COMPUTER VISION PROJECT

Activity Recognition

DR : Dina Khateb
TA : Abdel-Rahman Shaker
TA : Hadeer Hossin

1) Hadeer sabry abd elsalam
2) Heba massud awad
3) Ahmed adb elaty younis
4) Mohamed ahmed Mahmoud nassar

Table of Content

Chapter

1

Project Problem Definition

Activity Recognition Focused On Evaluation Of Isolated Single Person Behavior .

AR Is A Widely Studied Computer Vision Problem.

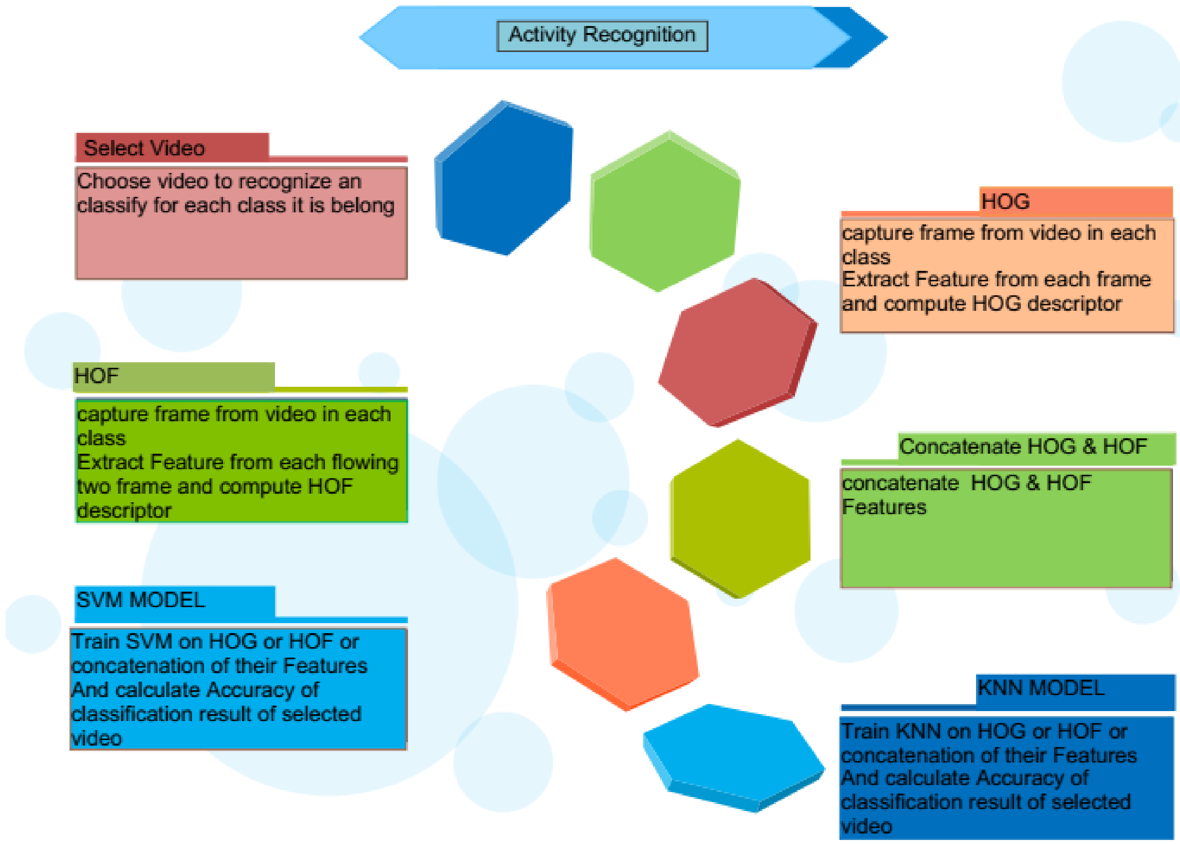
This Project Is Vision-Based Activity Recognition Targets To Learn How To Detect A Human Body From A Video And Describe The Activity Of The Human Using COMPUTER VISION METHOD Such As **HOG (**Histogram Of Gradients**)**

And **HOF (**Histogram of Optical Flow**)** Descriptors to Extract Features

And Use Machine-Learning Technique **SVM(**Support Vector Machine**)** and **KNN(**K-Nearest Neighbors**)** Classifier to Classify and Recognition.

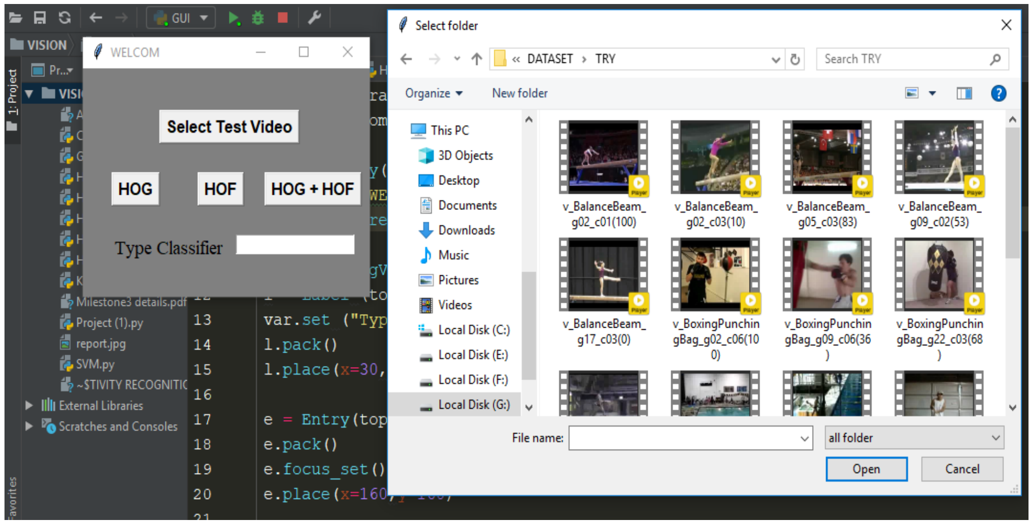
Chapter 2

Methodology:



1. **Run Program Then Select Video Which Want to Classify.**

2. Choose between one of feature Extraction METHODS either HOG or HOF or combination of them to compute Descriptors.

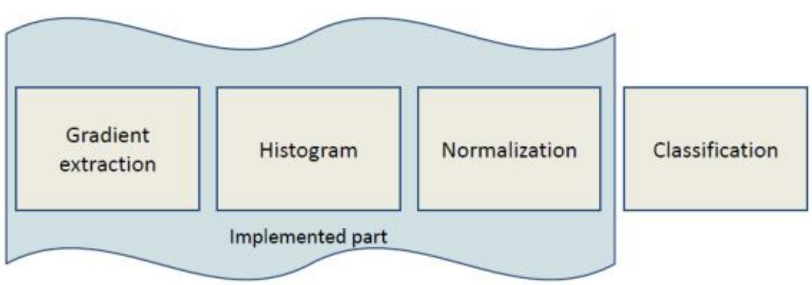


3. Apply classification technique SVM or KNN

4. Compute And Display ACCURACY of Prediction For Selected Algorithm

Techniques Implementation:

HOG:



Histogram of Oriented Gradients for Human Detection.

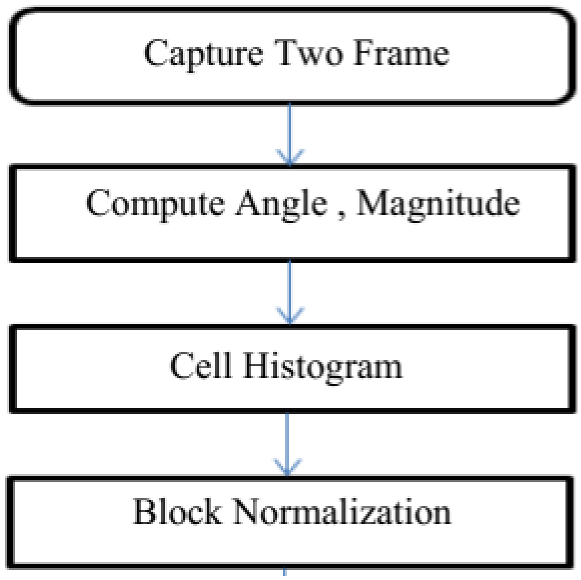
```python
import cv2

hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
hogParams = {'winStride':(8, 8) , 'padding': (16, 16), 'scale': 1.05}
```

HOG Feature Extraction procedure

```python
def HOG(vid , label):
    Features = []
    Labels = []
    vidcap = cv2.VideoCapture(vid)
    count = 0
    sec = 0
    while True:
        frameRate = 0.25  # //it will capture image in each 0.5 second
        vidcap.set(cv2.CAP_PROP_POS_MSEC, sec * 1000)
        ret, frame = vidcap.read(0)
        if ret:
            cv2.imwrite("frame%d.jpg" % count, frame)  # save frame as JPEG file
        count += 1
        if not ret:
            break
        sec = sec + frameRate
        sec = round(sec, 2)
        (rects, weights) = hog.detectMultiScale(frame, **hogParams)
        for (x, y, w, h) in rects:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        frame = cv2.resize(frame, (64, 128), interpolation=cv2.INTER_AREA)
        descriptor = hog.compute(frame)
```

HOF:



Calculate Optical flow for each consecutive Frame then find the magnitude and direction of gradient

```
for j in range(0, frame1.shape[1], 8):
    prvs1 = prvs[i:i + 8, j:j + 8]
    next1 = next[i:i + 8, j:j + 8]
    flow = cv2.calcOpticalFlowFarneback(prvs1, next1, None, 0.5, 3, 15, 3, 5, 1.2, 0)
    mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    angle = ang * 180 / np.pi / 2
    magnitude=cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
```

Calculate Histogram of Gradients in 8×8 cells

```python
def cell_histogram(cell_direction, cell_magnitude):
    hist_bins = np.array([0, 20, 40, 60, 80, 100, 120, 140, 160])
    cell_hist = np.zeros(shape=(hist_bins.size))
    cell_size = cell_direction.shape[0]
    for r in range(cell_size - 1):
        for c in range(cell_size - 1):
            current_direct = cell_direction[r, c]
            current_magnit = cell_magnitude[r, c]
            diff = np.abs(current_direct - hist_bins)
            first_idx = np.where(diff == np.min(diff))[0][0]
            if first_idx == 8:  # hist_bins.size - 1
                temp = hist_bins[[(first_idx - 1), (0)]]
                temp2 = np.abs(current_direct - temp)
                res = np.where(temp2 == np.min(temp2))[0][0]
            else:
                temp = hist_bins[[(first_idx - 1), (first_idx + 1)]]
                temp2 = np.abs(current_direct - temp)
                res = np.where(temp2 == np.min(temp2))[0][0]
            if res == 0 and first_idx != 0:
                second_idx = first_idx - 1
```

16×16 Block Normalization

```python
    for i in range(len(Big_Hof)-1):
        HoF_FEATURE_NO=[]
        for j in range(len(Big_Hof[0])-1):
            x=[Big_Hof[i][j]] + [Big_Hof[i][j+1] ]+[Big_Hof[i+1][j] ]+[Big_Hof[i+1][j+1] ]
            x=np.array(x)
            x=x.reshape(36,1)
            multArr= x*x
            sumMultArr= sum(multArr)
            res= math.sqrt(sumMultArr)
            divideRes= x/res
            HoF_FEATURE_NO.append(divideRes)
        HOF_NORM_FEATURE.append(HoF_FEATURE_NO)
```

Combination Of HOG & HOF :

```python
    # CREATE TRAIN FEATURE
    for sub_dir in os.listdir(parent_dir_train):
        if os.path.isdir(os.path.join(parent_dir_train, sub_dir)):
            for vid in glob.glob(os.path.join(parent_dir_train, sub_dir, file_type)):
                Train_F, Train_L = hog.HOG(vid,label)
                Train_HOF, Train_HOF_l = hof.HOF(vid,label)
                Train_Features.append(Train_F)
                Train_Labels.append(Train_L)
                Train_HOF_Feature.append(Train_HOF)
                Train_HOF_Labels.append(Train_HOF_l)
        label = label + 1
    Train_Features = np.array(Train_Features)
    Train_Labels = np.array(Train_Labels)
    Train_HOF_Feature = np.array(Train_HOF_Feature)
    Train_HOF_Labels = np.array(Train_HOF_Labels)
    for i in range(len(Train_Features)):
        finalTrain += Train_Features[i]
        finalLabel += Train_Labels[i]
        finalHofTrain += Train_HOF_Feature[i]
        finalHofLabel += Train_HOF_Labels[i]
    concatnationHogHof = finalTrain + finalHofTrain
    concatnationHogHofLable = finalLabel + finalHofLabel
```

SVM MODEL:

```python
def svm(trainFeature , trainlable , testFeature , testLable , actualLable):
    SVM = cv2.ml.SVM_create()
    SVM.setType(cv2.ml.SVM_C_SVC)  # AS Classifier
    SVM.setKernel(cv2.ml.SVM_RBF)
    SVM.setGamma(0.01)
    SVM.train(trainFeature, cv2.ml.ROW_SAMPLE, trainlable)
    Labels_pred = SVM.predict(testFeature)[1].ravel()
    Labels_pred = np.array(Labels_pred)
    acc=accuracy_score(testLable, Labels_pred) * 100
    print(actualLable , Labels_pred[0] ,acc)
    root = Tk()
```

KNN MODEL:

```python
from sklearn.neighbors import KNeighborsClassifier
def knn(trainFeature , trainlable , testFeature , testLable,actualLable):
    model = KNeighborsClassifier(n_neighbors=3)
    model.fit(trainFeature,trainlable)
    predictLable= model.predict(testFeature)
    acc = accuracy_score(testLable, predictLable) * 100
    root = Tk()
    T = Text(root, height=6, width=30)
    root.title("Final OutPut")
    T.pack()
    T.insert(END, "Actual Class : ")
    T.insert(END, actualLable+1)
    T.insert(END, "\n\n")
    T.insert(END, "Predicted Class : ")
    T.insert(END, int(predictLable[0])+1)
    T.insert(END, "\n\n")
    T.insert(END, "Accuracy SVM : ")
    T.insert(END, acc)

    mainloop()
```

Chapter

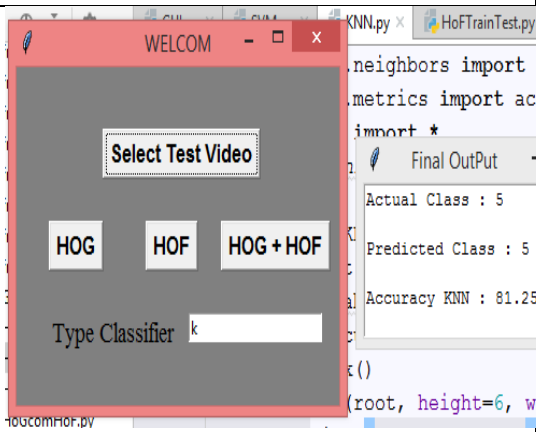
3

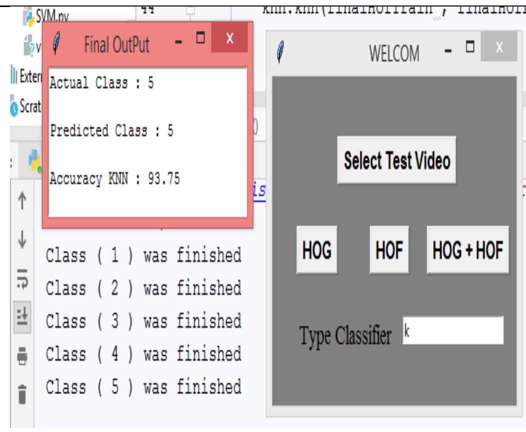
Accuracy for Each Algorithm:

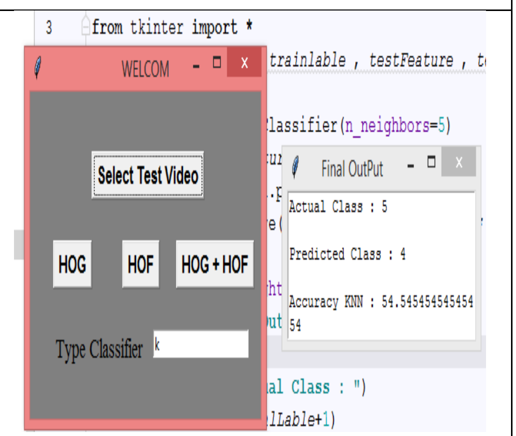
	SVM	SVM	KNN	KNN	KNN
	LINEAR	*RBF*	*K = 3*	*K = 5*	*K = 1*
HOG	**90.30%**	**90%**	**96%**	**89%**	**74.19%**
HOF	**100%**	**100%**	**93.75%**	**54.50%**	**81.25%**
HOG & HOF	**85%**	**100%**	**63%**	**65.60%**	**70.70%**

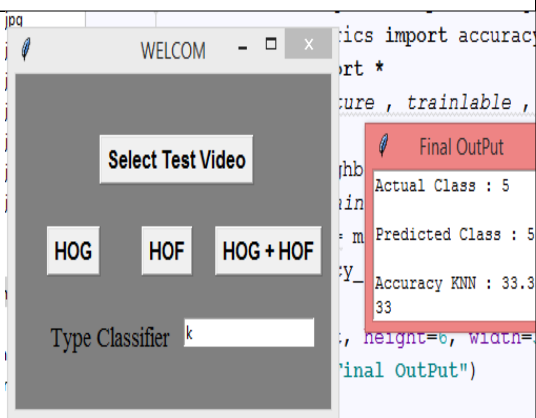
Chapter 4

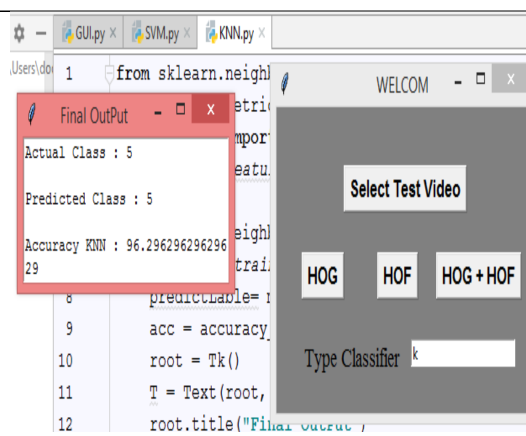
Snapshots:

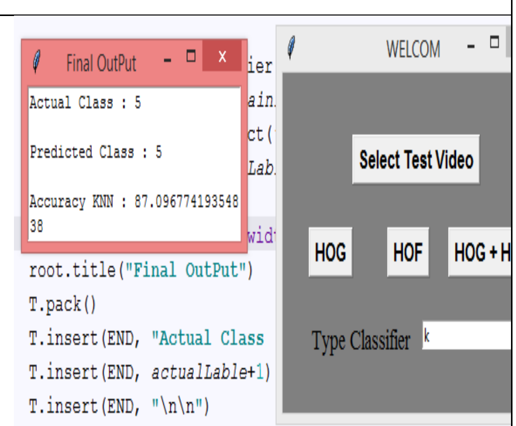
<table>
<tr><td colspan="3">KNN</td></tr>
<tr><td>K=1</td><td>K=3</td><td>K=5</td></tr>
<tr>
<td>

HOF</td>
<td>

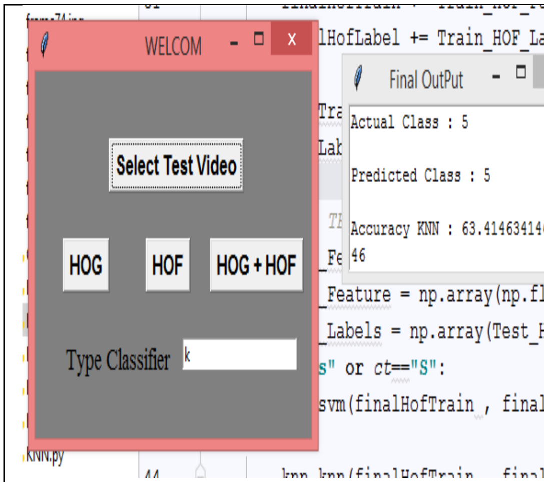
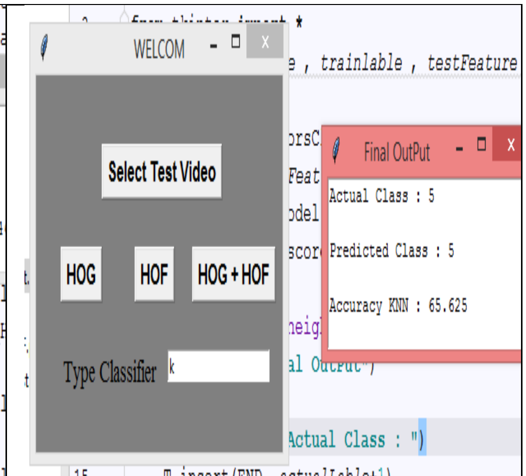
HOF</td>
<td>

HOF</td>
</tr>
<tr>
<td>

HOG</td>
<td>

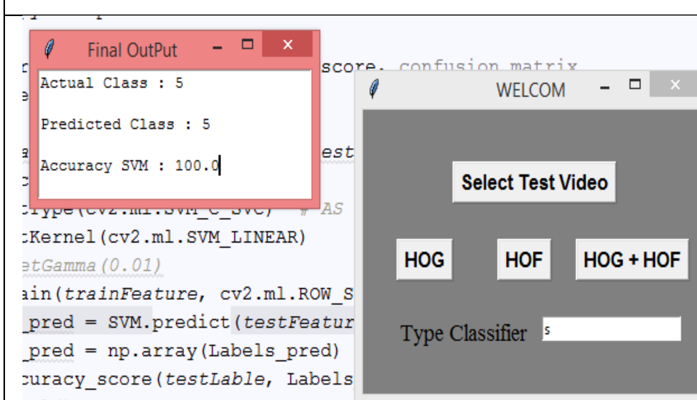
HOG</td>
<td>

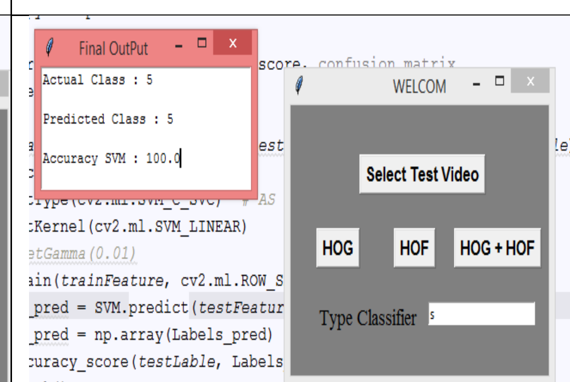
HOG</td>
</tr>
</table>

<table>
<tr>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>HOG & HOF

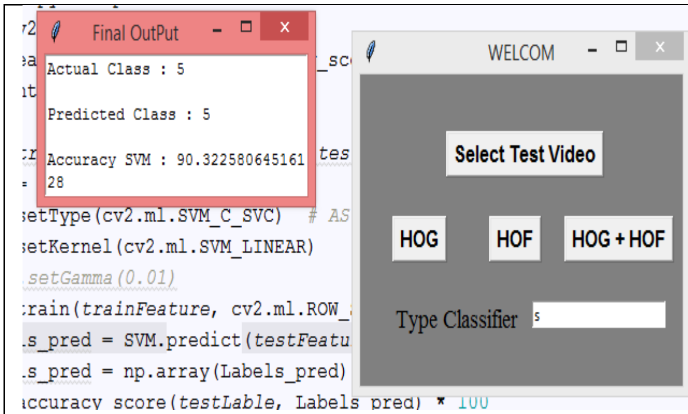
K = 3</td>
<td>HOG & HOF

K = 5</td>
</tr>
</table>

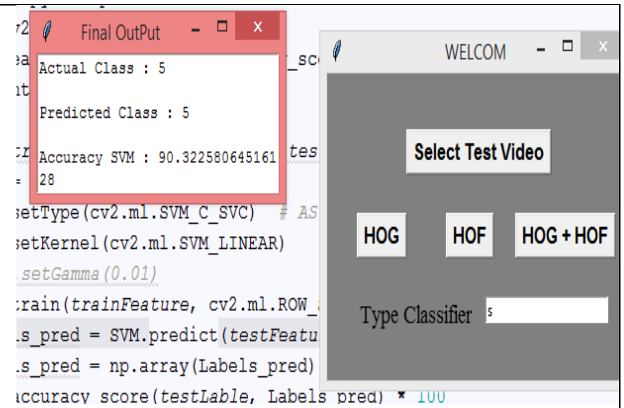
<table>
<tr>
<td colspan="2">SVM</td>
</tr>
<tr>
<td>Linear</td>
<td>RBF</td>
</tr>
<tr>
<td>

HOF</td>
<td>

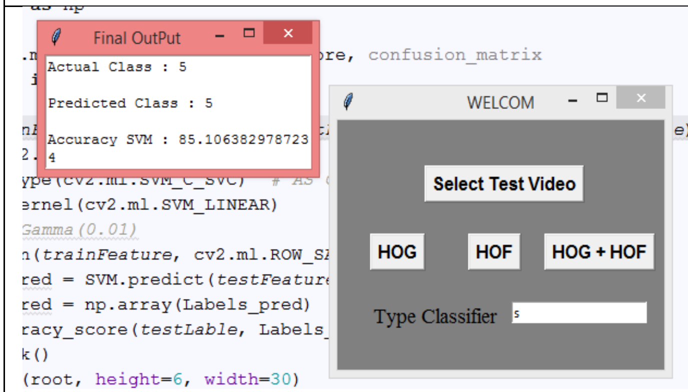
HOF</td>
</tr>
</table>



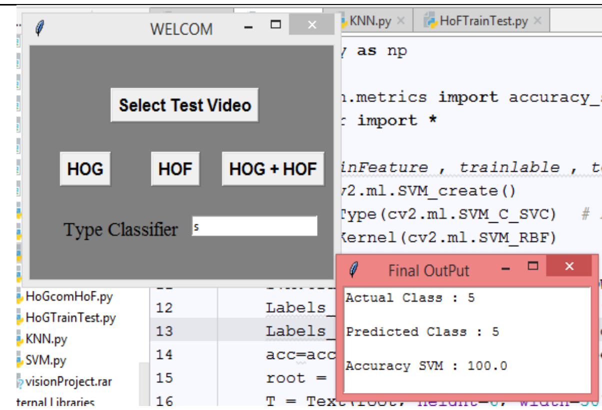
HOG



HOG



HOG & HOF



HOG & HOF

Chapter

5

Conclusion:

Detection Feature with HOG is More Accurate with KNN in Case Number of classes = 3 or 5

KNN is More Accurate Than SVM

On the other hand HOF Is More Accurate in SVM in Case of Kernel is Linear or RBF

More Train Videos More And More Variation of Video More Result Accuracy Come