Unmasking DeepFakes with simple Features

(COURSE: Digital Image Processing)

Team-25:
Shravan kumar(COE17B028)
Karthik Chowdary(COE17B045)
Venkata Krishna(EDM17B038)

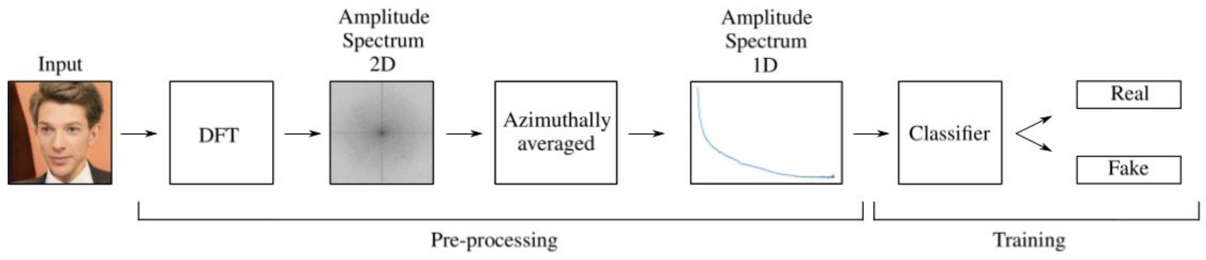
Problem Statement:

Multimedia security is one of the key challenges in today's world, as dependency on multimedia information is increasing day by day. Easily available image editing software has enabled every common use of a smartphone and computer, to hack into the information of the images and video and alter it to some extent. Malicious modification of digital images with the intent to deceive for the sake of altering the public perception is termed as Digital Image Forgery. To authenticate the genuineness of images, detection of image tampering i.e identifying deep fakes is the need of time.

Reference paper solution Vs Improvised solution :

- **Proposed Solution in the given research paper:**

The method proposed in the reference research paper is based on a classical frequency domain analysis followed by a basic classifier. classical frequency analysis of the images that reveal different behaviors at high frequencies.a certain range of frequency components behave when the images have been artificially generated.

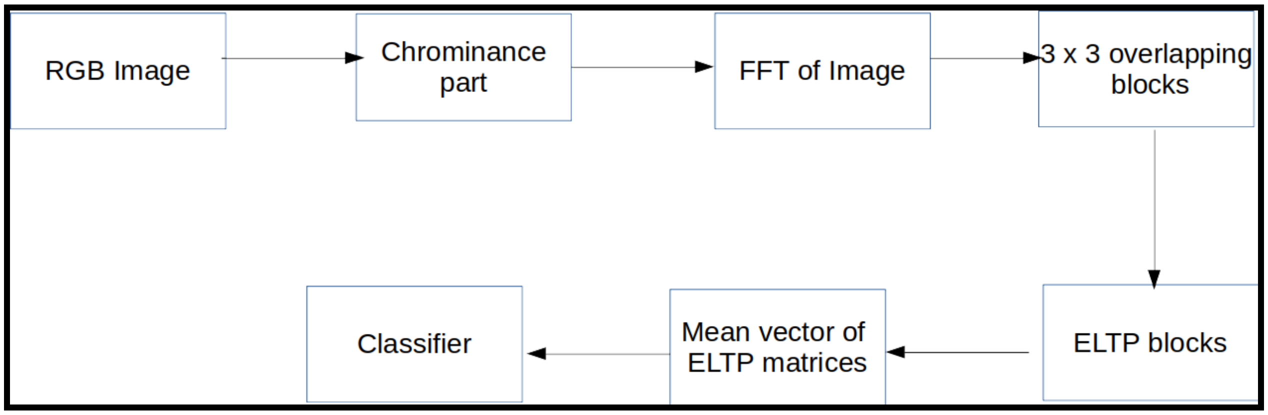


- **Improvised Solution:**

The improvised method looks for local texture descriptors i.e. local binary pattern (LBP) and local ternary pattern (LTP) for forgery detection in an image after applying FFT on the image which improved accuracy to a decent stage.

Image is commonly manipulated with two basic operations of copy-paste or image splicing. Image splicing is a very basic and harmful type of forgery. Image Splicing creates a composite image by blending cropped regions to the same or different images. Some post-processing operations like blurring are performed after pasting the region to completely merge the pasted portion with the background.

Taking into account the fact that pasting of a cropped portion onto a different image, replaces the existing micro-edge patterns of the host image with its own patterns, which make the pasted portion different from the rest of the image and such disturbance is peculiar along its boundary, visual descriptor LTP and its variants are used to encode the micro-edge patterns and DCT is used to encode the frequency content with respect to each blocks LTP value. This approach utilizes the chrominance component of the RGB image which is believed to capture the disturbance created by forgery better than any other color channel.



- **Prerequisites for the improvised solution:**

○ **Local Binary Pattern (LBP):** LBP was proposed for extracting local features of an image. LBP possesses a two-valued code matrix.

$$LBP_{x,y} = \sum_{x=0}^{X-1} s(g_x - g_t)2^x \qquad (1)$$
$$s(x) = \begin{cases} 1, \; if\, x \geq 0; \\ 0, \; otherwise \end{cases}$$

where ‘gx’ and ‘gt’ are the gray value of current neighborhood pixel and central pixel respectively. ‘X’ is the number of pixels in the block. A binary matrix is returned by the above equation which is then converted to decimal.

○ **Local Ternary Pattern (LTP):** LBP is prone to random and quantization noise in near-uniform regions because its value depends on center pixel value. Local ternary pattern (LTP) was proposed as an enhancement to LBP and generates 3- valued code.

$$s'(g_x, i_t, th) = \begin{cases} 1, \; g_x \geq g_t + th \\ 0, \; g_x - g_t < th \\ -1, \; g_x \leq g_x - th \end{cases} \qquad (2)$$

where ‘th’ is the constant threshold value.

○ **Enhanced Local Ternary Pattern(ELTP):** It is the extended version of LTP to attain better features for the image by introducing ELTP. The constant value in the LTP makes it less robust for gray level transformations.ELTP uses a dynamic threshold value based on the mean absolute deviation(mad) of the respective block. ELTP achieves significantly better results in texture classification in comparison to LBP and its variants.

$$s^e(g_x, g_t^e, t^e) = \begin{cases} 1, \; g_x - g_t^e \geq th^e \\ 0, \; g_x - g_t^e < th^e, \quad x = 0, 1, ..., X-1 \\ -1, \; g_x - g_t^e \leq -th^e \end{cases} \qquad (4)$$

● **Forgery Detection using FFT and ELTP (FFT-ELTP):**

1. **Extraction of Chrominance part from RGB Image:** Chrominance (chroma or C for short) is the signal used in video systems to convey the color information of the **picture**, separately from the accompanying luma signal (or Y' for short). **Chrominance** is usually represented as two color-difference components: U = B′ − Y′ (blue − luma) and V = R′ − Y′ (red − luma).
2. **Fast Fourier Transform**: The chrominance component is transformed using Fast Fourier transform and then segmented into overlapping blocks. Fast Fourier transform (FFT) is an efficient way to compute discrete Fourier transform (DFT). An image matrix is transformed to discrete Fourier coefficients by formulation.

$$Y_{p+1,q+1} = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} w_m^{jp} w_n^{kq} X_{j+1,k+1} \qquad (3)$$

3. Enhanced Local Ternary Pattern: Explained the usage in **prerequisites.**

4. Classifier: The generated feature vector in this approach will contain ELTP code for every FFT block of the image. This is fed to the SVM classifier for classification of the image as authentic or forged.

Implementation:

Given an image path/ file, the extract_feature function extracts the monochrome grayscale 8-bit image and produces the second-order histogram of it.
The below code is to find ELTP in the image.

```python
make_histogram = lambda: np.zeros(256, dtype='int')
DIRECTIONS = ('h', 'dr', 'v', 'dl')

def neighbours(x, y):
    return [
        (x-1, y-1), (x-1, y), (x-1, y+1), (x, y+1),
        (x+1, y+1), (x+1, y), (x+1, y-1), (x, y-1)
    ]
```

```python
def g(a, b):
    # hey, this shit makes g() 10s/50imgs faster
    if a == 0 or b == 0: return -1

    p = np.sign(a*b)
    if p == -1: return 1
    if p == 1: return  0
    if p == 0: return -1

def extract_feature(img):
    cimg = img[1:-1, 1:-1]
    # first-order derivate
    i = {'h': cimg - img[1:-1, 2:],
         'v': cimg - img[:-2, 1:-1],
         'dr': cimg - img[:-2, 2:],
         'dl': cimg - img[:-2, :-2],
    }

    width, height = cimg.shape
    indices = ((x, y) for x in range(1, width-1) for y in range(1, height-1))

    histograms = dict(
        ltpp = {'h':  make_histogram(),
                'v':  make_histogram(),
                'dr': make_histogram(),
                'dl': make_histogram(),
        },
        ltpn = {'h':  make_histogram(),
                'v':  make_histogram(),
                'dr': make_histogram(),
                'dl': make_histogram(),
        },
    )

    for (px, py) in indices:
        for direction in DIRECTIONS:
            # centred, first order derivate of image
            p = i[direction][px, py]

            ltpp = ['0', ] * 8
            ltpn = ['0', ] * 8
            for dc, (x, y) in enumerate(neighbours(px, py)):
                ltp = g(i[direction][x, y], p)
                if ltp == 1: ltpp[dc] = '1'
                if ltp == -1: ltpn[dc] = '1'

            ltpp = int(''.join(ltpp), 2)
            ltpn = int(''.join(ltpn), 2)
            histograms['ltpp'][direction][ltpp] += 1
            histograms['ltpn'][direction][ltpn] += 1

    return np.concatenate([histograms[sign][direction]
                           for direction in DIRECTIONS
                           for sign in ('ltpp', 'ltpn')])
```

Comparison of Results:

- **Datasets:** Three datasets are used here which are given in reference paper.
    - Dataset1 - Faces HQ
    - Dataset2 - CelebA low Q
    - Dataset3 - Prepro Dataset

- **Evaluation Metrics:** The performance of the proposed methodology has been evaluated for both approaches using accuracy and recall performance measures formulated in equations 8 and 9 respectively.

$$Accuracy = 100X\frac{TP+TN}{TP+TN+FP+FN} \quad (8)$$

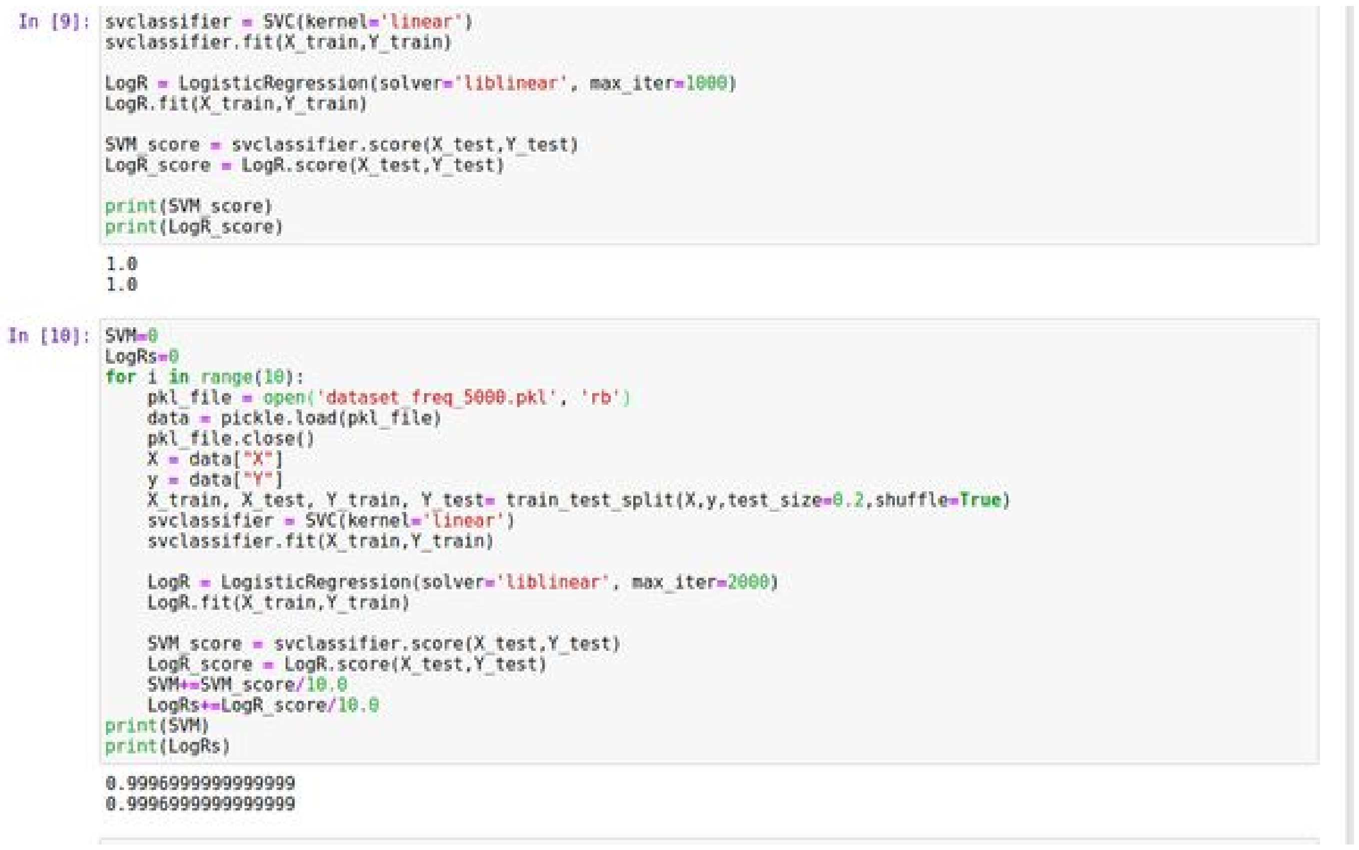
$$Recall = \frac{TP}{TP+FN}X100 \quad (9)$$

TP - True Positives   TN - True Negatives
FP - False Positives   FN - False Negatives

- **Reference Paper results:**

  - Dataset1- Faces HQ:

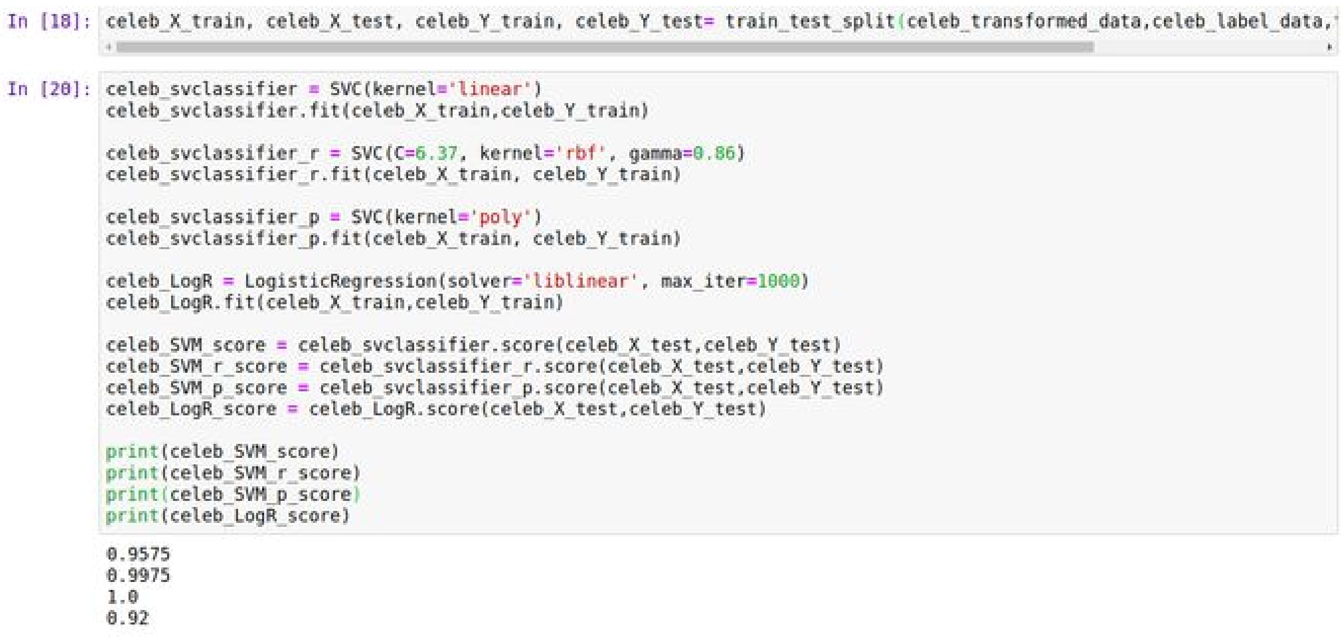
Number of Samples	SVM	Logistic Regression	KNN
5000	100%	100%	80%



```
In [9]: svclassifier = SVC(kernel='linear')
        svclassifier.fit(X_train,Y_train)

        LogR = LogisticRegression(solver='liblinear', max_iter=1000)
        LogR.fit(X_train,Y_train)

        SVM_score = svclassifier.score(X_test,Y_test)
        LogR_score = LogR.score(X_test,Y_test)

        print(SVM_score)
        print(LogR_score)

        1.0
        1.0

In [10]: SVM=0
         LogRs=0
         for i in range(10):
             pkl_file = open('dataset_freq_5000.pkl', 'rb')
             data = pickle.load(pkl_file)
             pkl_file.close()
             X = data["X"]
             y = data["Y"]
             X_train, X_test, Y_train, Y_test= train_test_split(X,y,test_size=0.2,shuffle=True)
             svclassifier = SVC(kernel='linear')
             svclassifier.fit(X_train,Y_train)

             LogR = LogisticRegression(solver='liblinear', max_iter=2000)
             LogR.fit(X_train,Y_train)

             SVM_score = svclassifier.score(X_test,Y_test)
             LogR_score = LogR.score(X_test,Y_test)
             SVM+=SVM_score/10.0
             LogRs+=LogR_score/10.0
         print(SVM)
         print(LogRs)

         0.9996999999999999
         0.9996999999999999
```

- Dataset2 - CelebA low Q:

Number of Samples	SVM	Logistic Regression	KNN
2000	95.75%	92%	60%

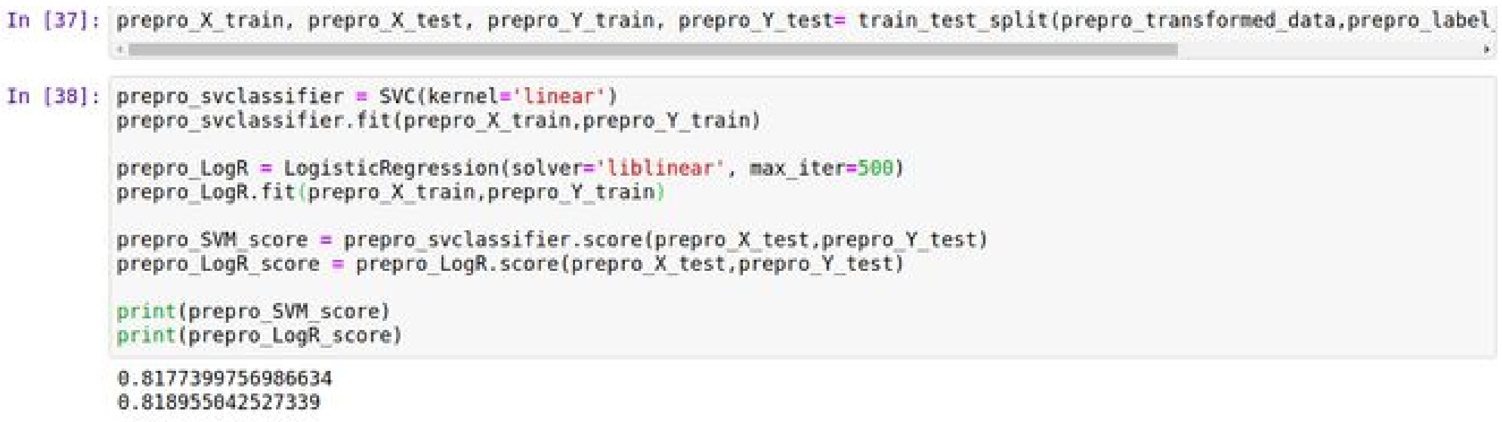


```
In [18]: celeb_X_train, celeb_X_test, celeb_Y_train, celeb_Y_test= train_test_split(celeb_transformed_data,celeb_label_data,
```

```
In [20]: celeb_svclassifier = SVC(kernel='linear')
         celeb_svclassifier.fit(celeb_X_train,celeb_Y_train)

         celeb_svclassifier_r = SVC(C=6.37, kernel='rbf', gamma=0.86)
         celeb_svclassifier_r.fit(celeb_X_train, celeb_Y_train)

         celeb_svclassifier_p = SVC(kernel='poly')
         celeb_svclassifier_p.fit(celeb_X_train, celeb_Y_train)

         celeb_LogR = LogisticRegression(solver='liblinear', max_iter=1000)
         celeb_LogR.fit(celeb_X_train,celeb_Y_train)

         celeb_SVM_score = celeb_svclassifier.score(celeb_X_test,celeb_Y_test)
         celeb_SVM_r_score = celeb_svclassifier_r.score(celeb_X_test,celeb_Y_test)
         celeb_SVM_p_score = celeb_svclassifier_p.score(celeb_X_test,celeb_Y_test)
         celeb_LogR_score = celeb_LogR.score(celeb_X_test,celeb_Y_test)

         print(celeb_SVM_score)
         print(celeb_SVM_r_score)
         print(celeb_SVM_p_score)
         print(celeb_LogR_score)

         0.9575
         0.9975
         1.0
         0.92
```

- Dataset3- Prepro Dataset:

Number of Samples	SVM	Logistic Regression
4115	81.77%	81.89%



```
In [37]: prepro_X_train, prepro_X_test, prepro_Y_train, prepro_Y_test= train_test_split(prepro_transformed_data,prepro_label_

In [38]: prepro_svclassifier = SVC(kernel='linear')
         prepro_svclassifier.fit(prepro_X_train,prepro_Y_train)

         prepro_LogR = LogisticRegression(solver='liblinear', max_iter=500)
         prepro_LogR.fit(prepro_X_train,prepro_Y_train)

         prepro_SVM_score = prepro_svclassifier.score(prepro_X_test,prepro_Y_test)
         prepro_LogR_score = prepro_LogR.score(prepro_X_test,prepro_Y_test)

         print(prepro_SVM_score)
         print(prepro_LogR_score)

         0.8177399756986634
         0.818955042527339
```

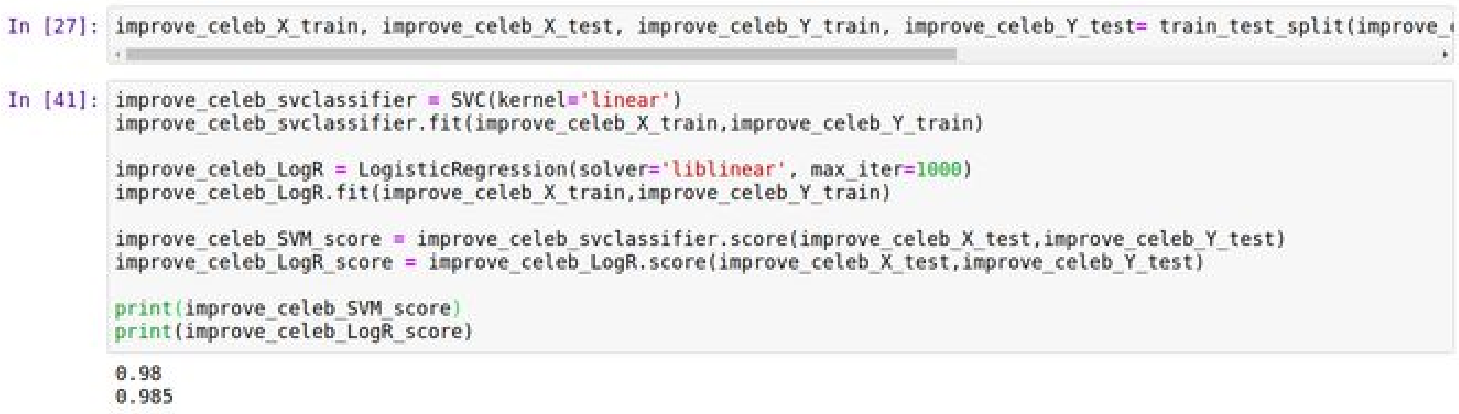
- **Our Method results:**

  - Dataset1- Faces HQ:

Number of Samples	SVM	Logistic Regression
5000	100%	100%

  - Dataset2 - CelebA low Q:

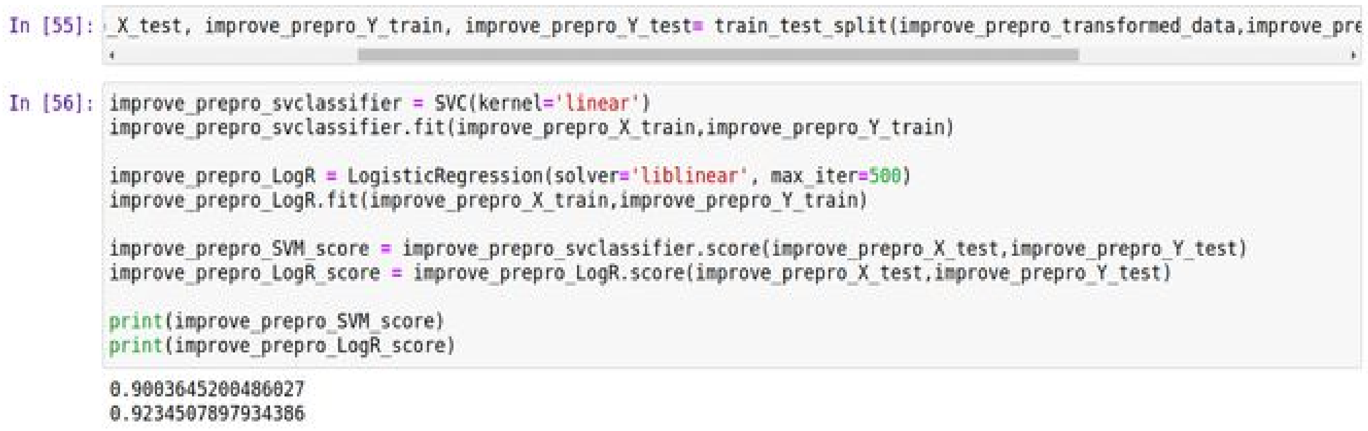
Number of Samples	SVM	Logistic Regression
2000	98%	98.5%



```
In [27]: improve_celeb_X_train, improve_celeb_X_test, improve_celeb_Y_train, improve_celeb_Y_test= train_test_split(improve_

In [41]: improve_celeb_svclassifier = SVC(kernel='linear')
         improve_celeb_svclassifier.fit(improve_celeb_X_train,improve_celeb_Y_train)

         improve_celeb_LogR = LogisticRegression(solver='liblinear', max_iter=1000)
         improve_celeb_LogR.fit(improve_celeb_X_train,improve_celeb_Y_train)

         improve_celeb_SVM_score = improve_celeb_svclassifier.score(improve_celeb_X_test,improve_celeb_Y_test)
         improve_celeb_LogR_score = improve_celeb_LogR.score(improve_celeb_X_test,improve_celeb_Y_test)

         print(improve_celeb_SVM_score)
         print(improve_celeb_LogR_score)

         0.98
         0.985
```

- Dataset3- Prepro Dataset:

Number of Samples	SVM	Logistic Regression
4115	90.03%	92.34%



```
In [55]: _X_test, improve_prepro_Y_train, improve_prepro_Y_test= train_test_split(improve_prepro_transformed_data,improve_pre

In [56]: improve_prepro_svclassifier = SVC(kernel='linear')
         improve_prepro_svclassifier.fit(improve_prepro_X_train,improve_prepro_Y_train)

         improve_prepro_LogR = LogisticRegression(solver='liblinear', max_iter=500)
         improve_prepro_LogR.fit(improve_prepro_X_train,improve_prepro_Y_train)

         improve_prepro_SVM_score = improve_prepro_svclassifier.score(improve_prepro_X_test,improve_prepro_Y_test)
         improve_prepro_LogR_score = improve_prepro_LogR.score(improve_prepro_X_test,improve_prepro_Y_test)

         print(improve_prepro_SVM_score)
         print(improve_prepro_LogR_score)

         0.9003645200486027
         0.9234507897934386
```

Conclusion:

Improvement:

- **Dataset-1:** It remained 100% accurate with both implementations.

- **Dataset-2:** Our version improved accuracy to 99% with Logistic Regression and to 98 % with SVM.(Improvement of 7- 8% in accuracy).
- **Dataset-3:** Our version improved accuracy to 90% with SVM and to 92% with Logistic Regression.(Improvement of 10-12% in accuracy).

Logistic Regression took an edge over SVM in accuracy. This might be because the feature vector is **non -linear**. SVM tries to find the best hyperplane that divides two classes. Logistic Regression uses non-linear classifiers .

Why..?

Logistic Regression produces probabilistic values while SVM produces 1 or 0. So in a few words LR makes no absolute prediction and it does not assume data is enough to give a final decision. This may be a good property when what we want is an estimation or we do not have high confidence into data.

Analysis based on ELTP(Enhanced Local ternary Pattern) is proved to be better than Classical Frequency analysis based on the results obtained. This is because ELTP properties give the boundary of forgery in an image when forging is done using image splicing.

References:

- https://amity.edu/icactm/Proceeding/Paper%20Index%20Content/24%20T4%20P11%20ID%20221.pdf
- Detailed explanation of how to calculate LBP: https://en.wikipedia.org/wiki/Local_binary_patterns
- What is Chrominance: https://en.wikipedia.org/wiki/Chrominance
- https://www.sciencedirect.com/science/article/pii/S2095809917307890
- https://github.com/cc-hpc-itwm/DeepFakeDetection
- https://shodhganga.inflibnet.ac.in/bitstream/10603/49510/10/10_chapter%205.pdf