In [8]:

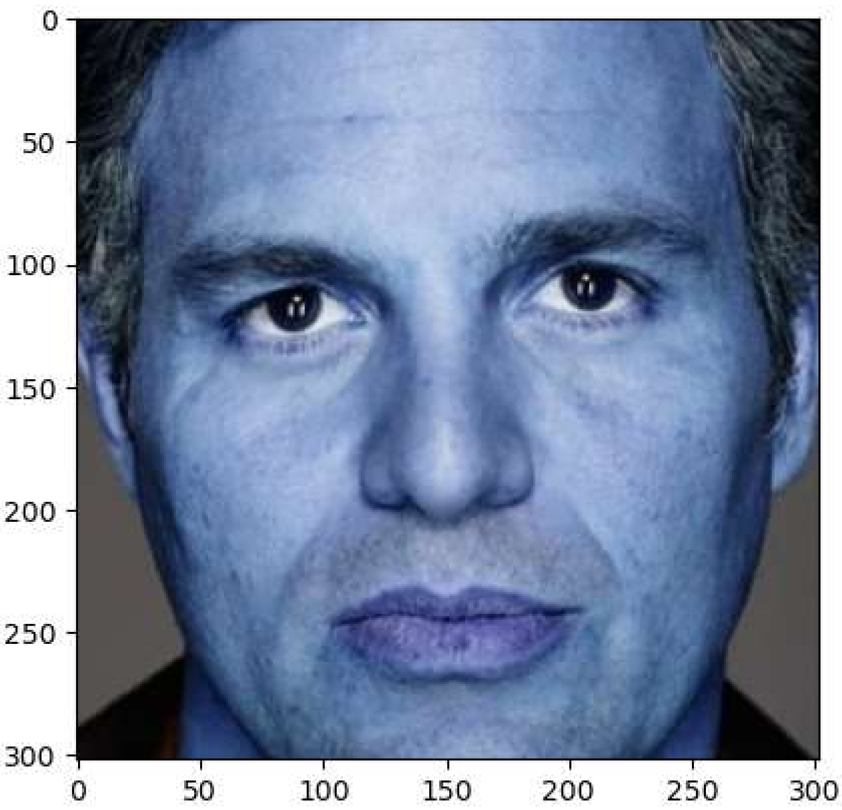
```
import numpy as np
import pandas as pd
import cv2
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

In [48]:

```
link=r"C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled Folder\Dataset
img = cv2.imread(link)
plt.imshow(img)
```

Out[48]:

```
<matplotlib.image.AxesImage at 0x1b7b1cf1fc0>
```



In [31]:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray.shape
```

Out[31]:

```
(302, 302)
```

In [40]:

```
face_cascade = cv2.CascadeClassifier(r"C:\Users\rohit\OneDrive\B Tech\3.2\
eye_cascade = cv2.CascadeClassifier(r"C:\Users\rohit\OneDrive\B Tech\3.2\P

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
faces
```

Out[40]:

```
array([[ 16,  10, 255, 255]])
```

In [41]:

```
(x,y,w,h) = faces[0]
x,y,w,h
```

Out[41]:

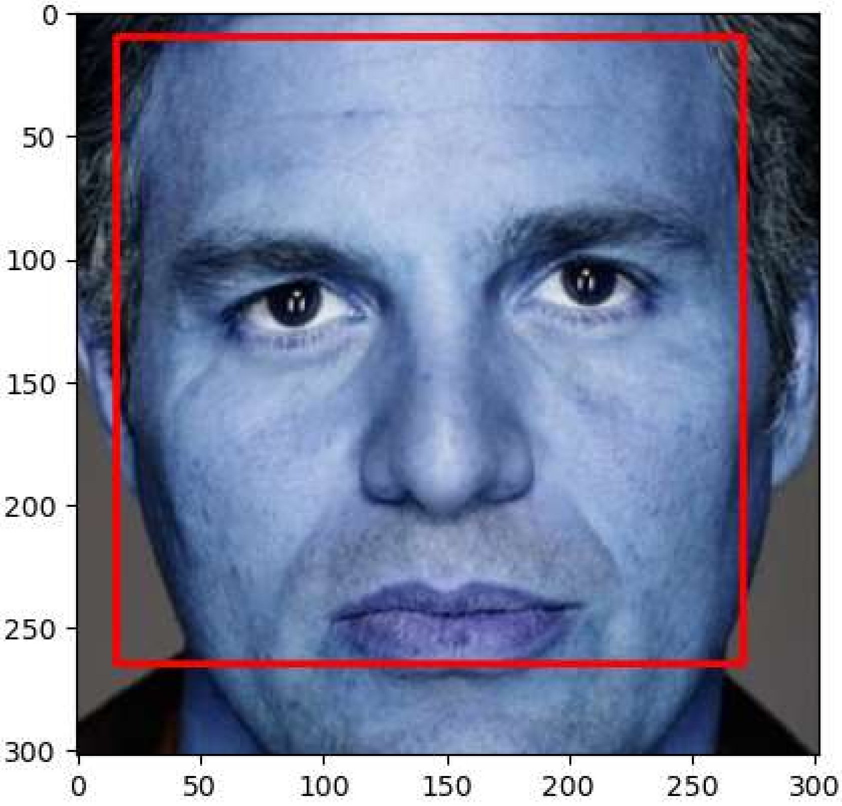
```
(16, 10, 255, 255)
```

In [42]:

```
face_img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(face_img)
```

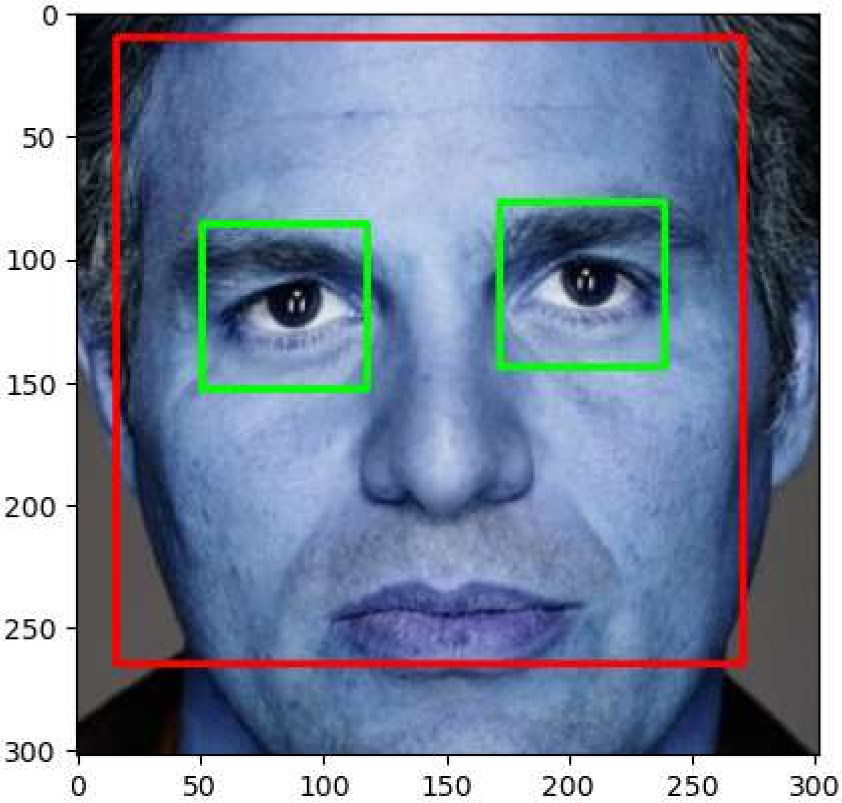
Out[42]:

```
<matplotlib.image.AxesImage at 0x1b7b164d8d0>
```



In [44]:

```
for (x,y,w,h) in faces:
    face_img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray= gray[y:y+h, x:x+w]
    roi_color = face_img[y:y+h,x:x+w]
    eyes= eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
plt.figure()
plt.imshow(face_img,cmap='gray')
plt.show()
```

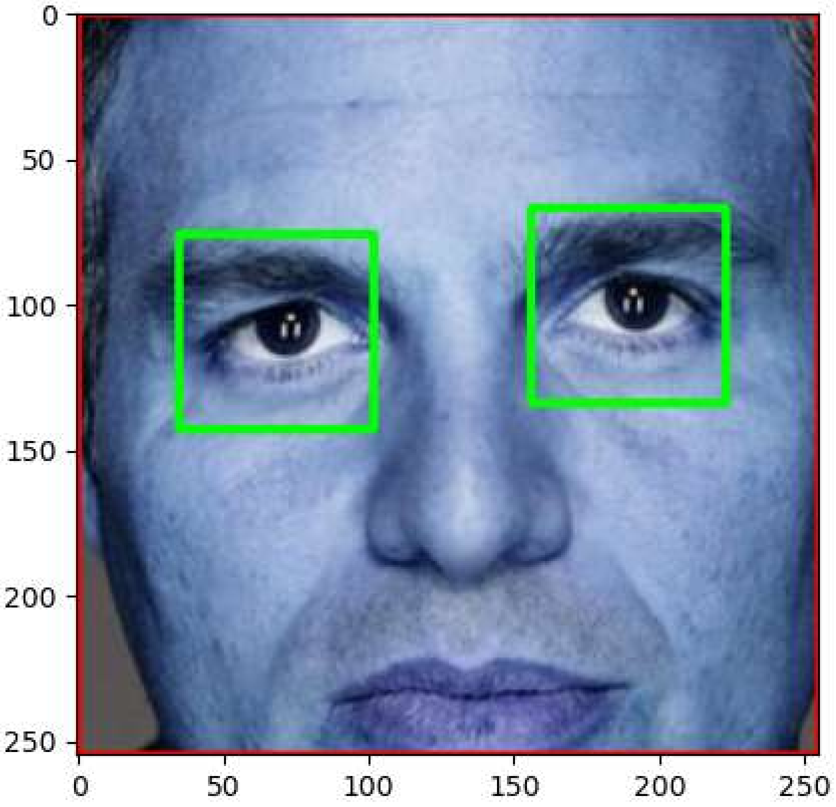


In [45]:

```
%matplotlib inline
plt.imshow(roi_color,cmap='gray')
```

Out[45]:

```
<matplotlib.image.AxesImage at 0x1b7b1823df0>
```



In [47]:

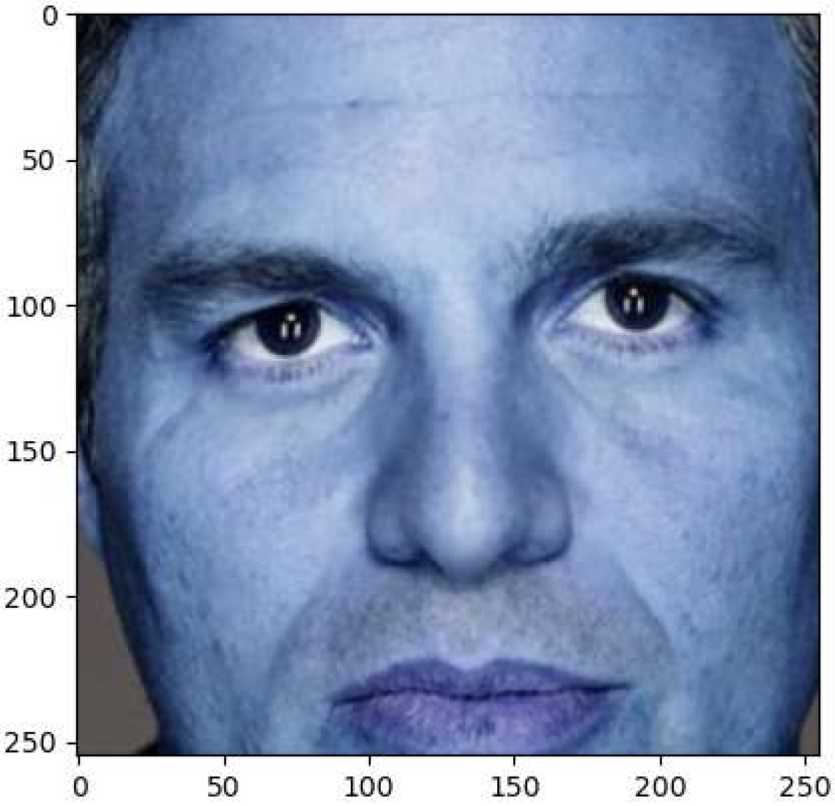
```
def get_cropped(img_path):
    img= cv2.imread(img_path)
    gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,1.3,5)
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)
        if len(eyes) >=2:
            return roi_color
```

In [49]:

```
img=get_cropped(link)
plt.imshow(img)
```

Out[49]:

```
<matplotlib.image.AxesImage at 0x1b7b1d90100>
```



In [59]:

```
import os
source = r'C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled Folder\dat
dest = r'C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled Folder\cropp

for i in os.listdir(source):
    cnt = 1
    for j in os.listdir(os.path.join(source,i)):
        s = os.path.join(source,os.path.join(i,j))
        d = os.path.join(dest,i)
        cropped_img = get_cropped(s)
        if(not os.path.exists(d)):
                os.makedirs(d)
        if(cropped_img is not None):
            print(os.path.join(d,i+str(cnt)))
            cv2.resize(cropped_img,(224,224))
            cv2.imwrite(os.path.join(d,i+str(cnt)+'.jpg'),cropped_img)
        cnt+=1
```

```
  Cell In[59], line 2
    source = r'C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled Fold
er\data'}

^
SyntaxError: unmatched '}'
```

In [16]:

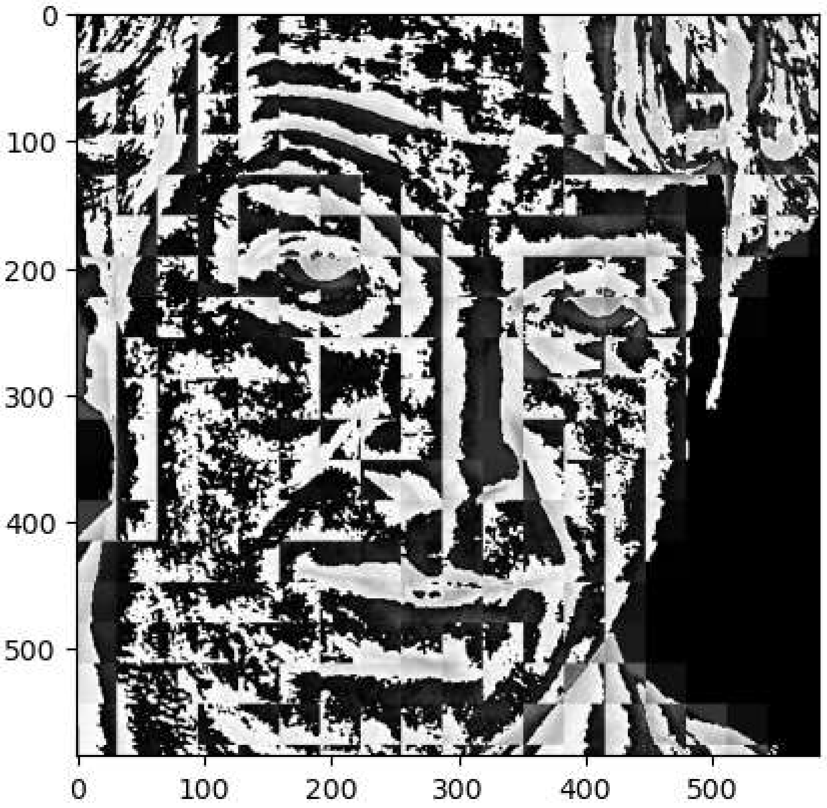
```
import numpy as np
import pywt
import cv2

def w2d(img, mode='haar', level=1):
    imArray = img
    #Datatype conversions
    #convert to grayscale
    imArray = cv2.cvtColor( imArray,cv2.COLOR_RGB2GRAY )
    #convert to float
    imArray =  np.float32(imArray)
    imArray /= 255;
    # compute coefficients
    coeffs=pywt.wavedec2(imArray, mode, level=level)

    #Process Coefficients
    coeffs_H=list(coeffs)
    coeffs_H[0] *= 0;

    # reconstruction
    imArray_H=pywt.waverec2(coeffs_H, mode);
    imArray_H *= 255;
    imArray_H =  np.uint8(imArray_H)

    return imArray_H
```

In [18]:

```
cropped_img=cv2.imread(r"C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untit
im_har = w2d(cropped_img,'db1',5)
plt.imshow(im_har, cmap='gray')
```

Out[18]:

```
<matplotlib.image.AxesImage at 0x2827ed8a1d0>
```



In [20]:

```
import os
d={}
cnt=0
for i in os.listdir(r'C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled
    d[i]=cnt
    cnt+=1
```

In [21]:

```
d
```

Out[21]:

```
{'bruce': 0, 'clint': 1, 'natasha': 2, 'steve': 3, 'thor': 4, 'tony': 5}
```

In [34]:

```
X=[]
y=[]
directory = r'C:\Users\rohit\OneDrive\B Tech\3.2\Practice\Untitled Folder\

for i in os.listdir(directory):
    for j in os.listdir(os.path.join(directory,i)):
#         print(os.path.join(directory,os.path.join(i,j)))
        img = cv2.imread(os.path.join(directory,os.path.join(i,j)))
        scaled_img = cv2.resize(img,(32,32))
        img_har = w2d(img,'db1',5)
        scaled_har = cv2.resize(img_har,(32,32))
        combined_img = np.vstack((scaled_img.reshape(32*32*3,1),scaled_har
        X.append(combined_img)
        y.append(d[i])
```

In [37]:

```
len(X)
```

Out[37]:

```
759
```

In [47]:

```
X=np.array(X).reshape(len(X),32*32*3+32*32).astype(float)
```

In [48]:

```
X.shape
```

Out[48]:

```
(759, 4096)
```

In [61]:

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
```

In [49]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)
```

In [54]:

```
pipe = Pipeline([('scaler',StandardScaler()),('svc',SVC(kernel = 'rbf', C=
pipe.fit(X_train,y_train)
pipe.score(X_test,y_test)
```

Out[54]:

```
0.7210526315789474
```

In [55]:
```
print(classification_report(y_test,pipe.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.66      0.71      0.68        35
           1       0.76      0.83      0.79        35
           2       0.89      0.82      0.86        40
           3       0.64      0.39      0.48        18
           4       0.77      0.73      0.75        37
           5       0.52      0.64      0.57        25

    accuracy                           0.72       190
   macro avg       0.71      0.69      0.69       190
weighted avg       0.73      0.72      0.72       190
```

In [62]:
```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
```

In [74]:
```
model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto',probability=True),
        'params' : {
            'svc__C': [1,10,100,1000],
            'svc__kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'randomforestclassifier__n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto')
        'params': {
            'logisticregression__C': [1,5,10]
        }
    }
}
```

In [75]:
```
model_params.items()
```

Out[75]:
```
dict_items([('svm', {'model': SVC(gamma='auto', probability=True), 'para
ms': {'svc__C': [1, 10, 100, 1000], 'svc__kernel': ['rbf', 'linear']}}),
('random_forest', {'model': RandomForestClassifier(), 'params': {'random
forestclassifier__n_estimators': [1, 5, 10]}}), ('logistic_regression',
{'model': LogisticRegression(solver='liblinear'), 'params': {'logisticre
gression__C': [1, 5, 10]}})])
```

In [81]:

```
scores = []
best_estimators = {}
import pandas as pd
for algo, mp in model_params.items():
    pipe = make_pipeline(StandardScaler(),mp['model'])
    clf = GridSearchCV(pipe,mp['params'],cv=5,return_train_score= False)
    clf.fit(X_train,y_train)
    scores.append({
        'model':algo,
        'best_score':clf.best_score_,
        'best_params':clf.best_params_
    })
    best_estimators[algo] = clf.best_estimator_
    print(1)
df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
```

```
1
1
1
```

In [82]:

```
df
```

Out[82]:

	model	best_score	best_params
0	svm	0.683714	{'svc__C': 1, 'svc__kernel': 'linear'}
1	random_forest	0.393743	{'randomforestclassifier__n_estimators': 10}
2	logistic_regression	0.678420	{'logisticregression__C': 10}

In [83]:

```
best_estimators['svm'].score(X_test,y_test)
```

Out[83]:

```
0.7421052631578947
```

In [84]:

```
best_estimators['random_forest'].score(X_test,y_test)
```

Out[84]:

```
0.4263157894736842
```

In [85]:

```
best_estimators['logistic_regression'].score(X_test,y_test)
```

Out[85]:

```
0.6947368421052632
```

In [88]:

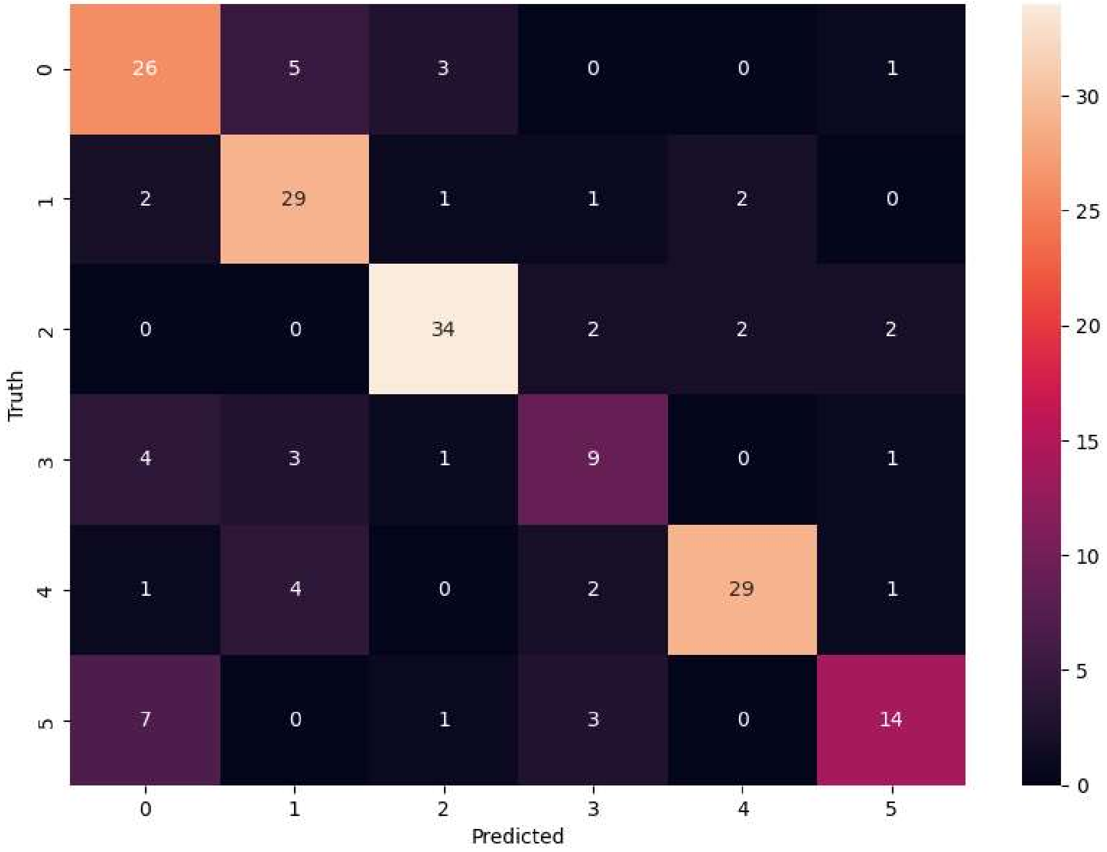
```
best_model = best_estimators['svm']
```

In [89]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,best_clf.predict(X_test))
cm
```

Out[89]:
```
array([[26,  5,  3,  0,  0,  1],
       [ 2, 29,  1,  1,  2,  0],
       [ 0,  0, 34,  2,  2,  2],
       [ 4,  3,  1,  9,  0,  1],
       [ 1,  4,  0,  2, 29,  1],
       [ 7,  0,  1,  3,  0, 14]], dtype=int64)
```

In [90]:
```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[90]:
```
Text(95.72222222222221, 0.5, 'Truth')
```



In [91]:
```
import joblib
joblib.dump(best_model,'classification.pkl')
```

Out[91]:
```
['classification.pkl']
```