Indian Institute of Science Education and Research Bhopal

Computer Vision(DSE-312/EECS-320)

Assignment-2

Name: Jiya Sinha
Roll No.: 22161
Time of submission: Marks Obtained:

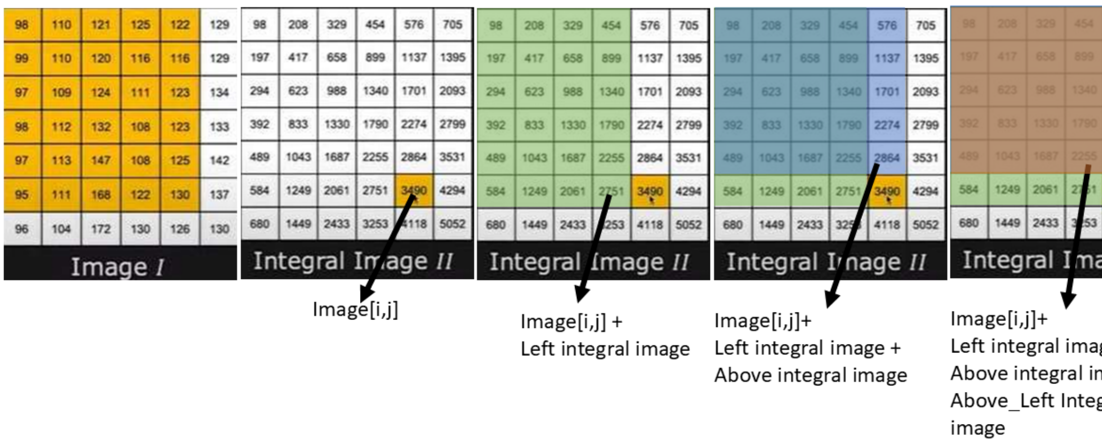
Please follow the instructions given in the assignment carefully.

Please provide your detailed answers and any explanations or diagrams directly below each question in the 'Answers' section.

1. Implement a face detection algorithm from scratch using Haar-like features and Integral Image computation. ***(Marks: 1+2+7)***

   - Capture an image of yourself using a webcam or upload a face image.
   - Compute the Integral Image to efficiently calculate pixel sums over rectangular regions. Use integral image to detect face using Haar features.

Answer:
Step 1: Calculate the Integral Image



Using the following:

Algorithm 1 Algorithm for finding integral image

Input Grayscale Image

Step 1: Find height and width of the image.

$(h, w) \leftarrow image.shape$

Step 2: Intitialize the integral image as zero , of the size same as that of the initial image .

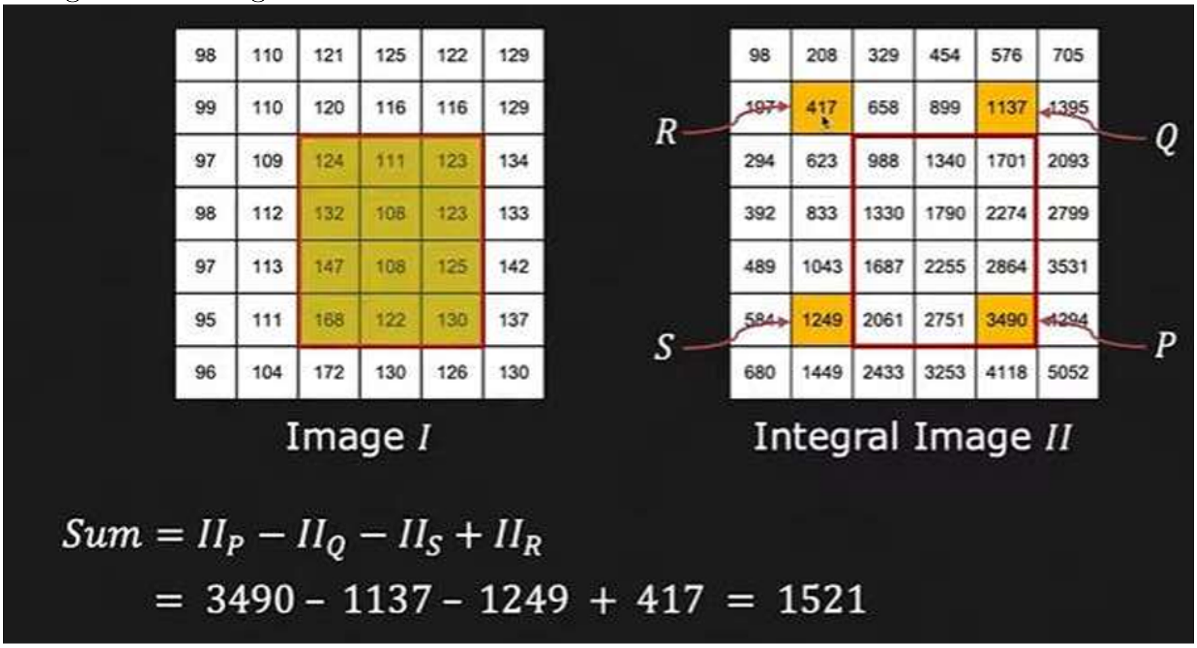
Step 3: Loop through the image, and for each pixel find the value of integral image at that location by:

$II[i, j] = I[i, j] + II[i, j-1] + II[i-1, j] - II[i-1, j-1]$

Step 4: Return the integral image

Finding the sum of rectangle:

Using the following:



Algorithm 2 Algorithm for Finding the Sum of a Rectangular Region in an Integral Image

---

Input Integral Image II, Coordinates P, Q, S, R (corners of the rectangle)
Step 1: Extract the dimensions of the integral image:

$$(h, w) \leftarrow II.shape$$

Step 2: Initialize the variable $total\_sum \leftarrow 0$
Step 3: **If** $0 \leq x_P < w$ **and** $0 \leq y_P < h$:

$$total\_sum \leftarrow total\_sum + II[y_P, x_P] \quad \text{(Include corner P)}$$

Step 4: **If** $0 \leq x_Q < w$ **and** $0 \leq y_Q < h$:

$$total\_sum \leftarrow total\_sum - II[y_Q, x_Q] \quad \text{(Exclude corner Q)}$$

Step 5: **If** $0 \leq x_S < w$ **and** $0 \leq y_S < h$:

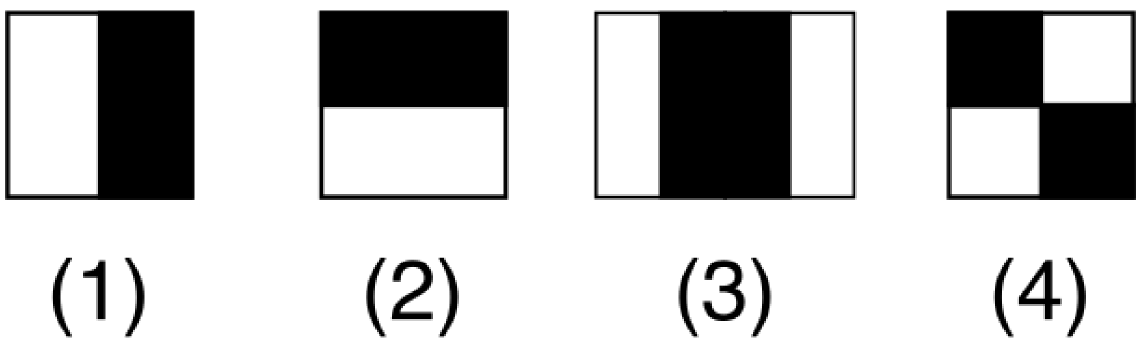
$$total\_sum \leftarrow total\_sum - II[y_S, x_S] \quad \text{(Exclude corner S)}$$

Step 6: **If** $0 \leq x_R < w$ **and** $0 \leq y_R < h$:

$$total\_sum \leftarrow total\_sum + II[y_R, x_R] \quad \text{(Include corner R)}$$

Step 7: Return $total\_sum$ (Sum of the pixels in the rectangular region)

In the next algorithms, sum_region is an additional function that takes input the top-left corner, height and width of the region and return P,Q,R,S for passing through Algorithm 2.
Step 2: Calculating the Harr Features
Harr Features:



Algorithm 3 Algorithm for Calculating Haar Features (1) and (2)

Input Integral Image II, Top-left corner coordinates (x, y), Width w, Height h, Vertical flag $vertical$

Step 1: Set $width\_by\_2 \leftarrow \frac{w}{2}$ and $height\_by\_2 \leftarrow \frac{h}{2}$

Step 2: **If** $vertical$ is **True**:

- Step 2.1: Calculate the white region value:

$$white \leftarrow \text{sum\_region}(x, y, width\_by\_2, h, II)$$

- Step 2.2: Calculate the black region value:

$$black \leftarrow \text{sum\_region}(x + width\_by\_2, y, width\_by\_2, h, II)$$

Step 3: **Else** (i.e., when $vertical$ is **False**):

- Step 3.1: Calculate the black region value:

$$black \leftarrow \text{sum\_region}(x, y, w, height\_by\_2, II)$$

- Step 3.2: Calculate the white region value:

$$white \leftarrow \text{sum\_region}(x, y + height\_by\_2, w, height\_by\_2, II)$$

Step 4: Return the difference between the white and black region values:

$$\text{Haar Feature} \leftarrow white - black$$

Algorithm 4 Algorithm for Calculating Three-Rectangle Haar Features

Input Integral Image II, Top-left corner coordinates (x, y), Width w, Height h

Step 1: Set $width\_by\_3 \leftarrow \frac{w}{3}$

Step 2: Calculate the sum of the first white rectangle:

$$white_1 \leftarrow \text{sum\_region}(x, y, width\_by\_3, h, II)$$

Step 3: Calculate the sum of the black rectangle in the middle:

$$black \leftarrow \text{sum\_region}(x + width\_by\_3, y, width\_by\_3, h, II)$$

Step 4: Calculate the sum of the second white rectangle:

$$white_2 \leftarrow \text{sum\_region}(x + 2 \cdot width\_by\_3, y, width\_by\_3, h, II)$$

Step 5: Return the difference between the sum of the white regions and the black region:

$$\text{Three-Rectangle Haar Feature} \leftarrow white_1 + white_2 - black$$

Algorithm 5 Algorithm for Calculating Four-Rectangle Haar Features

Input Integral Image II, Top-left corner coordinates (x, y), Width w, Height h

Step 1: Set $width\_by\_2 \leftarrow \frac{w}{2}$ and $height\_by\_2 \leftarrow \frac{h}{2}$

Step 2: Calculate the sum of the first black rectangle (top-left):

$$black_1 \leftarrow \text{sum\_region}(x, y, width\_by\_2, height\_by\_2, II)$$

Step 3: Calculate the sum of the second black rectangle (bottom-right):

$$black_2 \leftarrow \text{sum\_region}(x + width\_by\_2, y + height\_by\_2, width\_by\_2, height\_by\_2, II)$$

Step 4: Calculate the sum of the first white rectangle (top-right):

$$white_1 \leftarrow \text{sum\_region}(x + width\_by\_2, y, width\_by\_2, height\_by\_2, II)$$

Step 5: Calculate the sum of the second white rectangle (bottom-left):

$$white_2 \leftarrow \text{sum\_region}(x, y + height\_by\_2, width\_by\_2, height\_by\_2, II)$$

Step 6: Return the difference between the sums of the white and black regions:

$$\text{Four-Rectangle Haar Feature} \leftarrow white_1 + white_2 - black_1 - black_2$$

Step 3: Calculate the Harr features from sliding the window across the image

Algorithm 6 Algorithm for Extracting Haar Features from Sliding Windows

Input Integral Image II, Stride $stride$, Window Size $window\_size$

Step 1: Extract height h and width w of the integral image:

$$(h, w) \leftarrow II.shape$$

Step 2: Extract window height wh and width ww:

$$(ww, wh) \leftarrow window\_size$$

Step 3: Initialize an empty list $features$ to store extracted features.

Step 4: **For** y in range 0 to $h - wh + 1$ with step $stride$:

for each row of the image **do**

Step 5: **For** x in range 0 to $w - ww + 1$ with step $stride$:

for each column of the image **do**

Step 6: Extract the window from the integral image:

$$window \leftarrow II[y : y + wh, x : x + ww]$$

Step 7: Call $extract\_haar\_features(window, (x, y), window\_size)$ to get feature values.

Step 8: Append the feature values and position (x, y) to the features list:

$$features.append((feature\_values[0], (x, y)))$$

Step 9: Return the list of extracted features:

$$\text{Return } features$$

Algorithm 7 Algorithm for Extracting Haar Features from a Window

Input Window $window$, Top-Left Coordinate (x, y), Window Size $window\_size$

Step 1: Initialize an empty list $features$ to store extracted Haar features.

Step 2: Extract window height wh and width ww:

$$(ww, wh) \leftarrow window\_size$$

Step 3: Compute Haar Feature 1 (Two-Rectangle Feature Vertical):

$$f1 \leftarrow haar\_two\_rectangle(window, (x, y), ww, wh, \text{vertical} = \text{True})$$

Step 4: Compute Haar Feature 2 (Two-Rectangle Feature Horizontal):

$$f2 \leftarrow haar\_two\_rectangle(window, (x, y), ww, wh, \text{vertical} = \text{False})$$

Step 5: Compute Haar Feature 3 (Three-Rectangle Feature):

$$f3 \leftarrow haar\_three\_rectangle(window, (x, y), ww, wh)$$

Step 6: Compute Haar Feature 4 (Four-Rectangle Feature):

$$f4 \leftarrow haar\_four\_rectangle(window, (x, y), ww, wh)$$

Step 7: Append the computed Haar features as a dictionary to the features list:

$$features.append(\{"HaarFeature1" : f1, "HaarFeature2" : f2,$$

"Haar Feature 3": f3,
"Haar Feature 4": f4
})

Step 8: Return the list of Haar features:

$$\text{Return } features$$

=0

Step 4: Classifying the faces using thresholding

Algorithm 8 Algorithm for Classifying Faces Based on Haar Features

Input List of Features $features$, Lower Thresholds $lower\_thresh$, Upper Thresholds $upper\_thresh$

Step 1: Initialize an empty list $potential\_faces$ to store detected face regions.

Step 2: Extract window height wh and width ww:

$$(ww, wh) \leftarrow window\_size$$

Step 3: Loop through each $(feature\_values, (x, y))$ in $features$:

for each $(feature\_values, (x, y))$ in $features$ **do**

Step 3.1: Access the Haar features:

$$f1 \leftarrow feature\_values["HaarFeature1"]$$

$$f2 \leftarrow feature\_values["HaarFeature2"]$$

$$f3 \leftarrow feature\_values["HaarFeature3"]$$

$$f4 \leftarrow feature\_values["HaarFeature4"]$$

Step 3.2: Apply thresholding for each Haar feature:

if (lower_thresh[0] f1 upper_thresh[0] and
lower_thresh[1] f2 upper_thresh[1] and
lower_thresh[2] f3 upper_thresh[2] and
lower_thresh[3] f4 upper_thresh[3]) **then**

Step 3.2.1: Append the face coordinates and size to $potential\_faces$:

$$potential\_faces.append((x, y, ww, wh))$$

end if

end for

Step 4: Return the list of potential faces:

$$\text{Return } potential\_faces$$

Step 6: Drawing rectangle

Algorithm 9 Algorithm for Drawing Rectangles Around Detected Faces

Input Grayscale Image *image*, Detections *detections*
Output RGB Image with Rectangles *output_image*
Step 1: Convert the grayscale image to RGB format:

$$output\_image \leftarrow \text{stack}([image, image, image], \text{axis} = -1)$$

Step 2: **For each detection** (x, y, w, h) in *detections*:
Step 2.1: Draw the top edge of the rectangle:

$$output\_image[y, x : x + w, 0] \leftarrow 255$$

Step 2.2: Draw the bottom edge of the rectangle:

$$output\_image[y + h - 1, x : x + w, 0] \leftarrow 255$$

Step 2.3: Draw the left edge of the rectangle:

$$output\_image[y : y + h, x, 0] \leftarrow 255$$

Step 2.4: Draw the right edge of the rectangle:

$$output\_image[y : y + h, x + w - 1, 0] \leftarrow 255$$

Step 2.5: Set green and blue channels to 0 for the top edge:

$$output\_image[y, x : x + w, 1 : 3] \leftarrow 0$$

Step 2.6: Set green and blue channels to 0 for the bottom edge:

$$output\_image[y + h - 1, x : x + w, 1 : 3] \leftarrow 0$$

Step 2.7: Set green and blue channels to 0 for the left edge:

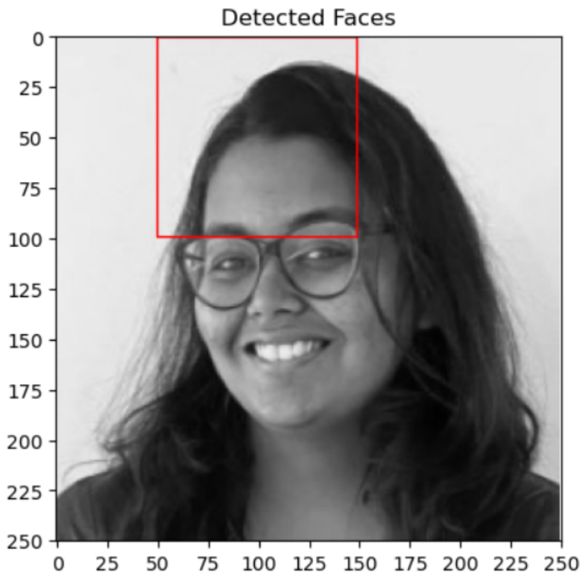
$$output\_image[y : y + h, x, 1 : 3] \leftarrow 0$$

Step 2.8: Set green and blue channels to 0 for the right edge:

$$output\_image[y : y + h, x + w - 1, 1 : 3] \leftarrow 0$$

Step 3: **Return** *output_image*

OUTPUT IMAGE:



This is not a perfect face detection since the thresholds have not being tuned properly.

2. Using dataset **link**, implement a face anti-spoofing model that performs classification based on the below different feature extraction methods to identify fake (spoof) and real image. Compare and analyze your results using metrics accuracy, f1-score, precision, recall and confusion matrix. Document the findings and discuss the failure and success of each method. (*Note: You have to use only 1000 images and not the whole dataset*) ***(Marks: 3+3+4)***

   (a) Using the raw pixel values of the face images as features. Train a Support Vector Machine (SVM) classifier on these raw pixel features to perform face recognition. Evaluate and analyze the performance of the model on the dataset.

   (b) Extract Local Binary Patterns (LBP) features from the face images for feature extraction. Train an SVM classifier using the LBP features to perform face recognition.

   (c) Compute edge images using any two edge detectors (canny, sobel, prewitt, etc.), then use them as input features independently to train an SVM classifier and perform classification.

Answer:

```python
# INPUT X,y ARE THE IMAGES AND IT'S LABEL.
# CLASSNUM TAKEN FOR FINDING THE METRICS LATER.
def classify(X, y, classnum):
    # SPLITTING THE DATA TO TRAIN AND TEST
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

    # USING SUPPORT VECTOR CLASSIFIER. RANDOM_STATE DEFINED TO
    ENSURE SAME RESULT IN EACH RUN.
    clf = SVC(random_state=42)

    # FITTING THE CLASSIFIER WITH TRAINING DATA
    clf.fit(X_train, y_train)

    # PREDICTING THE VALUES FROM TEST DATA
```

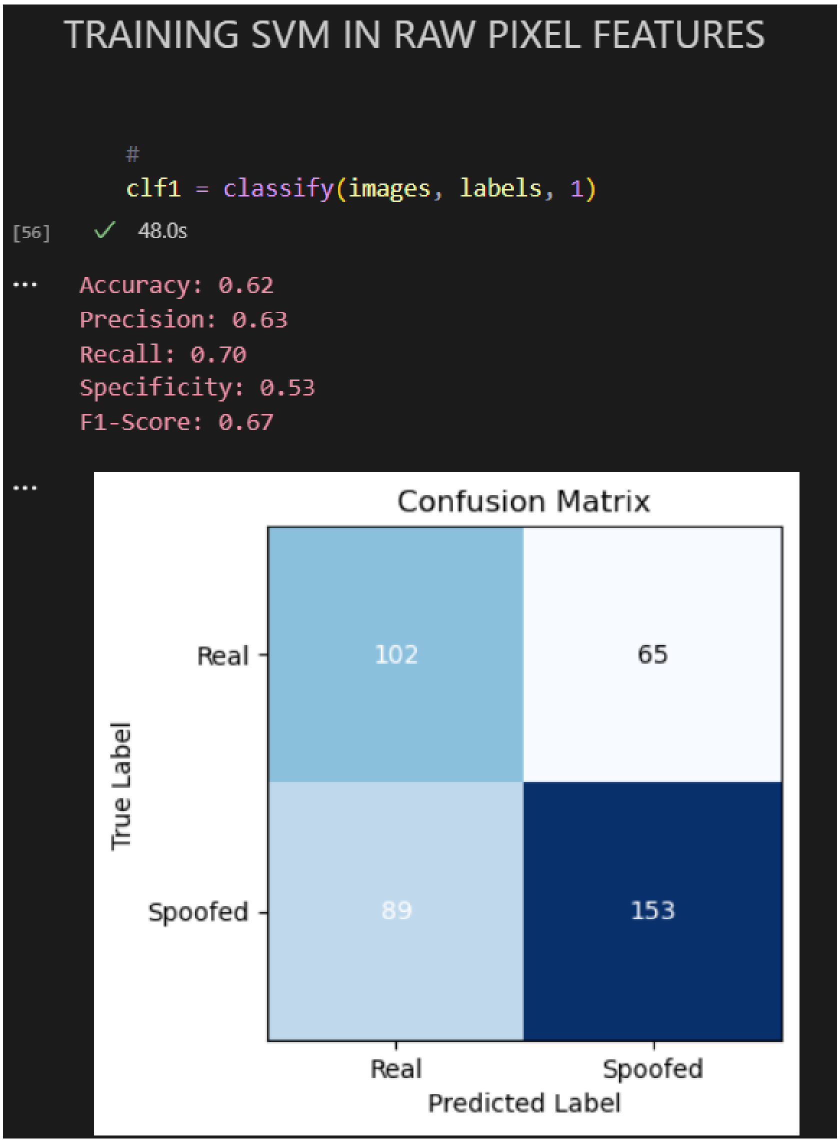
```
    y_pred = clf.predict(X_test)

    # SHOW THE METRICS OF THE CLASSIFIER
    show_metrics(y_test, y_pred, classnum)

    # RETURN THE CLASSIFIER
    return clf
```

Listing 1: FUNCTION TO CLASSIFY, PREDICT AND SHOW THE METRICS

The above function takes feature vectors and corresponding labels as input and splits it into training and testing data and then train the SVC classifier with training data. Using the model, then we predict the class label of the test data. using the true values and predicted value, we then find the metrics. This function is written to perform classification, prediction and showing of metrics together.

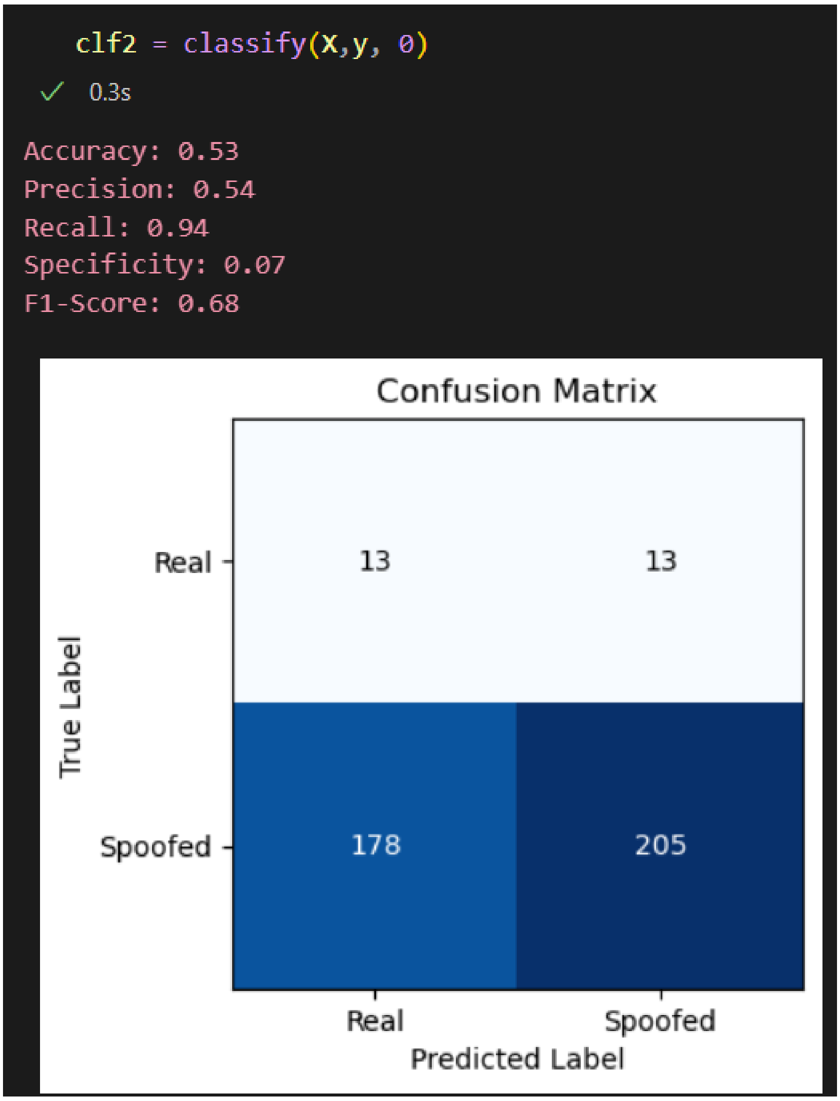
(a) 

Metric on training SVM on Raw Pixel Values

Reason for lower values of the metrics:

- Using raw pixel values directly as features might not capture the essential characteristics needed for effective classification.

- Due to the 1-D nature of the data given to the classifier, it is unable to capture many essential details for each class.

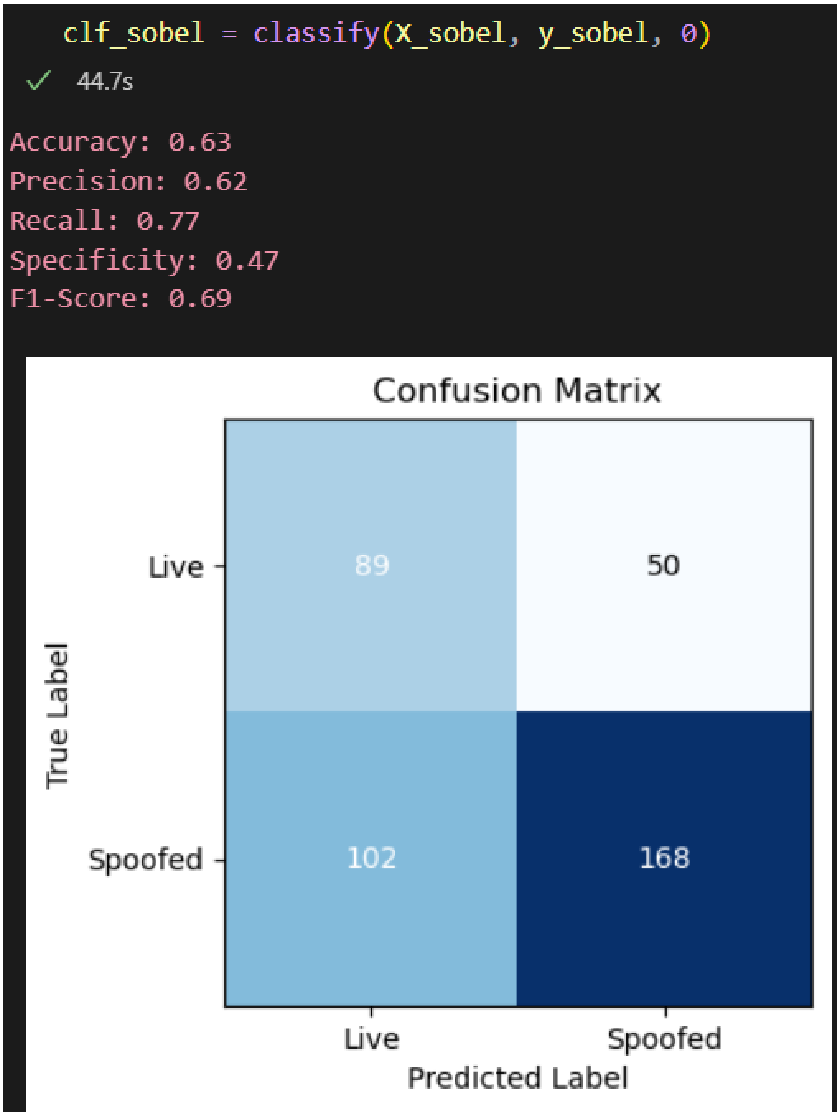
(b) 

Metric on training SVM on LBP Values

The classifier seems to be confused in this case. There is a large number of images (178) that has been falsely predicated Real.

Reason for this false predictions:

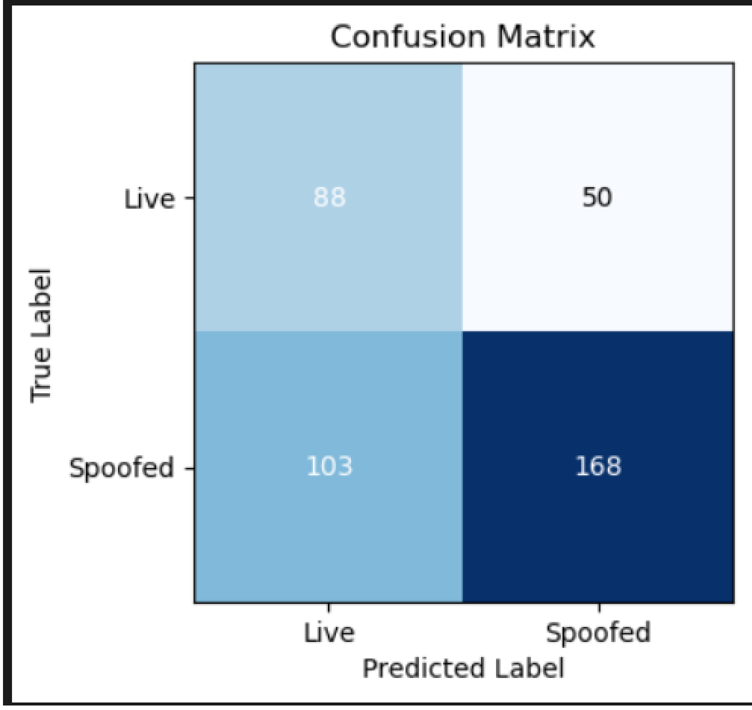
This might be due to the fact that LBP captures local patterns well. A real and spoofed face have **almost** same Local Patterns for eg, the eyes, that are mostly present in both real and spoofed images, might have similiar LBP in each image. Since the images are of persons only, there are high chances of similiar facial features in real and spoofed images are similiar. Due to this, the model might be confused.



(c)

Metric on training SVM on Edge Images Calulated using Sobel Edge Detector

```
clf_prewitt = classify(X_prewitt, y_prewitt, 0)
✓ 47.8s
Accuracy: 0.63
Precision: 0.62
Recall: 0.77
Specificity: 0.46
F1-Score: 0.69
```



Metric on training SVM on Edge Images Calulated using Prewitt's Edge Detector

Here also, a lot of images have been falsely predicted (152/153), this might be due to fact that edge detector finds the egde information and the dataset include some images with nomenclature "hard.." accessing which, we can clearly see that these images are **almost** like the real images and for such images, the edges might be same with a subtle difference. From the metrics of all the 4 ways, we can say that none of these method is "perfect" for the classification using SVM classifier.

Comparision: If we take F1 score as the basis of our comparision, then, we can see that all the models perform almost similiarly with the F1 score of around 0.68. Since, all the metrics have almost the same value, we can't say one of these model is better than the other in this case of Anti-Spoofing model. Introducing more features for classification might make the model better.