Digital Image Processing – 1

Mini-Project

Title: Currency detection and verification using image processing

Under the guidance of

Dr. Shruthi M.L.J.

Team Members:

Arya R Adkoli (PES1UG20EC034)

Aryan R (PES1UG20EC036)

SEM: V 'A'

Introduction

In today's world, where image manipulation has reached new heights, it has become very difficult to identify the authenticity of a currency note, which is very important for the functioning of society. In this project we try to use all the basic ideas of Image processing to find and extract certain features of a real and a fake note, compare them and finally determine whether the note is fake or real.

Currently, the Indian government has issued the following currency notes for circulation:

1) 10 Rupees
2) 20 Rupees
3) 50 Rupees
4) 100 Rupees
5) 200 Rupees
6) 500 Rupees
7) 2000 Rupees

In this project, we have created a model for the 10,20,50,200 and 500 rupees note for convenience.

This project is designed specifically for Indian bank notes, but it can also be easily modified to accommodate currencies of other countries. We have extracted all the basic security features of the note as mentioned by the RBI in their guidelines. We have used many Image processing topics such as:

1) Image acquisition
2) Image pre-processing
3) Image enhancement
4) Image restoration
5) Color image processing
6) Image recognition

This project was done using Python and the OpenCV library, using inbuilt features like filters, color mode conversion, and most importantly the orb algorithm.

Theoretical Details

1) **Pixel:** This pixel is a point on the image that takes on a specific shade, opacity or color. It is usually represented in one of the following:

   - Greyscale - A pixel is an integer with a value between 0 to 255 (0 is completely black and 255 is completely white).
   - RGB - A pixel is made up of 3 integers between 0 to 255 (the integers represent the intensity of red, green, and blue).

2) Image Acquisition: Image acquisition is the first step in image processing. This step is also known as preprocessing in image processing. It involves retrieving the image from a source, usually a hardware-based source.
3) Image Pre-processing: Image pre-processing is the term for operations on images at the lowest level of abstraction. These operations do not increase image information content but they decrease it if entropy is an information measure.
4) Image Enhancement: Image enhancement is the process of bringing out and highlighting certain features of interest in an image that has been obscured. This can involve changing the brightness, contrast, etc.
5) Image Restoration: Image restoration is the process of improving the appearance of an image. However, unlike image enhancement, image restoration is done using certain mathematical or probabilistic models.
6) Color Image Processing: Color image processing includes a number of color modeling techniques in a digital domain. This step has gained prominence due to the significant use of digital images over the internet.
7) Image Recognition: Image recognition refers to technologies that identify places, logos, people, objects, buildings, and several other variables in digital images.

Image Processing topics used

1) Median Filter: The median filter is the filtering technique used for noise removal from images and signals. Median filter is very crucial in the image processing field as it is well known for the preservation of edges during noise removal.

2) Binary Filter: Used for the erosion or dilation of objects, removal of noise in an image, detect edges, smoothing the image, and can also be used for area measurement.

3) Canny Filter: The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.

4) ORB Algorithm: ORB is basically a fusion of FAST key point detector and BRIEF descriptor with many modifications to enhance the performance. First it uses FAST to find key points, then apply Harris corner measure to find top N points among them. It also uses pyramid to produce multiscale-features.

5) HSV (Hue, Saturation and Value): It stores color information in a cylindrical representation of RGB color points. It attempts to depict the colors as perceived by the human eye. Hue value varies from 0-179, Saturation value varies from 0-255 and Value varies from 0-255. It is mostly used for color segmentation purpose.

Algorithm

Step 1: Resize the input image from the user to a standard size of (1080, 512) for easier computation.

Step 2: Select a particular region of the note and detect its hue value. This value will help to determine the denomination of the currency note.

Step 3: Apply Median filter to remove the noise in the input image and preserve the edges of the image.

Step 4: Convert the image from RGB to Grayscale.

Step 5: Convert the image to Binary image and threshold it to (lower limit = 150, upper limit = 250).

Step 6: Apply Canny filter to detect the edges and display it.

Step 7: Repeat the above steps for the anchor image for comparison purposes.

Step 8: Compare the input image with the anchor image using the ORB algorithm and determine the authenticity of the image.

Code

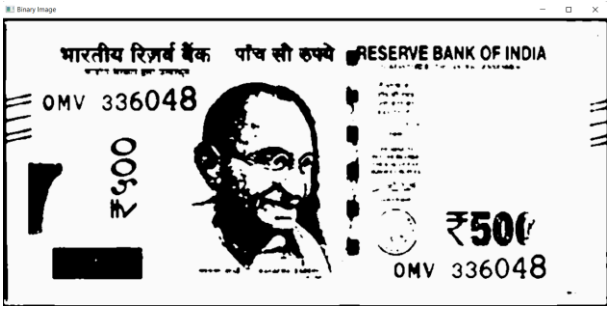
```
import cv2

fileName = input("Enter file location of the note: ")
note = cv2.imread(fileName)
cv2.imshow('Input image', note)
cv2.waitKey(0)
comparision_note = ''


# Function which detects the value of the currency note based on the hue value of
the input note
def detectCurrency(image):
    h, w, c = image.shape
    hpart = h//6
    wpart = w//6
    RegionForDet = image[:, (3*wpart):(5*wpart)]
    cv2.imshow('Region of note used to detect currency type', RegionForDet)
    cv2.waitKey(0)
    hsv = cv2.cvtColor(RegionForDet, cv2.COLOR_BGR2HSV)
    [h, s, v] = hsv[100, 100]
    print('Hue value in input image is = ', h)
    print('Value of the note is:')
    if h < 16:
        print('10 Rupee note')
        comparision_note = "Real_10.jpeg"
    elif h < 30:
        print("200")
        comparision_note = "Real_200.jpeg"
    elif h < 35:
        print("20")
        comparision_note = "Real_20.jpeg"
    elif h < 50:
        print("500")
        comparision_note = "Real_500.jpeg"
    elif h < 100:
        print("50")
        comparision_note = "Real_50.jpeg"
    else:
```

```
        print("Unfortunately, this model was not trained for this currency
note.")
    return comparision_note, h


# Resizing the image into a standard format to make computation easy
def resiziedImage(img):
    h, w, channels = img.shape
    print('Height and width of the input image are: ', h, w)
    dim = (1080, 512)
    resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
    cv2.imshow('Resized image', resized)
    cv2.waitKey(0)
    print('Dimensions of the resized image are: ', dim)
    return resized


# Applying Median filter to remove noise
def medianFilter(img):
    img = cv2.medianBlur(img, 5)
    cv2.imshow('Median Blur', img)
    cv2.waitKey(0)
    return img


# Converting image to Binary image to verify the note
def binaryImage(img):
    ret, bw_img = cv2.threshold(img, 150, 250, cv2.THRESH_BINARY)
    cv2.imshow("Binary Image", bw_img)
    cv2.waitKey(0)
    return bw_img


# Converting RGB image to Gray scale
def rgbtogray(img):
    img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Gray scale image', img1)
    cv2.waitKey(0)
    return img1


# Using the canny filter to get all the edges in the note
def canny(img):
    edged = cv2.Canny(img, 30, 200)
    cv2.imshow('Image after canny filter', edged)
    cv2.waitKey(0)
```

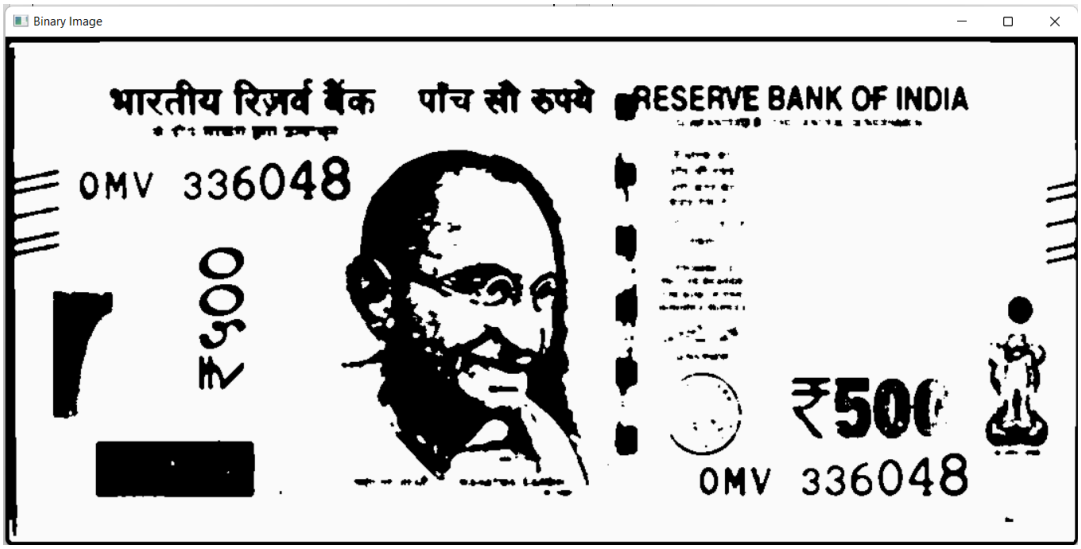
```
    return edged


# Using orb similarity function to get the similarity between input image and the
ideal currency note of the respective value
def orb_sim(img1, img2):
    orb = cv2.ORB_create()
    kp_a, desc_a = orb.detectAndCompute(img1, None)
    kp_a, desc_b = orb.detectAndCompute(img2, None)
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(desc_a, desc_b)
    similar_regions = [i for i in matches if i.distance < 50]
    if len(matches) == 0:
        return 0
    return len(similar_regions) / len(matches)


# Applying all above functions on the input image
img = resiziedImage(note)
comparision_note, hue = detectCurrency(img)
img = medianFilter(img)
img = rgbtogray(img)
img = binaryImage(img)
img = canny(img)

# Applying all above functions on the ideal currency note
real_note = cv2.imread(comparision_note)
img1 = resiziedImage(real_note)
img1 = medianFilter(img1)
img1 = rgbtogray(img1)
img1 = binaryImage(img1)
img1 = canny(img1)

# Finding the similarity between the above two processed images
orb_similarity = orb_sim(img, img1)
print("Similarity using orb = ", orb_similarity)

# Printing the final result
print("Verdict: ")
if orb_similarity < 0.8:
    print("Fake note")
else:
    print("Real note")
```

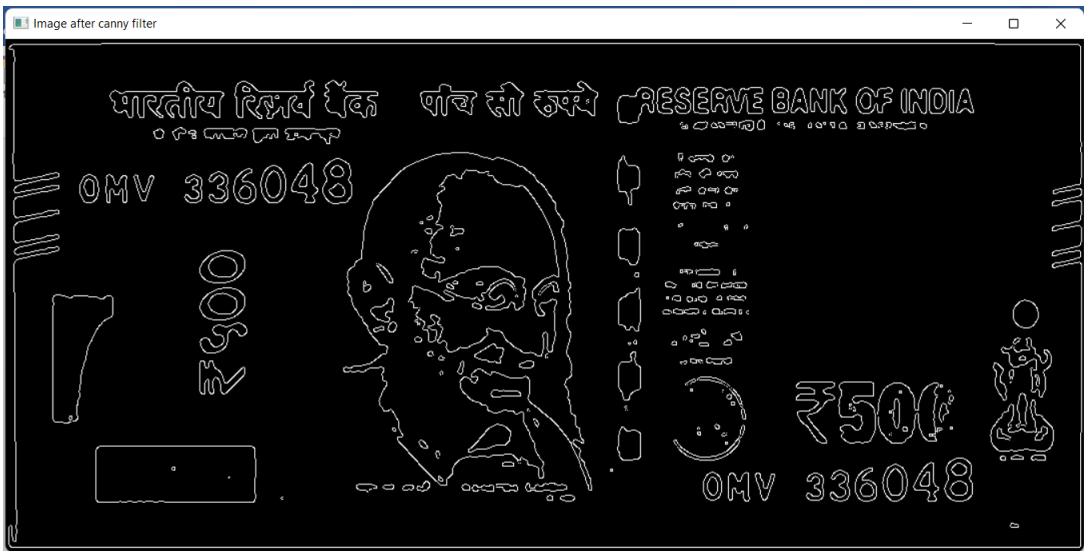
Results

Input image (Fake 500 note):



Anchor image (Real 500 note):





The above 2 images obtained after applying the Canny filter are compared using the ORB algorithm. The ORB function gives the percentage of similarity between two images. If the percentage is greater than 80%, the input currency note image is considered authentic.

The output for the above two images in the terminal is:

```
PS D:\Py_AllPrograms\Notes> python -u "d:\Py_AllPrograms\Notes\Currency_detection.py"
Enter file location of the note: d:\\Py_AllPrograms\\Notes\\Fake_500.jpeg
Height and width of the input image are:  485 1079
Dimensions of the resized image are:  (1080, 512)
Hue value in input image is =  42
Value of the note is:
500
Height and width of the input image are:  485 1079
Dimensions of the resized image are:  (1080, 512)
Similarity using orb =  0.6124401913875598
Verdict:
Fake note
```

Since the canny images are only 61% similar, the input currency note image is fake.