60/100
1/1 - 0s - loss: 38.0378 - categorical_accuracy: 0.3500 - val_loss: 20.827
6 - val_categorical_accuracy: 0.2500
Epoch 61/100
1/1 - 0s - loss: 79.7726 - categorical_accuracy: 0.4000 - val_loss: 20.644
6 - val_categorical_accuracy: 0.2500
Epoch 62/100
1/1 - 0s - loss: 49.4827 - categorical_accuracy: 0.4500 - val_loss: 20.506
9 - val_categorical_accuracy: 0.2500
Epoch 63/100
1/1 - 0s - loss: 60.3377 - categorical_accuracy: 0.4000 - val_loss: 20.548
6 - val_categorical_accuracy: 0.2500
Epoch 64/100
1/1 - 0s - loss: 74.1359 - categorical_accuracy: 0.3500 - val_loss: 19.563
7 - val_categorical_accuracy: 0.2500
Epoch 65/100
1/1 - 0s - loss: 48.2662 - categorical_accuracy: 0.5000 - val_loss: 18.377
```

```
4 - val_categorical_accuracy: 0.2500
Epoch 66/100
1/1 - 0s - loss: 58.5786 - categorical_accuracy: 0.3000 - val_loss: 17.809
2 - val_categorical_accuracy: 0.2500
Epoch 67/100
1/1 - 0s - loss: 81.5004 - categorical_accuracy: 0.4000 - val_loss: 17.190
5 - val_categorical_accuracy: 0.2500
Epoch 68/100
1/1 - 0s - loss: 64.1031 - categorical_accuracy: 0.4000 - val_loss: 16.992
3 - val_categorical_accuracy: 0.2500
Epoch 69/100
1/1 - 0s - loss: 39.6110 - categorical_accuracy: 0.4500 - val_loss: 16.374
4 - val_categorical_accuracy: 0.2500
Epoch 70/100
1/1 - 0s - loss: 60.4411 - categorical_accuracy: 0.4000 - val_loss: 16.416
4 - val_categorical_accuracy: 0.2500
Epoch 71/100
1/1 - 0s - loss: 58.5033 - categorical_accuracy: 0.5500 - val_loss: 16.519
4 - val_categorical_accuracy: 0.2500
Epoch 72/100
1/1 - 0s - loss: 49.1837 - categorical_accuracy: 0.5000 - val_loss: 17.052
5 - val_categorical_accuracy: 0.2500
Epoch 73/100
1/1 - 0s - loss: 44.2951 - categorical_accuracy: 0.5500 - val_loss: 17.352
1 - val_categorical_accuracy: 0.2500
Epoch 74/100
1/1 - 0s - loss: 54.6483 - categorical_accuracy: 0.5000 - val_loss: 18.510
2 - val_categorical_accuracy: 0.2500
Epoch 75/100
1/1 - 0s - loss: 60.9242 - categorical_accuracy: 0.3500 - val_loss: 18.356
2 - val_categorical_accuracy: 0.2500
Epoch 76/100
1/1 - 0s - loss: 71.5737 - categorical_accuracy: 0.3500 - val_loss: 17.643
0 - val_categorical_accuracy: 0.2500
Epoch 77/100
1/1 - 0s - loss: 87.6602 - categorical_accuracy: 0.3500 - val_loss: 17.625
1 - val_categorical_accuracy: 0.2500
Epoch 78/100
1/1 - 0s - loss: 40.4779 - categorical_accuracy: 0.6000 - val_loss: 17.744
2 - val_categorical_accuracy: 0.2500
Epoch 79/100
1/1 - 0s - loss: 25.3578 - categorical_accuracy: 0.4500 - val_loss: 17.219
8 - val_categorical_accuracy: 0.2500
Epoch 80/100
1/1 - 0s - loss: 82.0976 - categorical_accuracy: 0.4000 - val_loss: 15.955
1 - val_categorical_accuracy: 0.2500
Epoch 81/100
1/1 - 0s - loss: 31.2810 - categorical_accuracy: 0.4500 - val_loss: 16.251
7 - val_categorical_accuracy: 0.2500
Epoch 82/100
1/1 - 0s - loss: 36.3191 - categorical_accuracy: 0.5000 - val_loss: 17.343
5 - val_categorical_accuracy: 0.2500
Epoch 83/100
1/1 - 0s - loss: 70.0540 - categorical_accuracy: 0.2500 - val_loss: 16.930
2 - val_categorical_accuracy: 0.2500
Epoch 84/100
1/1 - 0s - loss: 49.8867 - categorical_accuracy: 0.5500 - val_loss: 17.067
2 - val_categorical_accuracy: 0.2500
Epoch 85/100
1/1 - 0s - loss: 46.2870 - categorical_accuracy: 0.6000 - val_loss: 17.965
5 - val_categorical_accuracy: 0.2500
```

```
Epoch 86/100
1/1 - 0s - loss: 61.4414 - categorical_accuracy: 0.4500 - val_loss: 18.229
1 - val_categorical_accuracy: 0.2500
Epoch 87/100
1/1 - 0s - loss: 23.7357 - categorical_accuracy: 0.6000 - val_loss: 19.973
1 - val_categorical_accuracy: 0.2500
Epoch 88/100
1/1 - 0s - loss: 40.5450 - categorical_accuracy: 0.4500 - val_loss: 21.065
8 - val_categorical_accuracy: 0.2500
Epoch 89/100
1/1 - 0s - loss: 69.8042 - categorical_accuracy: 0.4500 - val_loss: 22.825
3 - val_categorical_accuracy: 0.2500
Epoch 90/100
1/1 - 0s - loss: 31.9600 - categorical_accuracy: 0.6000 - val_loss: 22.498
3 - val_categorical_accuracy: 0.2500
Epoch 91/100
1/1 - 0s - loss: 33.3062 - categorical_accuracy: 0.6500 - val_loss: 23.482
8 - val_categorical_accuracy: 0.2500
Epoch 92/100
1/1 - 0s - loss: 24.0902 - categorical_accuracy: 0.5500 - val_loss: 22.206
6 - val_categorical_accuracy: 0.2500
Epoch 93/100
1/1 - 0s - loss: 38.0623 - categorical_accuracy: 0.4000 - val_loss: 21.438
0 - val_categorical_accuracy: 0.2500
Epoch 94/100
1/1 - 0s - loss: 41.7188 - categorical_accuracy: 0.6500 - val_loss: 21.547
5 - val_categorical_accuracy: 0.2500
Epoch 95/100
1/1 - 0s - loss: 36.6019 - categorical_accuracy: 0.6000 - val_loss: 21.341
8 - val_categorical_accuracy: 0.2500
Epoch 96/100
1/1 - 0s - loss: 42.7024 - categorical_accuracy: 0.5500 - val_loss: 21.101
1 - val_categorical_accuracy: 0.2500
Epoch 97/100
1/1 - 0s - loss: 40.1970 - categorical_accuracy: 0.5500 - val_loss: 20.668
6 - val_categorical_accuracy: 0.2500
Epoch 98/100
1/1 - 0s - loss: 63.6004 - categorical_accuracy: 0.4000 - val_loss: 20.883
6 - val_categorical_accuracy: 0.2500
Epoch 99/100
1/1 - 0s - loss: 61.2708 - categorical_accuracy: 0.5000 - val_loss: 21.460
0 - val_categorical_accuracy: 0.2500
Epoch 100/100
1/1 - 0s - loss: 25.0615 - categorical_accuracy: 0.6000 - val_loss: 20.776
2 - val_categorical_accuracy: 0.2500
```

Prediction using PCA with CNN

In [25]:

```
predictions = model.predict_classes(X_pca_test, verbose=0)
predictions1 = le.inverse_transform(predictions)
print(classification_report(y_test,predictions))

def write_predictions(predictions1, fname):
    pd.DataFrame({"ImageId": list(range(1,len(predictions1)+1)), "Label": predictions1}).to

write_predictions(predictions1, "pca-keras-mlp.csv")
```

```
WARNING:tensorflow:From <ipython-input-25-314914ef6b95>:1: Sequential.predic
t_classes (from tensorflow.python.keras.engine.sequential) is deprecated and
will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer ac
tivation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.40      1.00      0.57         2
           3       0.50      1.00      0.67         2
           4       0.00      0.00      0.00         4

    accuracy                           0.44         9
   macro avg       0.23      0.50      0.31         9
weighted avg       0.20      0.44      0.28         9


C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Training for FisherFace

In [26]:

```
model = Sequential()
layers = 1
units = 128
lda_std = np.std(X_lda_train)
model.add(Dense(units, input_dim=3, activation='relu'))
model.add(GaussianNoise(lda_std))
for i in range(layers):
    model.add(Dense(units, activation='relu'))
    model.add(GaussianNoise(lda_std))
    model.add(Dropout(0.1))
model.add(Dense(5, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['categorical_a
history = model.fit(X_lda_train, Y_train, epochs=100, batch_size=256, validation_split=0.15
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 128)               512       
_________________________________________________________________
gaussian_noise_2 (GaussianNo (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512     
_________________________________________________________________
gaussian_noise_3 (GaussianNo (None, 128)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 645       
=================================================================
Total params: 17,669
Trainable params: 17,669
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
1/1 - 0s - loss: 5.3841 - categorical_accuracy: 0.2500 - val_loss: 1.7232 -
val_categorical_accuracy: 0.0000e+00
Epoch 2/100
1/1 - 0s - loss: 10.0663 - categorical_accuracy: 0.2000 - val_loss: 1.6891 -
val_categorical_accuracy: 0.2500
Epoch 3/100
1/1 - 0s - loss: 6.4097 - categorical_accuracy: 0.3000 - val_loss: 1.6831 -
val_categorical_accuracy: 0.2500
Epoch 4/100
1/1 - 0s - loss: 7.5133 - categorical_accuracy: 0.1500 - val_loss: 1.6610 -
val_categorical_accuracy: 0.2500
Epoch 5/100
1/1 - 0s - loss: 8.9568 - categorical_accuracy: 0.1000 - val_loss: 1.6422 -
val_categorical_accuracy: 0.2500
Epoch 6/100
1/1 - 0s - loss: 6.5044 - categorical_accuracy: 0.2000 - val_loss: 1.6179 -
val_categorical_accuracy: 0.2500
Epoch 7/100
1/1 - 0s - loss: 5.1359 - categorical_accuracy: 0.2000 - val_loss: 1.5906 -
val_categorical_accuracy: 0.2500
Epoch 8/100
```

```
1/1 - 0s - loss: 3.0878 - categorical_accuracy: 0.4500 - val_loss: 1.5877 -
val_categorical_accuracy: 0.2500
Epoch 9/100
1/1 - 0s - loss: 7.7606 - categorical_accuracy: 0.0000e+00 - val_loss: 1.567
5 - val_categorical_accuracy: 0.2500
Epoch 10/100
1/1 - 0s - loss: 6.0047 - categorical_accuracy: 0.2000 - val_loss: 1.5504 -
val_categorical_accuracy: 0.2500
Epoch 11/100
1/1 - 0s - loss: 6.4555 - categorical_accuracy: 0.1000 - val_loss: 1.5351 -
val_categorical_accuracy: 0.2500
Epoch 12/100
1/1 - 0s - loss: 8.3776 - categorical_accuracy: 0.1500 - val_loss: 1.5248 -
val_categorical_accuracy: 0.2500
Epoch 13/100
1/1 - 0s - loss: 5.6271 - categorical_accuracy: 0.4000 - val_loss: 1.5188 -
val_categorical_accuracy: 0.2500
Epoch 14/100
1/1 - 0s - loss: 4.9292 - categorical_accuracy: 0.2000 - val_loss: 1.4958 -
val_categorical_accuracy: 0.2500
Epoch 15/100
1/1 - 0s - loss: 4.4232 - categorical_accuracy: 0.5000 - val_loss: 1.4938 -
val_categorical_accuracy: 0.5000
Epoch 16/100
1/1 - 0s - loss: 5.0297 - categorical_accuracy: 0.2500 - val_loss: 1.4847 -
val_categorical_accuracy: 0.5000
Epoch 17/100
1/1 - 0s - loss: 5.1295 - categorical_accuracy: 0.3500 - val_loss: 1.4774 -
val_categorical_accuracy: 0.5000
Epoch 18/100
1/1 - 0s - loss: 6.4431 - categorical_accuracy: 0.2500 - val_loss: 1.4676 -
val_categorical_accuracy: 0.5000
Epoch 19/100
1/1 - 0s - loss: 5.6839 - categorical_accuracy: 0.3000 - val_loss: 1.4554 -
val_categorical_accuracy: 0.5000
Epoch 20/100
1/1 - 0s - loss: 9.9393 - categorical_accuracy: 0.1000 - val_loss: 1.4497 -
val_categorical_accuracy: 0.5000
Epoch 21/100
1/1 - 0s - loss: 5.4627 - categorical_accuracy: 0.3500 - val_loss: 1.4360 -
val_categorical_accuracy: 0.5000
Epoch 22/100
1/1 - 0s - loss: 5.7367 - categorical_accuracy: 0.2000 - val_loss: 1.4284 -
val_categorical_accuracy: 0.5000
Epoch 23/100
1/1 - 0s - loss: 5.5326 - categorical_accuracy: 0.3500 - val_loss: 1.4133 -
val_categorical_accuracy: 0.5000
Epoch 24/100
1/1 - 0s - loss: 4.7908 - categorical_accuracy: 0.3000 - val_loss: 1.4003 -
val_categorical_accuracy: 0.5000
Epoch 25/100
1/1 - 0s - loss: 6.9866 - categorical_accuracy: 0.3000 - val_loss: 1.3956 -
val_categorical_accuracy: 0.5000
Epoch 26/100
1/1 - 0s - loss: 6.4174 - categorical_accuracy: 0.2500 - val_loss: 1.3897 -
val_categorical_accuracy: 0.5000
Epoch 27/100
1/1 - 0s - loss: 6.8416 - categorical_accuracy: 0.3000 - val_loss: 1.3778 -
val_categorical_accuracy: 0.5000
Epoch 28/100
1/1 - 0s - loss: 4.0708 - categorical_accuracy: 0.3000 - val_loss: 1.3687 -
```

```
val_categorical_accuracy: 0.5000
Epoch 29/100
1/1 - 0s - loss: 4.7055 - categorical_accuracy: 0.3000 - val_loss: 1.3645 -
val_categorical_accuracy: 0.5000
Epoch 30/100
1/1 - 0s - loss: 5.2581 - categorical_accuracy: 0.3000 - val_loss: 1.3521 -
val_categorical_accuracy: 0.5000
Epoch 31/100
1/1 - 0s - loss: 5.2629 - categorical_accuracy: 0.3000 - val_loss: 1.3345 -
val_categorical_accuracy: 0.5000
Epoch 32/100
1/1 - 0s - loss: 3.8200 - categorical_accuracy: 0.3000 - val_loss: 1.3248 -
val_categorical_accuracy: 0.5000
Epoch 33/100
1/1 - 0s - loss: 5.2988 - categorical_accuracy: 0.2500 - val_loss: 1.3130 -
val_categorical_accuracy: 0.5000
Epoch 34/100
1/1 - 0s - loss: 5.9176 - categorical_accuracy: 0.1000 - val_loss: 1.3054 -
val_categorical_accuracy: 0.5000
Epoch 35/100
1/1 - 0s - loss: 4.4180 - categorical_accuracy: 0.3500 - val_loss: 1.2993 -
val_categorical_accuracy: 0.5000
Epoch 36/100
1/1 - 0s - loss: 4.6414 - categorical_accuracy: 0.2500 - val_loss: 1.2805 -
val_categorical_accuracy: 0.5000
Epoch 37/100
1/1 - 0s - loss: 5.2168 - categorical_accuracy: 0.5000 - val_loss: 1.2726 -
val_categorical_accuracy: 0.5000
Epoch 38/100
1/1 - 0s - loss: 2.8863 - categorical_accuracy: 0.5500 - val_loss: 1.2638 -
val_categorical_accuracy: 0.7500
Epoch 39/100
1/1 - 0s - loss: 4.7412 - categorical_accuracy: 0.4000 - val_loss: 1.2522 -
val_categorical_accuracy: 0.7500
Epoch 40/100
1/1 - 0s - loss: 6.0428 - categorical_accuracy: 0.2000 - val_loss: 1.2447 -
val_categorical_accuracy: 0.7500
Epoch 41/100
1/1 - 0s - loss: 4.0016 - categorical_accuracy: 0.3000 - val_loss: 1.2373 -
val_categorical_accuracy: 0.5000
Epoch 42/100
1/1 - 0s - loss: 6.5319 - categorical_accuracy: 0.2500 - val_loss: 1.2332 -
val_categorical_accuracy: 0.5000
Epoch 43/100
1/1 - 0s - loss: 3.8340 - categorical_accuracy: 0.3500 - val_loss: 1.2311 -
val_categorical_accuracy: 0.5000
Epoch 44/100
1/1 - 0s - loss: 5.5920 - categorical_accuracy: 0.3500 - val_loss: 1.2195 -
val_categorical_accuracy: 0.5000
Epoch 45/100
1/1 - 0s - loss: 6.8957 - categorical_accuracy: 0.2500 - val_loss: 1.2134 -
val_categorical_accuracy: 0.5000
Epoch 46/100
1/1 - 0s - loss: 2.9809 - categorical_accuracy: 0.3500 - val_loss: 1.2212 -
val_categorical_accuracy: 0.5000
Epoch 47/100
1/1 - 0s - loss: 3.1482 - categorical_accuracy: 0.4500 - val_loss: 1.2226 -
val_categorical_accuracy: 0.5000
Epoch 48/100
1/1 - 0s - loss: 4.9945 - categorical_accuracy: 0.2500 - val_loss: 1.2153 -
val_categorical_accuracy: 0.5000
```

```
Epoch 49/100
1/1 - 0s - loss: 5.1230 - categorical_accuracy: 0.3000 - val_loss: 1.2084 -
val_categorical_accuracy: 0.5000
Epoch 50/100
1/1 - 0s - loss: 3.9845 - categorical_accuracy: 0.4000 - val_loss: 1.2143 -
val_categorical_accuracy: 0.5000
Epoch 51/100
1/1 - 0s - loss: 4.7045 - categorical_accuracy: 0.3000 - val_loss: 1.2105 -
val_categorical_accuracy: 0.5000
Epoch 52/100
1/1 - 0s - loss: 5.4466 - categorical_accuracy: 0.2000 - val_loss: 1.2009 -
val_categorical_accuracy: 0.7500
Epoch 53/100
1/1 - 0s - loss: 6.4719 - categorical_accuracy: 0.3500 - val_loss: 1.1887 -
val_categorical_accuracy: 0.7500
Epoch 54/100
1/1 - 0s - loss: 2.7615 - categorical_accuracy: 0.4500 - val_loss: 1.1829 -
val_categorical_accuracy: 0.7500
Epoch 55/100
1/1 - 0s - loss: 5.2828 - categorical_accuracy: 0.3000 - val_loss: 1.1729 -
val_categorical_accuracy: 0.7500
Epoch 56/100
1/1 - 0s - loss: 3.8435 - categorical_accuracy: 0.5000 - val_loss: 1.1586 -
val_categorical_accuracy: 0.7500
Epoch 57/100
1/1 - 0s - loss: 4.2200 - categorical_accuracy: 0.2500 - val_loss: 1.1594 -
val_categorical_accuracy: 0.7500
Epoch 58/100
1/1 - 0s - loss: 6.4572 - categorical_accuracy: 0.3500 - val_loss: 1.1597 -
val_categorical_accuracy: 0.7500


Epoch 59/100
1/1 - 0s - loss: 4.4949 - categorical_accuracy: 0.3000 - val_loss: 1.1463 -
val_categorical_accuracy: 0.7500
Epoch 60/100
1/1 - 0s - loss: 5.4255 - categorical_accuracy: 0.3000 - val_loss: 1.1489 -
val_categorical_accuracy: 0.7500
Epoch 61/100
1/1 - 0s - loss: 4.9741 - categorical_accuracy: 0.3000 - val_loss: 1.1413 -
val_categorical_accuracy: 0.7500
Epoch 62/100
1/1 - 0s - loss: 3.5989 - categorical_accuracy: 0.4000 - val_loss: 1.1310 -
val_categorical_accuracy: 0.7500
Epoch 63/100
1/1 - 0s - loss: 4.7183 - categorical_accuracy: 0.3000 - val_loss: 1.1314 -
val_categorical_accuracy: 0.7500
Epoch 64/100
1/1 - 0s - loss: 7.1114 - categorical_accuracy: 0.3500 - val_loss: 1.1263 -
val_categorical_accuracy: 0.7500
Epoch 65/100
1/1 - 0s - loss: 4.2826 - categorical_accuracy: 0.4500 - val_loss: 1.1222 -
val_categorical_accuracy: 0.7500
Epoch 66/100
1/1 - 0s - loss: 5.7943 - categorical_accuracy: 0.3000 - val_loss: 1.1135 -
val_categorical_accuracy: 0.7500
Epoch 67/100
1/1 - 0s - loss: 4.8158 - categorical_accuracy: 0.5000 - val_loss: 1.1007 -
val_categorical_accuracy: 0.7500
Epoch 68/100
1/1 - 0s - loss: 3.3951 - categorical_accuracy: 0.4500 - val_loss: 1.1018 -
```

```
val_categorical_accuracy: 0.7500
Epoch 69/100
1/1 - 0s - loss: 3.0798 - categorical_accuracy: 0.5500 - val_loss: 1.0926 -
val_categorical_accuracy: 0.7500
Epoch 70/100
1/1 - 0s - loss: 3.9432 - categorical_accuracy: 0.5000 - val_loss: 1.0784 -
val_categorical_accuracy: 0.7500
Epoch 71/100
1/1 - 0s - loss: 3.9183 - categorical_accuracy: 0.4000 - val_loss: 1.0768 -
val_categorical_accuracy: 0.7500
Epoch 72/100
1/1 - 0s - loss: 2.9723 - categorical_accuracy: 0.4500 - val_loss: 1.0674 -
val_categorical_accuracy: 0.7500
Epoch 73/100
1/1 - 0s - loss: 5.5061 - categorical_accuracy: 0.4500 - val_loss: 1.0649 -
val_categorical_accuracy: 0.7500
Epoch 74/100
1/1 - 0s - loss: 2.0273 - categorical_accuracy: 0.5500 - val_loss: 1.0512 -
val_categorical_accuracy: 0.7500
Epoch 75/100
1/1 - 0s - loss: 4.1350 - categorical_accuracy: 0.4000 - val_loss: 1.0458 -
val_categorical_accuracy: 0.7500
Epoch 76/100
1/1 - 0s - loss: 5.0646 - categorical_accuracy: 0.1500 - val_loss: 1.0356 -
val_categorical_accuracy: 0.7500
Epoch 77/100
1/1 - 0s - loss: 3.1082 - categorical_accuracy: 0.4000 - val_loss: 1.0392 -
val_categorical_accuracy: 0.7500
Epoch 78/100
1/1 - 0s - loss: 4.3220 - categorical_accuracy: 0.3000 - val_loss: 1.0240 -
val_categorical_accuracy: 0.7500
Epoch 79/100
1/1 - 0s - loss: 2.9567 - categorical_accuracy: 0.4000 - val_loss: 1.0220 -
val_categorical_accuracy: 0.7500
Epoch 80/100
1/1 - 0s - loss: 3.7043 - categorical_accuracy: 0.4000 - val_loss: 1.0070 -
val_categorical_accuracy: 0.7500
Epoch 81/100
1/1 - 0s - loss: 3.5842 - categorical_accuracy: 0.4000 - val_loss: 1.0060 -
val_categorical_accuracy: 0.7500
Epoch 82/100
1/1 - 0s - loss: 2.6388 - categorical_accuracy: 0.5000 - val_loss: 1.0083 -
val_categorical_accuracy: 0.7500
Epoch 83/100
1/1 - 0s - loss: 4.6578 - categorical_accuracy: 0.4500 - val_loss: 1.0084 -
val_categorical_accuracy: 0.7500
Epoch 84/100
1/1 - 0s - loss: 6.0589 - categorical_accuracy: 0.1500 - val_loss: 1.0014 -
val_categorical_accuracy: 0.7500
Epoch 85/100
1/1 - 0s - loss: 4.2128 - categorical_accuracy: 0.5500 - val_loss: 1.0015 -
val_categorical_accuracy: 0.7500
Epoch 86/100
1/1 - 0s - loss: 4.6290 - categorical_accuracy: 0.4500 - val_loss: 0.9982 -
val_categorical_accuracy: 0.7500
Epoch 87/100
1/1 - 0s - loss: 5.0515 - categorical_accuracy: 0.4500 - val_loss: 0.9959 -
val_categorical_accuracy: 0.7500
Epoch 88/100
1/1 - 0s - loss: 4.2153 - categorical_accuracy: 0.4500 - val_loss: 0.9858 -
val_categorical_accuracy: 0.7500
```

```
Epoch 89/100
1/1 - 0s - loss: 4.6492 - categorical_accuracy: 0.4500 - val_loss: 1.0009 -
val_categorical_accuracy: 0.7500
Epoch 90/100
1/1 - 0s - loss: 5.1254 - categorical_accuracy: 0.3000 - val_loss: 1.0048 -
val_categorical_accuracy: 0.7500
Epoch 91/100
1/1 - 0s - loss: 4.6039 - categorical_accuracy: 0.3000 - val_loss: 1.0053 -
val_categorical_accuracy: 0.7500
Epoch 92/100
1/1 - 0s - loss: 3.6217 - categorical_accuracy: 0.4500 - val_loss: 1.0134 -
val_categorical_accuracy: 0.7500
Epoch 93/100
1/1 - 0s - loss: 4.5866 - categorical_accuracy: 0.3500 - val_loss: 1.0030 -
val_categorical_accuracy: 0.7500
Epoch 94/100
1/1 - 0s - loss: 5.3963 - categorical_accuracy: 0.3000 - val_loss: 0.9848 -
val_categorical_accuracy: 0.7500
Epoch 95/100
1/1 - 0s - loss: 4.1662 - categorical_accuracy: 0.4000 - val_loss: 0.9745 -
val_categorical_accuracy: 0.7500
Epoch 96/100
1/1 - 0s - loss: 3.0889 - categorical_accuracy: 0.4500 - val_loss: 0.9678 -
val_categorical_accuracy: 0.7500
Epoch 97/100
1/1 - 0s - loss: 5.8430 - categorical_accuracy: 0.3500 - val_loss: 0.9700 -
val_categorical_accuracy: 0.7500
Epoch 98/100
1/1 - 0s - loss: 4.0007 - categorical_accuracy: 0.5500 - val_loss: 0.9724 -
val_categorical_accuracy: 0.7500
Epoch 99/100
1/1 - 0s - loss: 4.3422 - categorical_accuracy: 0.4500 - val_loss: 0.9634 -
val_categorical_accuracy: 0.7500
Epoch 100/100
1/1 - 0s - loss: 3.4452 - categorical_accuracy: 0.5000 - val_loss: 0.9532 -
val_categorical_accuracy: 0.7500
```

Prediction using Fisherface using CNN

In [27]:

```
predictions = model.predict_classes(X_lda_test, verbose=0)
predictions1 = le.inverse_transform(predictions)
print(classification_report(y_test,predictions))

def write_predictions(predictions1, fname):
    pd.DataFrame({"ImageId": list(range(1,len(predictions1)+1)), "Label": predictions1}).to

write_predictions(predictions1, "lda-keras-mlp.csv")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       0.00      0.00      0.00         2
           2       0.00      0.00      0.00         0
           3       0.29      1.00      0.44         2
           4       0.00      0.00      0.00         4

    accuracy                           0.33         9
   macro avg       0.26      0.40      0.29         9
weighted avg       0.17      0.33      0.21         9


C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being s
et to 0.0 in labels with no true samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being s
et to 0.0 in labels with no true samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\MuZ\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being s
et to 0.0 in labels with no true samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```