M S RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute Affiliated to VTU)

Department of Information Science and Engineering

A Report on

Extraction of Credit Card Details

Submitted in partial fulfillment of the Mini Project for the subject

Digital Image Processing- IS52A1

Project Team-

USN	Name
1MS15IS085	V.S.PRUTHVI
1MS15IS108	SHARMITHA.S.BYSANI
1MS15IS114	SHREEHARI N.W
1MS15IS135	VISHNUPRIYA

ABSTRACT

An algorithm to detect and analyze credit card details in medium resolution images is presented. Algorithm uses techniques like Black hat morphological process, Laplacian and Otsu's thresholding for segmenting the 4 groupings of the credit card number. These groupings are then used to map the digits with an OCR image. This algorithm will help us identify the type of credit card from the number extracted after the mapping the digits. Python is used to implement the algorithm which takes approx 200ms to execute an image.

INTRODUCTION

The presence of cameras in cell phones these days is becoming extremely main-stream.As more and more people are equipped with these cameras it becomes feasible to develop a range of applications that utilize the camera for purposes other than simply taking a snapshot. One such application is to use the camera to take images of credit cards and extract as much data as possible by taking an image of a credit card we can extract credit card number, CVV, validity date etc. This data can further be processed by the information gathered from the image of the credit card. To achieve such an application, the camera must be able to take images with decent resolution. Even though the aim of this project is to process low resolution images, a decent level of quality is expected. With the help of this image, various algorithms are applied to extract the credit card image among-st the other objects in the background, obtaining card type etc. This report details the development and implementation of an algorithm that can be used for this purpose. The report analyses the problem and establishes the requirements for the algorithm. It then examines the major steps of the algorithm and how it meets the requirements. The results of testing the algorithm are discussed before the report concludes with a summary.

PROBLEM STATEMENT

In the year after demonetization, digital transactions have grown considerably. Indeed, disruptions in the digital space have not only revolutionized the way we manage our finances, they have also made cashless transactions the preferred choice of many among us. Digital wallets and credit/debit cards have been the alternative to cash after demonetization to cope with the cash crunch.

Cash payment constitutes 97 percent of the total payment. Government has set a target of Rs 2,500 crore on digital transactions in FY 2018 (Rs 625 Crores in the first quarter of which merely 37 percent was met). According to a recent report on Digital Payment in India 2020 by Google, the total payment via digital instruments is expected to touch $500 billion by 2020.Digital payments have a huge potential.

India has the third-largest internet user base in the world with 300 million users. Out of which, nearly 50 percent of them are mobile-only internet users. Over 93% of people in rural India have not done any digital transactions. So the real potential lies there. Companies like Paytm, Mobikwik etc. are gaining new users, but people aren't aware about how to add money to

their wallets easily. It is a tough task to them to fill all the details on their debit cards to the required fields in the payment option of the wallets as they are unaware of the various details on the credit card.

This automation system provides the user a one click facility to enter all his details in the required fields easily by just clicking the photo or choosing the photo from the gallery. Adding a small encrypting system to this automation system makes it reliable in all the ways. Thus, encouraging the people in rural India to opt for digital mode of payment for better life and economic system of the country.

OBJECTIVE

- Extracting credit card from the background regardless of the texture of the background and regardless of overall quality and resolution of the image.
- Recognition of credit card number, expiry date or validity, Name of the card holder and CVV.
- Analyzing the credit card type from the credit card number extracted.
- Application of the extracted data in various domains like payment gateways by auto filling the required fields like card number, CVV, card holder name and validity.
- The objective of this project is mainly to extract important and essential information from a credit card. As input is an image of credit card above mentioned steps like extraction, recognition analysis are performed sequentially.

REQUIREMENTS SPECIFICATION

❖ Technical Requirements

- **Minimum Hardware Requirements**
  - Intel(R) Pentium(R) D CPU 2.66 GHz processor
  - 4 GB RAM
  - 250 GB HDD
  - Windows 7 64-bit

- **Minimum Software Requirements**
  - Python IDE
  - Packages:-
    - Imutils
    - Numpy
    - Opencv2

❖ Functional Requirements of the system:

We have classified these functional requirements as follow:

- Taking/ choosing the desired credit card.
- Recognition of the credit card.
- Extracting the card details for different uses.

1. Taking/ choosing the desired image:

- Description: The most important thing here is the use of an Android mobile phone and its camera. The user can take a picture of a credit card or choose one from the mobile's directory.
- The user must use a camera of typical resolution and take a picture of a credit card or choose one from existing ones in his phone.

2. Recognition of the Card:

- Description: The card will be recognized from the image taken by the mobile's camera or from any chosen image from the phone directory.

- The card will be recognized and ready to be used for the following
    - Extracting the card number, expiry date, card holder name, CVV.
    - Analysing the card type and best available payment gateways.

3. Extracting the card details for different uses:

- Description: Once the card is recognized and ready to be used to extract the the card number, expiry date, card holder name, CVV. Algorithm will extract the details and provide the analysis of the details extracted.
- This can be stored in a database and can be given a nickname which makes the further transactions easy for the user by typing the nickname in his future transactions.

❖ Non-Functional requirements:

1. Product Requirements:

- **Usability Requirements:** The application shall be used friendly and doesn't require any guidance to be used. In other words, the application has to be as simple as possible, so its users shall use it easily. Actually, the interface is quite simple and straight forward so that anyone can understand it. The user should only provide a picture of his/her credit card.

- **Reliability Requirements:** The application should not have any unexpected failure. In order to avoid any failure's occurrence, the specifications have been respected and followed correctly. The only problem that may occur in some cases
  (Eg: details may not be clearly visible) is that the application does not get 100% of the details of the card.

2. Efficiency Requirements:

➢ **Performance:** The application response time shall be adequate and sufficient enough, that's why the time required for this application to response to its user's actions has to been managed and controlled. But in order to maintain the performance of the application, the user has to follow the required steps to get the desired result.

➢ **Portability Requirements:** The application should be compatible with different version of OpenCV and python, so if the version of OpenCV or python is upgraded, the application should be upgraded as well.

3. External Requirements:

➢ **Ethical Requirements:** This application should protect the confidentiality of the user's personal information and any personal data stored on his\her credit card image.

➢ **Legislative Requirements:**

- **Security:** The security signature and certificate of the application is required as in any application.
- **Privacy:** The application shall protect the user's data and make sure to keep it confidential.

METHODOLOGY & FLOW-CHARTS

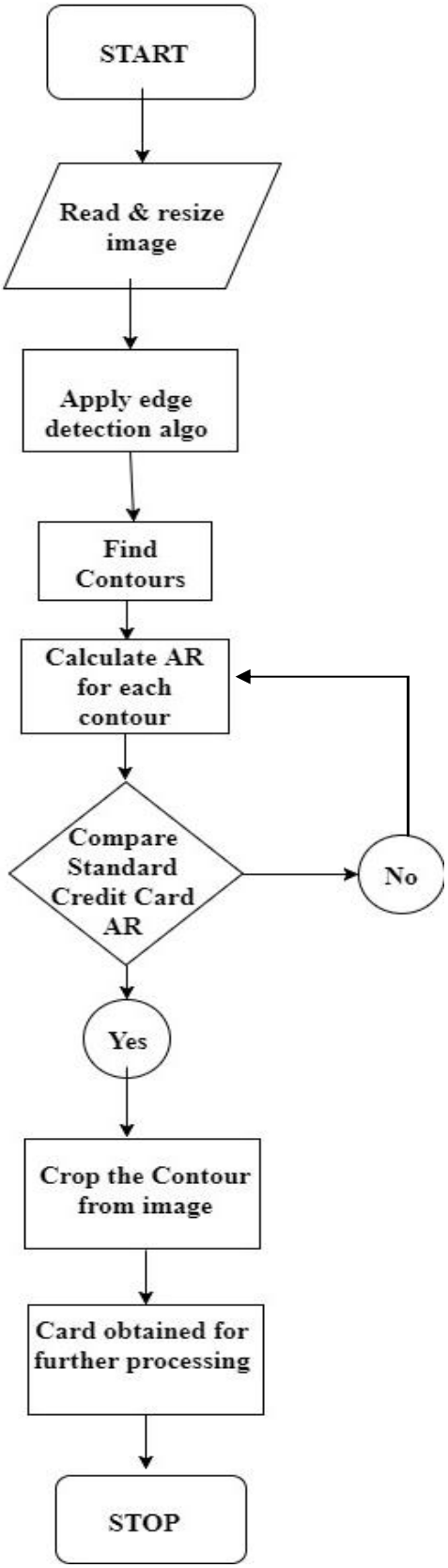

Fig 1: Flow Chart for extracting credit card from background.

STEP 1: The end-user captures a picture of his credit card and feeds it as input to the credit card reading system. This image would consist of additional unnecessary background details, which must be removed and only the cropped credit card image is required. Firstly, we read this image captured by the user. In order to detect the edges in the image, we apply an edge detection algorithm like Canny Edge Technique. Next, the contours in the image are detected. The boundaries or the region of interest of all the objects in the image are returned. The Aspect Ratio (AR) is calculated for all the contours. Among the returned contours, the aspect ratios are compared with the aspect ratio of a credit card as all credit cards have same aspect ratio. The contour having aspect ratio equal to the aspect ratio of the credit card is cropped out from the original image. Now this cropped out image consisting of only the credit card retrieved from the original image is utilized for processing in further steps.

STEP 2: For extracting the details of the credit card, we require a reference OCR image with font similar to the font on the credit card. This reference image is inputted and is gray scaled. Next we will need to extract the individual digits from the OCR reference image. This is done by first applying an edge detection algorithm like Binary Inverse thresholding. To obtain the outlines of the digits in the reference image, we apply the technique of contouring. The contours obtained are looped which gives the ROI's for each digit of the reference image which is stored in a dictionary.

STEP 3: The output cropped credit card image from step 1 is the input image for this step. The initial task is to resize it to a standard or predefined size and grayscale the image. For efficient and precise digit extraction, only the portion of the card containing the credit card number is masked out using bitwise operations. To this masked image, morphological transformations like tophat or blackhat operators are applied to obtain the light regions of the image from the dark background (i.e., to separate credit card number from background). Scharr or Laplacian Gradient technique is applied to the image obtained after morphological operations. Next, the image is subjected to a suitable thresholding method in order to binarize the image.To recognise the four digit groupings of the credit card number, we apply contouring , sort them , loop and store the ROI's of the 4 digit groupings in a list. This list of four digit groupings is subjected to a for loop where we again apply thresholding and contouring techniques to obtain the individual digit ROI's and store them in a list or dictionary.

Now, using the match template property, we try to match the ROI's of the OCR digits in the dictionary obtained from step 2 with the ROI's of the credit card number digits obtained in this step. On comparing these two dictionaries, the function returns a series of scores (one for each digit 0-9), the maximum score is considered as this would score would represent the correctly identified digit. This score is obtained from a specific index whose integer name represents the most-likely digit based on the comparisons to each template.

Finally, we have obtained the credit card number using which we can analyze the type of credit card based on the first number of the card. The same algorithm can be applied for card holder name, expiry date and CVV extraction.

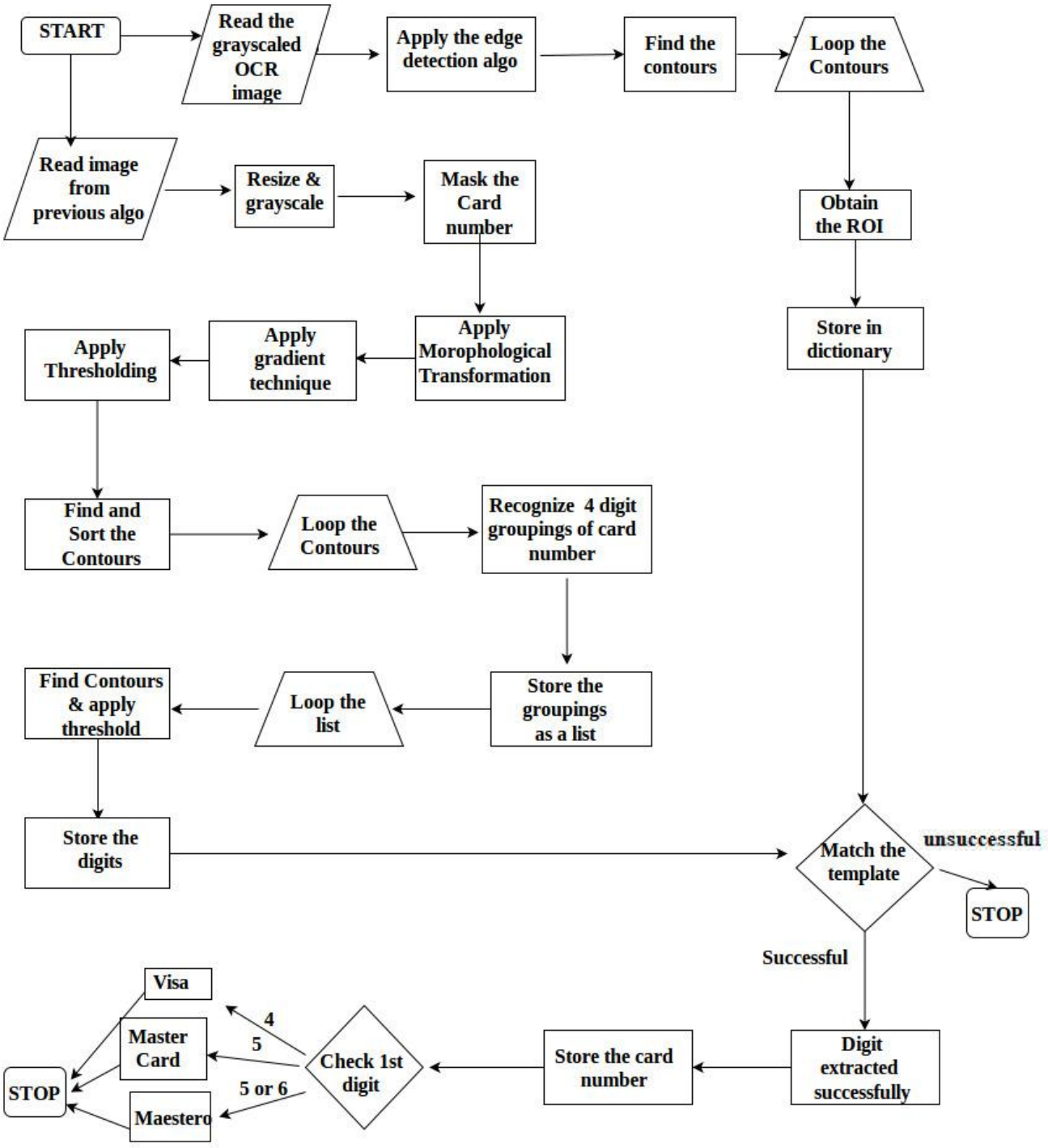

Fig 2: Flow Chart for extracting the credit card details.

Detailed explanation of the functions used during implementation.

➢ Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image and second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation, which has different variants like Opening, Closing, Gradient etc.

❖ Top Hat Operator

It is the difference between Input image and Opening of the image. Below is an example for a 9x3 rectangular and 5x5 square kernel.

```
rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))

sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

tophat = cv2.morphologyEx(card_gray, cv2.MORPH_TOPHAT, rectkernel)
```

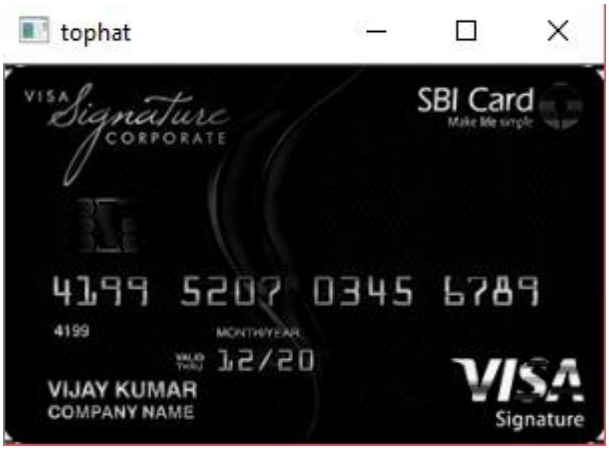

Fig 3: Credit card image after applying tophat morphological transformation.

❖ Black Hat Operator

It is the difference between the closing of input image and the original input image.

```
blackhat = cv2.morphologyEx(card_gray, cv2.MORPH_BLACKHAT,  rectKernel)
```

Fig 4: Credit card image after applying Blackhat morphological transformation.

➢ Image Gradients

❖ Sobel and Scharr Derivatives

Sobel operators is a joint Gaussian smoothing plus differentiation operation, making it is more resistant to noise. The direction in which the derivatives need to be calculated, that is, vertical or horizontal must be explicitly mentioned by the arguments of the gradient function (x-order and y-order).The size of the kernel denoted by k size can also be specified in the arguments list. A k size value of -1 indicates the use of a 3x3 Scharr filter which gives better results than a 3x3 Sobel filter.

```
gradX = cv2.Sobel(card_gray, ddepth=cv2.CV_32F, dx=1, dy=0,ksize=-1)
```

Fig 5: Credit card image after applying Scharr/Sobel gradient.

❖ Laplacian Derivatives

This derivative calculates the Laplacian of the image given by the relation, $\Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$, where each derivative is found using Sobel derivatives. If ksize = 1, then following kernel is used for filtering:

$$kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
gradX = cv2.Laplacian(card_gray,cv2.CV_64F)
```

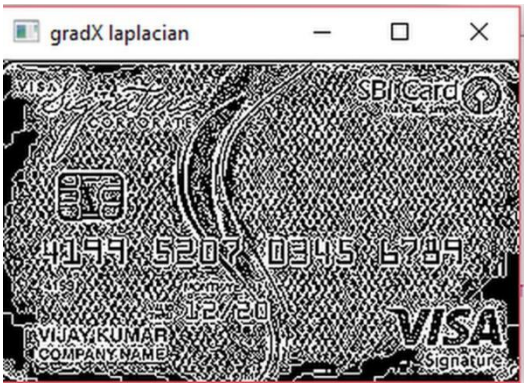

Fig 6: Credit card image after applying Laplacian gradient.

➢ Canny Edge Detection

This edge detection algorithm is a multi-stage algorithm and which goes through the following stages:

- Noise Reduction
- Finding Intensity Gradient of the Image
- Non-maximum Suppression
- Hysteresis Thresholding

First argument for the canny edge function is the input image. Second and third arguments are the minVal and maxVal values respectively. Third argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is L2gradient which specifies the equation for finding gradient magnitude. Mentioning the third and fourth arguments are not mandatory. In the example shown below, the arguments specified in the function are the input image followed by the minVal and maxVal values.

thresh = cv2.Canny(card_gray,100,200)

Fig 7: Credit card image after applying Canny Edge Detection Algorithm.

➢ Image Thresholding

❖ Simple Thresholding

Thresholding is considered as the simplest method of image segmentation. From a grayscaled image, the thresholding technique can be used to create binary images, If a pixel value is greater than a threshold value, it is assigned a particular value (say 255 indicating white), else it is assigned another value (say 0 indicating black). The function used is cv2.threshold. First argument for the function is the source image, which should be a gray scaled. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. Different styles of thresholding defined as the fourth parameter of the

function can be classified as follows:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

Two outputs are obtained. First one is a retVal which returns the threshold value. Second output is the thresholded image.

threshold = cv2.threshold(card_gray, 180, 255, cv2.THRESH_BINARY_INV)

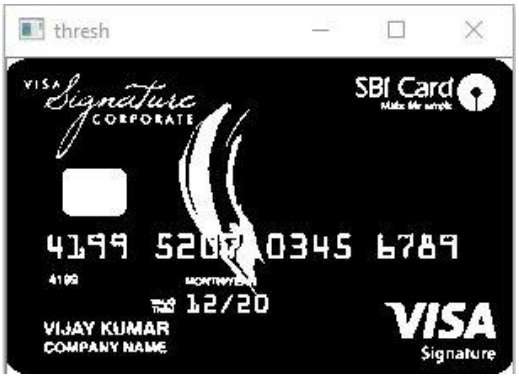

Fig 8: Credit card image after applying Binary threshold.

❖ Adaptive Thresholding

We always use a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In such cases, we use for adaptive thresholding. Here, the algorithm calculates the threshold for a small region of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination. It has three 'special' input parameters and only one output argument.

Adaptive Method - It decides how thresholding value is calculated.

- cv2.ADAPTIVE_THRESH_MEAN_C : threshold value is the mean of neighborhood area.
- cv2.ADAPTIVE_THRESH_GAUSSIAN_C : threshold value is the weighted sum of neighborhood values where weights are a Gaussian window.
-

Block Size - It decides the size of neighborhood area.

C - It is just a constant which is subtracted from the mean or weighted mean calculated.

```
thresh=cv2.adaptiveThreshold(gradX,255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 115, 1)
```

Fig 9: Credit card image after applying Adaptive Gaussian Threshold.

❖ Otsu's Binarization

The retVal parameter returned by the threshold function is used in Otsu's Binarization. In global thresholding, we used an arbitrary value for threshold. A bimodal image is an image whose histogram has two peaks. Otsu's binarization approximately takes a value in the middle of these two peaks as the threshold value. For non-bimodal images, this binarization technique will not be accurate. We use the same cv2.threshold() function with an extra flag cv2.THRESH_OTSU. Zero is passed as the threshold value. The algorithm then finds the optimal threshold value and returns as the second output, retVal. If Otsu thresholding is not used, then retVal is same as the threshold value is used.

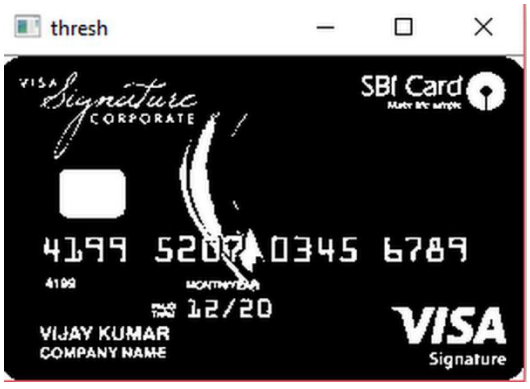

Fig 10: Credit card image after applying Otsu's Threshold.

- **Summary table after applying various combinations of gradients, morphological transforms and thresholds on the credit card image.**

CARD1	CARD2	CARD3	TOPHAT	BLACKHAT	SCHAAR	LAPLACIAN	OTSU	ADAPTIVE-GAUSSIAN	CANNY EDGE
■□■■	□□□□	■□■□	✓		✓			✓	
■■■■	■■■■	■□■□	✓		✓		✓		
■■■■	■■■■	■■■□	✓			✓	✓		
■□■■	■■■■	■□□□	✓			✓		✓	
■□■■	■■■■	■■■■		✓	✓		✓		
■□■■	■■■■	■■■□		✓	✓			✓	
■■■■	■■■■	■■■■		✓		✓	✓		
■□□■	■■■■	■■□□		✓		✓		✓	
■■□□	■■■■	■■□□		✓		✓			✓
■■■■	□□□■	■■■■		✓	✓				✓
■□■■	□■■■	■■■■	✓		✓				✓
■□■■	■■■■	□■■□	✓			✓			✓

Few Snapshots:-

Combination	Card 1	Card 2	Card 3
BlackHat Laplacian Canny Edge			

SELECTED COMBINATION

BlackHat Laplacian Otsu			
BlackHat Scharr Canny Edge			
TopHat Laplacian Canny Edge			
TopHat Scharr Adaptive Gaussian			

Why we chose the selected combination ?

From the above experimentation we conclude that Blackhat morphological transformation, Laplacian gradient and Otsu thresholding is able to detect all the 4 groupings of the card number perfectly. Therefore, we are using this combination of operations to detect the groupings of the card.

CODE:-

```
from imutils import contours
import numpy as np
import argparse
import imutils
import cv2
from credit_card_reco_1 import card_detection


#reading the images
image = cv2.imread("blue.png")
im1=card_detection(image)
card = cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY)
card_copy=im1


#dictionary used to find card type
FIRST_NUMBER = {
"2": "Visa",
        "3": "American Express",
        "4": "Visa",
        "5": "MasterCard",
        "6": "Maestro"
}


im=card_copy.copy()
ref2 = cv2.imread('ref2.jpg')

#resizing the input images
card = imutils.resize(card, width=300)
card_copy = imutils.resize(card_copy, width=300)
im = imutils.resize(im, width=300)

#masking and grayscaling the images
mask = np.zeros(card.shape, np.uint8)
mask[97:125, 0:400] = 255
card_gray = cv2.bitwise_and(card,card,mask = mask)
ref2 = cv2.cvtColor(ref2, cv2.COLOR_BGR2GRAY)

#thresholding the reference image
retval,ref2_threshold = cv2.threshold(ref2, 180, 255, cv2.THRESH_BINARY_INV)

#finding contours for refernece image
refCnts = cv2.findContours(ref2_threshold.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
refCnts = refCnts[0] if imutils.is_cv2() else refCnts[1]
refCnts = contours.sort_contours(refCnts, method="left-to-right")[0]
```

```
digits2 = {}

# loop over the OCR-A reference contours
for (i, c) in enumerate(refCnts):
    (x, y, w, h) = cv2.boundingRect(c)
    roi = ref2_threshold[y:y + h, x:x + w]
    roi = cv2.resize(roi, (57, 88))
     # update the digits dictionary, mapping the digit name to the ROI
    digits2[i] = roi

cv2.imshow('card_gray',card_gray)
cv2.waitKey()
cv2.destroyAllWindows()

rectKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 3))
sqKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

blackhat = cv2.morphologyEx(card_gray, cv2.MORPH_BLACKHAT, rectKernel)

cv2.imshow('tophat',blackhat)

#laplacian
gradX = cv2.Laplacian(blackhat,cv2.CV_64F)

gradX = np.absolute(gradX)
(minVal, maxVal) = (np.min(gradX), np.max(gradX))
gradX = (255 * ((gradX - minVal) / (maxVal - minVal)))
gradX = gradX.astype("uint8")

cv2.imshow('gradX laplacian',gradX)

#applying morph closing to close gaps between credit card numbers
gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKernel)

# otsu threshold
thresh = cv2.threshold(gradX, 0, 255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

#morph close to close gaps between credit card number regions
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, sqKernel)
cv2.imshow('thresh',thresh)

#apply contours ro find four digit groupings on the credit card
cnts = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
locs = []
r=[]

for (i, c) in enumerate(cnts):
    (x, y, w, h) = cv2.boundingRect(c)
    ar = w / float(h)  #aspect ratio calculation
```

```
    if ar > 2.5 and ar < 4.0:
        if (w > 40 and w < 55) and (h > 10 and h < 20):
            locs.append((x, y, w, h))
            r.append(i)
    cv2.rectangle(card_copy, (x,y), (x+w,y+h), (0, 255, 0), 2)

cv2.imshow('card_copy',card_copy)
#cv2.imshow('card',card)

locs = sorted(locs, key=lambda x:x[0])
print("locs=",locs)
digits_card={}

output = []
# loop over the 4 groupings of 4 digits
for (i, (gX, gY, gW, gH)) in enumerate(locs):
            # initialize the list of group digits
            groupOutput = []

            # extract the group ROI of 4 digits from the grayscale image,
            # then apply thresholding to segment the digits from the
            # background of the credit card
            group = im[gY - 5:gY + gH + 5, gX - 5:gX + gW + 5]
            group=  cv2.resize(group,(250,200))
            group = cv2.cvtColor(group, cv2.COLOR_BGR2GRAY)

            group = cv2.adaptiveThreshold(group, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 115, 1)
            mask1 = np.zeros(group.shape, np.uint8)
            mask1[30:150, 20:230] = 255
            group= cv2.bitwise_and(group,group,mask = mask1)

            cv2.imshow('groups',group)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

            digitCnts = cv2.findContours(group.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
            digitCnts = digitCnts[0] if imutils.is_cv2() else digitCnts[1]
            digitCnts = contours.sort_contours(digitCnts,method="left-to-right")[0]

            for (j,c) in enumerate(digitCnts):
                (x, y, w, h) = cv2.boundingRect(c)
                roi_card = group[y:y + h, x:x + w]
                roi_card = cv2.resize(roi_card, (57, 88))

                digits_card[j]=roi_card

                # initialize a list of template matching scores
                scores = []
```

```
                # loop over the reference digit name and digit ROI
                for (ocr_digit, ocr_digitROI) in digits2.items():
                    # apply correlation-based template matching, take the
                    # score, and update the scores list
                    result = cv2.matchTemplate(roi_card, ocr_digitROI,cv2.TM_CCOEFF)
                    (_, score, _, _) = cv2.minMaxLoc(result)
                    scores.append(score)

                # the classification for the digit ROI will be the reference
                # digit name with the *largest* template matching score
                groupOutput.append(str(np.argmax(scores)))
                # draw the digit classifications around the group
            cv2.rectangle(card_copy, (gX - 5, gY - 5),(gX + gW + 5, gY + gH + 5), (0, 0, 255), 2)
            cv2.putText(card_copy, "".join(groupOutput), (gX, gY - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 2)

            # update the output digits list
            output.extend(groupOutput)

cv2.imshow('card_copy',card_copy)
cv2.waitKey()
cv2.destroyAllWindows()

# display the output credit card information to the screen
print("CREDIT CARD NUMBER #: {}".format("".join(output)))
print("CREDIT CARD TYPE: {}".format(FIRST_NUMBER[output[0]]))
```

Fig 11: Final output

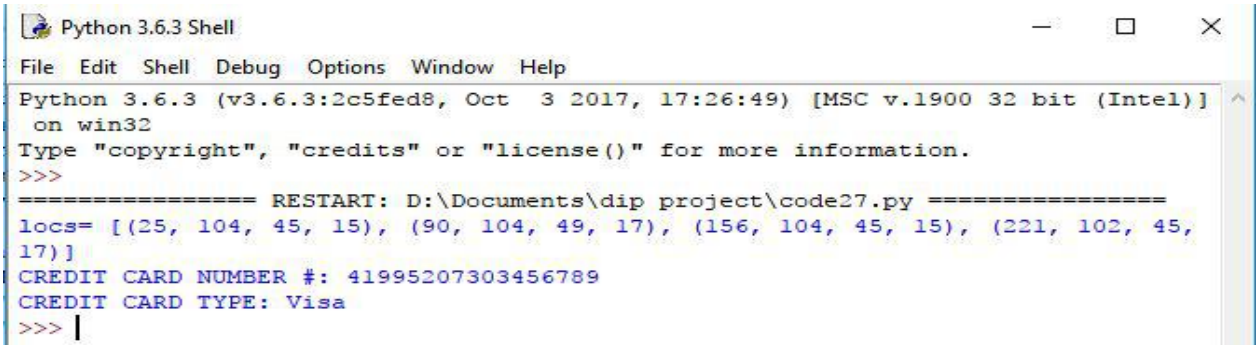

Fig 12: Final output on terminal

CONCLUSION

This report has detailed the development and implementation of an algorithm to detect credit card details in images taken from cell phone cameras. The algorithm applies a canny edge detection algorithm to segment the image into objects and background. It then classifies the objects based on their aspect ratios and crops the credit card from the given input image. Black hat morphological transformation, Laplacian gradient and Otsu threshold is applied to segment the 4 digit groupings on the card. Digits are extracted from the groupings by applying adaptive threshold over the contours of the group. The digits are mapped to a referenced OCR image to find the card number. Similar technique is used to find the expiry date, name, CVV, etc. A succession of tests is then applied to filter out false positives before sampling the image to obtain the data. The algorithm has been designed to be invariant to resolution. Testing has been performed with four test images, the first 2 images were accurate and rest 2 images were partially right. The algorithm correctly identified all of the details without error. The algorithm has primarily been implemented in python using OpenCv. The algorithm typically processes a VGA resolution image in 200ms.

REFERENCES

https://www.pyimagesearch.com/

https://stackoverflow.com

https://docs.opencv.org/master/index.html

https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ

https://pythonprogramming.net/loading-images-python-opencv-tutorial/