1. **INTRODUCTION**

The OCR-A font was designed in the late 1960s such that both (1) OCR algorithms at that time and (2) humans could easily recognize the characters The font is backed by standards organizations including ANSI and ISO among others. Despite the fact that modern OCR systems don’t need specialized fonts such as OCR-A, it is still widely used on ID cards, statements, and credit cards. In fact, there are quite a few fonts designed specifically for OCR including OCR-B and MICR E-13B.



**Figure 1.1:** The OCR-A font, originally developed to aid Optical Character Recognition systems

While you might not write a paper check too often these days, the next time you do, you’ll see the MICR E-13B font used at the bottom containing your routing and account numbers. MICR stands for Magnetic Ink Character Recognition code. Magnetic sensors, cameras, and scanners all read your checks regularly.



**Figure 1.2:** The OCR-B font, an alternative to OCR-A

Each of the above fonts have one thing in common - they are designed for easy OCR. For this project, we will make a template matching system for the OCR-A font, commonly found on the front of credit/debit cards.

**1.1 Problem definition**

OCR-A font, a font created specifically to aid Optical Character Recognition algorithms. We’ll then devise a computer vision and image processing algorithm that can localize the four groupings of four digits on a credit card, Extract each of these four groupings followed by segmenting each of the sixteen numbers individually, Recognize each of the sixteen credit card digits by using template matching and the OCR-A font. Finally, we’ll look at some examples of applying our payment cards OCR algorithm to actual images.

**1.2 Related work**

Francium tech blog had worked on a project to extract information, such as Name, ID number and Date of Birth (DOB) from scanned images of Identity Cards such as Passport or National Identity card issued by any country. To do this, they broke down the problem into sub-problems as below:

* Identify Regions of Interest (ROI) containing the required information
* Crop the regions identified above
* OCR on the identified region of interest

While the second and third steps are trivial, we used YOLO for the first step. YOLO — You Only Look Once, is a state-of-the-art, real time object detection system. YOLO is a fully convolutional network with 75 convolutional layers, skip connections and upsampling layers. A convolutional layer with stride 2 is used instead of pooling to downsample the feature maps. YOLO can do **object detection + object classification + multiple object detection** all at the same time. Assuming each information in our ID card as an object, we used YOLO to identify the bounding boxes for these information.

The biggest challenge in any object detection task is to get annotated images. We resized all the images in our dataset to have the same height and width (416 x 416) and used **Label Img**for labelling Name, ID number and DOB and marking the coordinates of the ROI. The annotations (coordinates of bounding box + labels) are saved as an XML file in PASCAL VOC format. We used YOLO in tensorflow to re-trained the last two (convolution) layers with the ID cards dataset, while the previous layers are initialized with the weights from YOLOv2. This allows to use the feature detectors already trained on the large corpus.

YOLO predicts:

* The offsets relative to the top left corner of the grid predicting the object and the dimensions (width & height) of the bounding box. This can be used to obtain the top left and bottom right coordinates of the bounding box.
* Probability that an object is contained in the above bounding box (using sigmoid function)
* Probability that the object belongs to one of the training classes (Name, Member Id and Group here)

Since the information that we require from the ID cards are non-overlapping and mutually exclusive, we preferred to use YOLOv2 which uses softmax for class prediction. The class probabilities from softmax are mutually exclusive. The output from YOLO looks like this:

[{‘label’: ‘Name’,  
 ‘confidence’: 0.57865334,  
 ‘topleft’: {‘x’: 8, ‘y’: 126},  
 ‘bottomright’: {‘x’: 150, ‘y’: 170}},  
 {‘label’: ‘DOB’,  
 ‘confidence’: 0.86376363,  
 ‘topleft’: {‘x’: 0, ‘y’: 271},  
 ‘bottomright’: {‘x’: 217, ‘y’: 308}},  
 {‘label’: ‘ID number’,  
 ‘confidence’: 0.42626178,  
 ‘topleft’: {‘x’: 26, ‘y’: 172},  
 ‘bottomright’: {‘x’: 152, ‘y’: 213}}]

**1.3 Proposed Algorithm**

To overcome the problems in the existing payment forms, we designed a new model for auto filling sensitive details of payment card owner.

**1.3.1 Card number recognition**

In this section, our template matching algorithm with Python + OpenCV to automatically recognize payment card digits. In order to accomplish this, we’ll need to apply a number of image processing operations, including thresholding, computing gradient magnitude representations, morphological operations, and contour extraction. These techniques have been used in other blog posts to [detect barcodes in images](https://www.pyimagesearch.com/2015/11/30/detecting-machine-readable-zones-in-passport-images/) and [recognize machine-readable zones in passport images](https://www.pyimagesearch.com/2015/11/30/detecting-machine-readable-zones-in-passport-images/). Since there will be many image processing operations applied to help us detect and extract the credit card digits, I’ve included numerous intermediate screenshots of the input image as it passes through our image processing pipeline. These additional screenshots will give you extra insight as to how we are able to chain together basic image processing techniques to build a solution to a computer vision project.

**1.3.1.1 Input reference and original images**

First, we load the reference  OCR-A image, followed by converting it to grayscale and thresholding + inverting it. In each of these operations we store or overwrite ref, our reference image. such that the digits appear as white on a black background and invert it, such that the digits appear as white on a black.

****

**Figure 1.3:** The OCR-A font for the digits 0-9. We will be using this font along with template matching to OCR credit card digits in images.

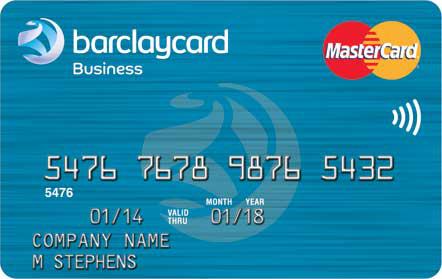
Then we find the contours present in the reference image. Then, due to how OpenCV 2.4, 3, and 4 versions [store the returned contour information differently](https://www.pyimagesearch.com/2015/08/10/checking-your-opencv-version-using-python/), we check the version and make an appropriate change to refCnts. Next, we [sort the contours](https://www.pyimagesearch.com/2015/04/20/sorting-contours-using-python-and-opencv/) from left-to-right as well as initialize a dictionary, digits , which maps the digit name to the region of interest. At this point, we should loop through the contours, extract, and associate ROIs with their corresponding digits. Then we loop through the reference image contours. In the loop, **i** holds the digit name/number and **c** holds the contour.

We compute a bounding box around each contour, c , storing the (x, y)-coordinates and width/height of the rectangle. Now we extract the roi  from ref  (the reference image) using the bounding rectangle parameters. This ROI contains the digit. We resize each ROI to a fixed size of 57×88 pixels. We need to ensure every digit is resized to a fixed size in order to apply template matching for digit recognition later. We associate each digit 0-9 (the dictionary keys) to each roi  image (the dictionary values).

**At this point, we are done extracting the digits from our reference image and associating them with their corresponding digit name.**

Our next goal is to isolate the 16-digit credit card number in the input --image . We need to find and isolate the numbers before we can initiate template matching to identify each of the digits. These image processing steps are quite interesting and insightful, especially if you have never developed an image processing pipeline before, so be sure to pay close attention. Let’s continue by initializing a couple structuring kernels. You can think of a kernel as a small matrix which we slide across the image to do (convolution) operations such as blurring, sharpening, edge detection, or other image processing operations. We construct two such kernels - one rectangular and one square. We will use the rectangular one for a [Top-hat morphological operator](https://en.wikipedia.org/wiki/Top-hat_transform) and the square one for a closing operation. We’ll see these in action shortly. Now let’s prepare the image we are going to OCR. Now we load our command line argument image  which holds the photo of the credit card. Then, we resize it to width=300 , maintaining the aspect ratio followed by converting it to grayscale

Let’s take a look at our input image:

****

**Figure 1.4:** The example input credit card image that we will be OCR’ing

Followed by our resize and grayscale operations:

****

**Figure 1.5:** Converting the image to grayscale is a requirement prior to applying the rest of our image processing pipeline.

Now that our image is grayscaled and the size is consistent, let’s perform a morphological operation: Using our rectKernel and our gray  image, we perform a [Top-hat](https://en.wikipedia.org/wiki/Top-hat_transform) morphological operation, storing the result as tophat. The Top-hat operation reveals light regions against a dark background (i.e. the credit card numbers) as you can see in the resulting image below:

****

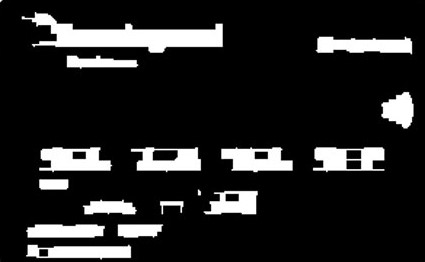
**Figure 1.6:** Applying a tophat operations reveals light regions (i.e., the credit card digits) against a dark background.

The next step in our effort to isolate the digits is to compute a [Scharr gradient](https://en.wikipedia.org/wiki/Sobel_operator) of the tophat  image in the *x*-direction. We complete this computation, storing the result as gradX. After computing the absolute value of each element in the gradX  array, we take some steps to scale the values into the range *[0-255]*. To do this we compute the minVal  and maxVal  of gradX followed by our scaling equation(i.e., min/max normalization). The last step is to convert gradX  to a uint8  which has a range of*[0-255]*. The result is shown in the image below:

****

**Figure 1.7:** Computing the Scharr gradient magnitude representation of the image reveals vertical changes in the gradient.

To close the gaps, we do a closing operation on. Notice that we use our rectKernel  again. Subsequently we perform an [Otsu](https://en.wikipedia.org/wiki/Otsu%27s_method) and binary threshold of the gradX  image followed by another closing operation. The result of these steps is shown here:



**Figure 1.8:** Thresholding our gradient magnitude representation reveals candidate regions” for the credit card numbers we are going to OCR.

We find the contours and store them in a list, cnts . Then, we initialize a list to hold the digit group locations. Now let’s loop through the contours while filtering based on the aspect ratio of each, allowing us to prune the digit group locations from other, irrelevant areas of the credit card:

We loop through the contours the same way we did for the reference image. After computing the bounding rectangle for each contour, we calculate the aspect ratio, ar , by dividing the width by the height. Using the aspect ratio, we analyze the shape of each contour. If ar  is between 2.5 and 4.0 (wider than it is tall), as well as the w  between 40 and 55 pixels and h  between 10 and 20 pixels, we append the bounding rectangle parameters in a convenient tuple to locs.

***Note:*** These the values for the aspect ratio and minimum width and height were found experimentally on my set of input credit card images. You may need to change these values for your own applications.

The following image shows the groupings that we have found - for demonstration purposes, I had OpenCV draw a bounding box around each group:

****

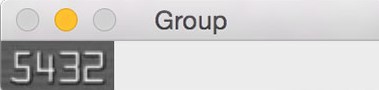
**Figure 1.9:** Highlighting the four groups of four digits (sixteen overall) on a credit card.

We sort the locs  according to the x-value so they will be ordered from left to right. We initialize a list, output , which will hold the image’s credit card number. Now that we know where each group of four digits is, let’s loop through the four sorted groupings and determine the digits therein. This loop is rather long and is broken down into three code blocks.

**1.3.1.1 Localizing the four groups**

In the first block for this loop, we extract and pad the group by 5 pixels on each side, apply thresholding, and find and sort contours. For the details, be sure to refer to the code.

Shown below is a single group that has been extracted:

****

**Figure 1.10:** An example of extracting a single group of digits from the input credit card for OCR.

Using cv2.boundingRect  we obtain parameters necessary to extract a ROI containing each digit. In order for template matching to work with some degree of accuracy, we resize the roi  to the same size as our reference OCR-A font digit images (57×88 pixels).

We initialize a scores  list. Think of this as our confidence score - the higher it is, the more likely it is the correct template. Now, let’s loop (third nested loop) through each reference digit **and perform template matching. This is where the heavy lifting is done for this script**. OpenCV, has a handy function called cv2.matchTemplate  in which you supply two images: one being the template and the other being the input image. The goal of applyingcv2.matchTemplate  to these two images is to determine how similar they are. In this case we supply the reference digitROI  image and the roi  from the credit card containing a candidate digit. Using these two images we call the template matching function and store the result. Next, we extract the score  from the result and append it to our scores  list. This completes the inner-most loop. Using the scores (one for each digit 0-9), we take the maximum score — the maximum score should be our correctly identified digit. We find the digit with the max score, grabbing the specific index via np.argmax . The integer name of this index represents the most-likely digit based on the comparisons to each template (again, keeping in mind that the indexes are already pre-sorted 0-9)

**1.3.1.2 Extracting each of these four groups and Segmenting each digit**

Finally, let’s draw a rectangle around each group and view the credit card number on the image in red text. For the third and final block for this loop, we draw a 5-pixel padded rectangle around the group followed by drawing the text on the screen. The last step is to append the digits to the output list. The Pythonic way to do this is to use theextend  function which appends each element of the iterable object (a list in this case) to the end of the list. To see how well the script performs, let’s output the results to the terminal and display our image on the screen.

**1.3.1.3 Recognizing each digit using template matching**

1. Stores credit card types in a dictionary.
2. Takes a reference image and extracts the digits.
3. Stores the digit templates in a dictionary.
4. Localizes the four credit card number groups, each holding four digits (for a total of 16 digits).
5. Extracts the digits to be “matched”.
6. Performs template matching on each digit, comparing each individual ROI to each of the digit templates 0-9, whilst storing a score for each attempted match.
7. Finds the highest score for each candidate digit, and builds a list called output  which contains the credit card number.
8. Outputs the credit card number and credit card type to our terminal and displays the output image to our screen.

It’s now time to see the script in action and check on our results.

**1.3.2 Owner details retrieval**

The retrieved owner number is sent to the database where it undergoes comparision. If number exists, the details of the owner are automatically filled on tkinter payment form. For this we can use sqlite3. SQLite3 can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249. You do not need to install this module separately because it is shipped by default along with Python version 2.5.x onwards.

To use sqlite3 module, you must first create a connection object that represents the database and then optionally you can create a cursor object, which will help you in executing all the SQL statements. In our project these are cursor classes played important role.

**connection.cursor([cursorClass])**

This routine creates a **cursor** which will be used throughout of your database programming with Python. This method accepts a single optional parameter cursorClass. If supplied, this must be a custom cursor class that extends sqlite3.Cursor.

**connection.execute(sql [, optional parameters])**

This routine is a shortcut of the above execute method provided by the cursor object and it creates an intermediate cursor object by calling the cursor method, then calls the cursor's execute method with the parameters given.

**1.4 Hardware and Software Requirements:**

* **Hardware**
  + Processor speed : 1.0gHz
  + Ram : 512 MB
  + Hard Disk : 40GB
  + Monitor : 15vga colour
* **Software** 
  + Platform : Python
  + OS: Windows **xp** and above..
  + Tools : Anaconda-(Spyder)

**1.5.3 TOOLS**

**Python Idle:**

Integrated development environment or integrated development and learning environment has been bundled with the default implementation of the language since 1.5.2b1. It is packaged as an optional part of the Python packaging with many [Linux distributions](https://en.wikipedia.org/wiki/Linux_distributions). It is completely written in Python and the [Tkinter](https://en.wikipedia.org/wiki/Tkinter) GUI toolkit ([wrapper](https://en.wikipedia.org/wiki/Wrapper_function) functions for [Tcl](https://en.wikipedia.org/wiki/Tcl)/[Tk](https://en.wikipedia.org/wiki/Tk_(framework))). IDLE is intended to be a simple [IDE](https://en.wikipedia.org/wiki/Integrated_development_environment) and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter.

**Spyder:**

It is an [open source](https://en.wikipedia.org/wiki/Open-source_software) cross-platform [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for scientific programming in the [Python language](https://en.wikipedia.org/wiki/Python_(programming_language)). Spyder integrates with a number of prominent packages in the scientific Python stack, including [NumPy](https://en.wikipedia.org/wiki/NumPy), [SciPy](https://en.wikipedia.org/wiki/SciPy), [Matplotlib](https://en.wikipedia.org/wiki/Matplotlib), [pandas](https://en.wikipedia.org/wiki/Pandas_(software)), [IPython](https://en.wikipedia.org/wiki/IPython), [SymPy](https://en.wikipedia.org/wiki/SymPy) and [Cython](https://en.wikipedia.org/wiki/Cython), as well as other open source software.It is released under the [MIT license](https://en.wikipedia.org/wiki/MIT_license).

Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.

Spyder is extensible with first- and third-party plugins, includes support for interactive tools for data inspection and embeds Python-specific code quality assurance and introspection instruments, such as Pyflakes, Pylint and Rope. It is available cross-platform through [Anaconda](https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)), on Windows, on macOS through [MacPorts](https://en.wikipedia.org/wiki/MacPorts), and on major Linux distributions such as [Arch Linux](https://en.wikipedia.org/wiki/Arch_Linux), [Debian](https://en.wikipedia.org/wiki/Debian), [Fedora](https://en.wikipedia.org/wiki/Fedora_(operating_system)), [Gentoo Linux](https://en.wikipedia.org/wiki/Gentoo_Linux), [openSUSE](https://en.wikipedia.org/wiki/OpenSUSE) and [Ubuntu](https://en.wikipedia.org/wiki/Ubuntu_(operating_system)).

Spyder uses Qt for its GUI, and is designed to use either of the [PyQt](https://en.wikipedia.org/wiki/PyQt) or [PySide](https://en.wikipedia.org/wiki/PySide) Python bindings. QtPy, a thin abstraction layer developed by the Spyder project and later adopted by multiple other packages, provides the flexibility to use either backend.

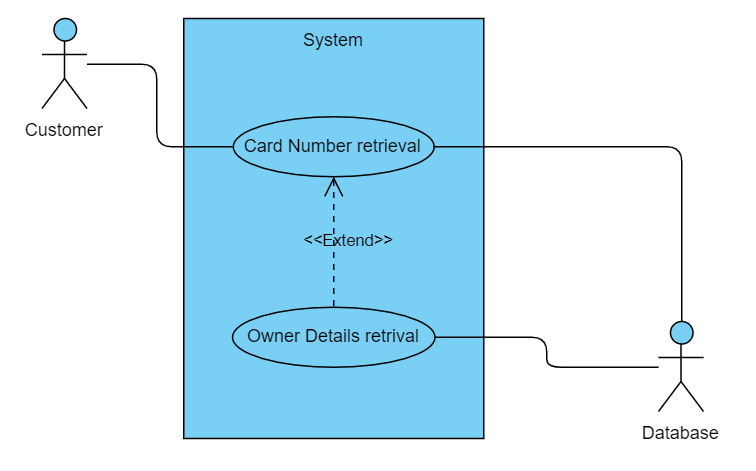
**2. SOFTWARE ENVIRONMENT**

**2.1 Data Description**

In this section we are going to discuss about how the image undergoes different phases to retrieve the number printed on it which will display on the screen.

**2.2 UML DIAGRAMS**

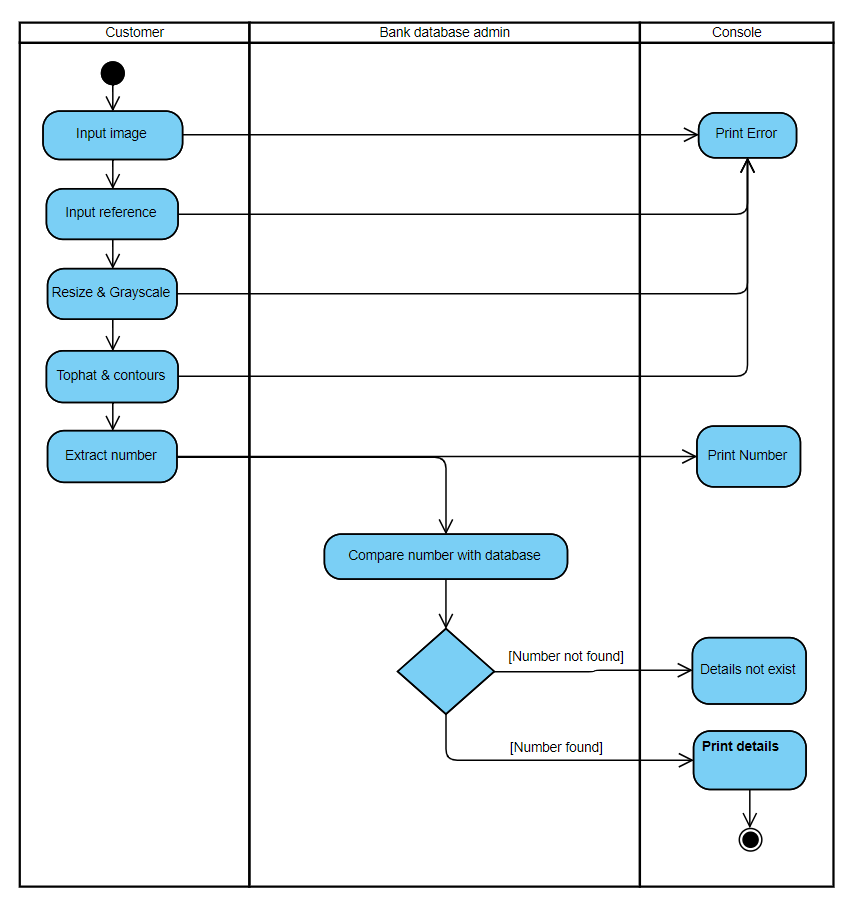
**Design phase:** It seeks to develop detailed specifications that emphasize the physical solution to the user's information technology needs.



**Figure 2.1:** Design phase of our project.

**Activity diagram:**

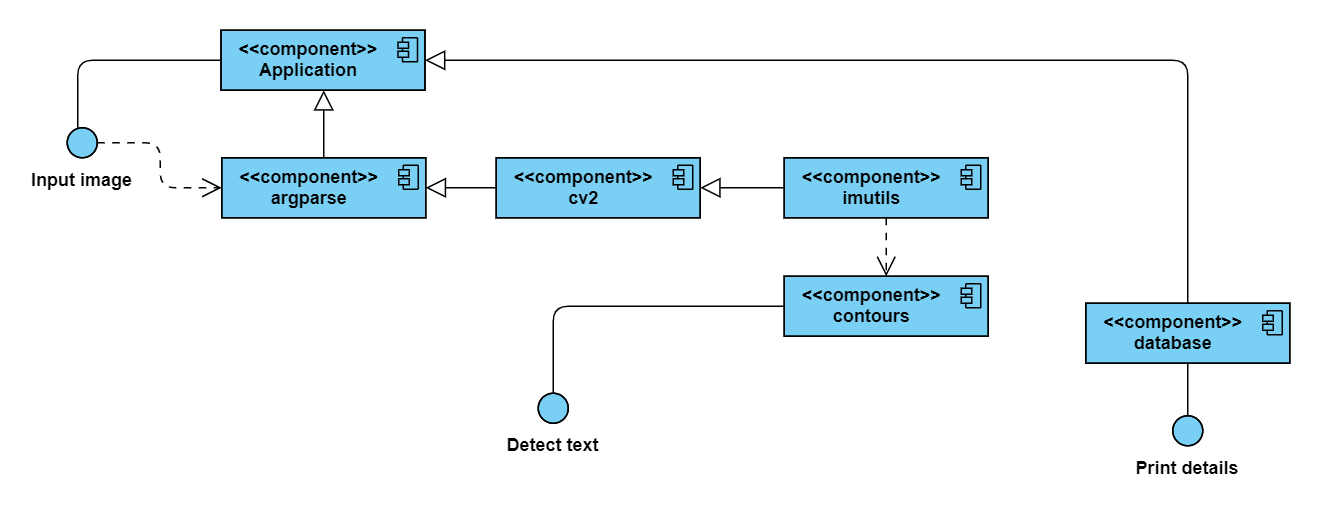
It is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.



**Figure 2.2:** Activity diagram of an application to recognize the number of payment card and extracting its owner details.

**Component diagram:**

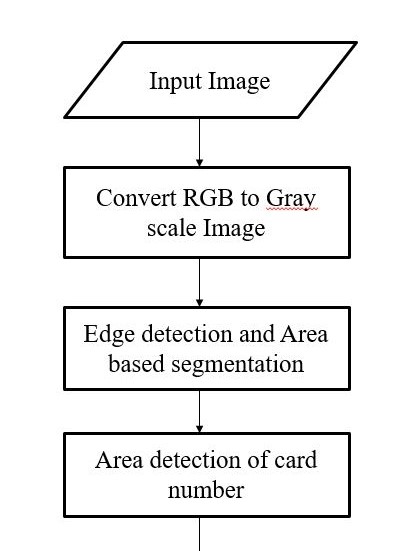
It is a special kind of diagram in UML that does not describe the functionality of the system but it describes the components used to make those functionalities.

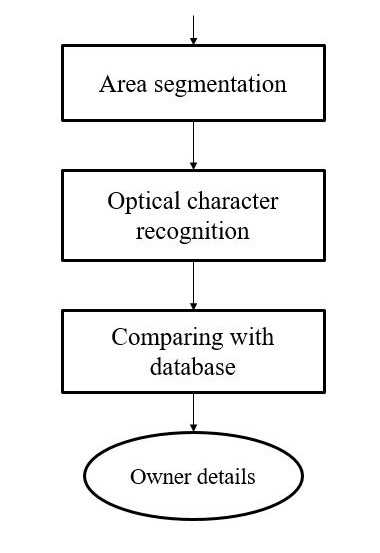


**Figure 2.3:** Component diagram of an application to recognize the number of payment card and extracting its owner details.

**System design:**



****

****

**3. IMPLEMENTATION**

This section gives you overview about the classes and modules that have been used for our project.

**Source code:**

**Table\_db\_create.py:**

import sqlite3 as sqlitedb

print("\nConnecting to database..")

con = sqlitedb.connect('PaymentCards\_DB.sqlite')

print("\nConnected.")

print("\nCreating cursor..")

cur = con.cursor()

print("\nCreated.")

print("\nCreating table..")

cur.execute('DROP TABLE IF EXISTS CARDOWNERDETAILS')

cur.execute('CREATE TABLE CARDOWNERDETAILS (CARDNUMBER TEXT, FNAME TEXT, LNAME TEXT, EMAIL TEXT, PHONE TEXT, COUNTRY TEXT, STATE TEXT, TOWN TEXT, PINCODE TEXT, DOORNUM TEXT, NEARBYLOC TEXT)')

print("\nCreated..")

**insert\_into\_table.py:**

import sqlite3 as sqlitedb

print("\nConnecting to database..")

con = sqlitedb.connect('PaymentCards\_DB.sqlite')

print("\nConnected.")

print("\nCreating cursor..")

cur = con.cursor()

print("\nCreated.")

print("\nInserting columns..")

loop = int(input("Enter how many records you will enter : "))

for i in range(0,loop):

cardnumber = input("Enter card number : ")

fname = input("Enter fname : ")

lname = input("Enter lname : ")

email = input("Enter email : ")

phone = input("Enter phone number : ")

country = input("Enter country : ")

state = input("Enter state : ")

town = input("Enter town : ")

pincode = input("Enter pincode : ")

doornum = input("Enter door number : ")

nearbyloc = input("Enter near by location : ")

try:

if cur.execute('INSERT INTO CARDOWNERDETAILS (CARDNUMBER,FNAME,LNAME,EMAIL,PHONE,COUNTRY,STATE,TOWN,PINCODE,DOORNUM,NEARBYLOC) VALUES(?,?,?,?,?,?,?,?,?,?,?)',(cardnumber,fname,lname,email,phone,country,state,town,pincode,doornum,nearbyloc)):

print("\nInsertion success..")

else:

print("\nInsertion failed!")

con.commit()

except:

print("\nInvalid roll number!")

**printing\_records.py:**

import sqlite3 as sqlitedb

print("\nConnecting to database..")

con = sqlitedb.connect('PaymentCards\_DB.sqlite')

print("\nConnected.")

print("\nCreating cursor..")

cur = con.cursor()

print("\nCreated.")

print("\nDetails printing..\n")

cur.execute("SELECT \* FROM CARDOWNERDETAILS")

col\_name\_list = [tuple[0] for tuple in cur.description]

print(col\_name\_list)

result = cur.execute('SELECT \* FROM CARDOWNERDETAILS')

for i in result:

print(i)

**payment\_ocr.py:**

from imutils import contours

import numpy as np

import imutils

import cv2

import sqlite3 as sqlitedb

from tkinter import \*

from tkinter import filedialog

from PIL import ImageTk, Image

top = Tk()

top.geometry("715x505")

top.title('Address')

top.configure(background='#D6EAF8')

def order\_placed():

order\_placed\_label = Label(top, bg="#58D68D", text = "Order placed").place(x = 170, y = 470)

main\_label = Label(top, text="Payment", font="times", width=65,bg="#3498DB", fg="#FDFEFE", justify="center").place(x=0, y=10)

cardnumber\_label = Label(top, text = "Card number").place(x = 60,y = 60)

fname\_label = Label(top, text = "First name").place(x = 60,y = 100)

lname\_label = Label(top, text = "Last name").place(x = 60,y = 140)

email\_label = Label(top, text = "Email id").place(x = 60,y = 180)

phone\_label = Label(top, text = "Phone number").place(x = 60,y = 220)

country\_label = Label(top, text = "Country").place(x = 60,y = 260)

address\_label = Label(top, text = "Address").place(x = 60,y = 300)

cardnumber\_entry = Entry(top)

cardnumber\_entry.place(x = 150, y = 60)

fname\_entry = Entry(top)

fname\_entry.place(x = 150, y = 100)

lname\_entry = Entry(top)

lname\_entry.place(x = 150, y = 140)

email\_entry = Entry(top)

email\_entry.place(x = 150, y = 180)

phone\_entry = Entry(top)

phone\_entry.place(x = 150, y = 220)

country\_entry = Entry(top)

country\_entry.place(x = 150, y = 260)

address\_entry = Text(top, height=5, width=20)

address\_entry.place(x = 150, y = 300)

sbmitbtn = Button(top,command = order\_placed ,text = "Place order",padx=5,pady=0.5,font="times", activebackground = "#58D68D", activeforeground = "#FDFEFE", bg = "#3498DB", fg = "#FDFEFE", relief="groove").place(x = 150, y = 410)

img\_path = filedialog.askopenfilename(filetypes=[("PNG", "\*.png"), ("JPEG", "\*.jpg"), ("All files", "\*")])

FIRST\_NUMBER = {

"3": "American Express",

"4": "Visa",

"5": "MasterCard",

"6": "Discover Card"

}

ref\_img = "E:/Study/Project/Automatic details extraction from Payment cards with OCR and OpenCV/ocr\_a\_reference.png"

ref = cv2.imread(ref\_img)

ref = cv2.cvtColor(ref, cv2.COLOR\_BGR2GRAY)

ref = cv2.threshold(ref, 10, 255, cv2.THRESH\_BINARY\_INV)[1]

refCnts = cv2.findContours(ref.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]

digits = {}

for (i, c) in enumerate(refCnts):

(x, y, w, h) = cv2.boundingRect(c)

roi = ref[y:y + h, x:x + w]

roi = cv2.resize(roi, (57, 88))

digits[i] = roi

rectKernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (9, 3))

sqKernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

image = cv2.imread(img\_path)

image = imutils.resize(image, width=300)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

tophat = cv2.morphologyEx(gray, cv2.MORPH\_TOPHAT, rectKernel)

tophat = cv2.morphologyEx(gray, cv2.MORPH\_TOPHAT, rectKernel)

gradX = cv2.Sobel(tophat, ddepth=cv2.CV\_32F, dx=1, dy=0,

ksize=-1)

gradX = np.absolute(gradX)

(minVal, maxVal) = (np.min(gradX), np.max(gradX))

gradX = (255 \* ((gradX - minVal) / (maxVal - minVal)))

gradX = gradX.astype("uint8")

gradX = cv2.morphologyEx(gradX, cv2.MORPH\_CLOSE, rectKernel)

thresh = cv2.threshold(gradX, 0, 255,

cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)[1]

thresh = cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, sqKernel)

cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if imutils.is\_cv2() else cnts[1]

locs = []

for (i, c) in enumerate(cnts):

(x, y, w, h) = cv2.boundingRect(c)

ar = w / float(h)

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))

locs = sorted(locs, key=lambda x:x[0])

output = []

for (i, (gX, gY, gW, gH)) in enumerate(locs):

groupOutput = []

group = gray[gY - 5:gY + gH + 5, gX - 5:gX + gW + 5]

group = cv2.threshold(group, 0, 255,

cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)[1]

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 c in digitCnts:

(x, y, w, h) = cv2.boundingRect(c)

roi = group[y:y + h, x:x + w]

roi = cv2.resize(roi, (57, 88))

scores = []

for (digit, digitROI) in digits.items():

result = cv2.matchTemplate(roi, digitROI, cv2.TM\_CCOEFF)

(\_, score, \_, \_) = cv2.minMaxLoc(result)

scores.append(score)

groupOutput.append(str(np.argmax(scores)))

cv2.rectangle(image, (gX - 5, gY - 5),

(gX + gW + 5, gY + gH + 5), (0, 0, 255), 2)

cv2.putText(image, "".join(groupOutput), (gX, gY - 15),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.65, (0, 0, 255), 2)

output.extend(groupOutput)

print("Credit Card Type: {}".format(FIRST\_NUMBER[output[0]]))

final\_image = image

#cv2.imshow("Image", final\_image)

cv2.waitKey(1)

array\_to\_image = Image.fromarray(final\_image)

array\_to\_image.save("array to image.png")

cardnumber = int("".join(output))

print(int("".join(output)))

main\_label = Label(top, text="Payment", font="times", bg="#3498DB", fg="#FDFEFE", width=37, justify="center").place(x=0, y=10)

cardnumber\_label = Label(top, text = "Card number").place(x = 60,y = 60)

fname\_label = Label(top, text = "First name").place(x = 60,y = 100)

lname\_label = Label(top, text = "Last name").place(x = 60,y = 140)

email\_label = Label(top, text = "Email id").place(x = 60,y = 180)

phone\_label = Label(top, text = "Phone number").place(x = 60,y = 220)

country\_label = Label(top, text = "Country").place(x = 60,y = 260)

address\_label = Label(top, text = "Address").place(x = 60,y = 300)

cardnumber\_entry = Entry(top)

cardnumber\_entry.place(x = 150, y = 60)

fname\_entry = Entry(top)

fname\_entry.place(x = 150, y = 100)

lname\_entry = Entry(top)

lname\_entry.place(x = 150, y = 140)

email\_entry = Entry(top)

email\_entry.place(x = 150, y = 180)

phone\_entry = Entry(top)

phone\_entry.place(x = 150, y = 220)

country\_entry = Entry(top)

country\_entry.place(x = 150, y = 260)

address\_entry = Text(top, height=5, width=20)

address\_entry.place(x = 150, y = 300)

con = sqlitedb.connect('PaymentCards\_DB.sqlite')

cur = con.cursor()

result\_cursor = cur.execute('SELECT \* FROM CARDOWNERDETAILS WHERE CARDNUMBER = ?',(cardnumber,))

for result\_tuple in result\_cursor:

result\_list = list(result\_tuple)

cardnumber\_entry.insert(0,result\_list[0])

fname\_entry.insert(0,result\_list[1])

lname\_entry.insert(0,result\_list[2])

email\_entry.insert(0,result\_list[3])

phone\_entry.insert(0,result\_list[4])

country\_entry.insert(0,result\_list[5])

address\_entry.insert(INSERT,result\_list[6] +"\n"+ result\_list[7] +"\n"+ result\_list[8] +"\n"+ result\_list[9]+"\n"+ result\_list[10]+"\n")

final\_image\_getting= ImageTk.PhotoImage(Image.open("array to image.png"))

final\_image\_label = Label(top, image=final\_image\_getting).place(x = 400, y = 60)

top.mainloop()

**4. TESTING**

**4.1 System Testing**

Testing is done with the test data, specially designed to show that the system will operate successfully in all conditions. The system testing is a confirmation that everything is correct and an opportunity to show the user that the system works. The final step involves Validation testing, which determines whether the software, functions as the user expected. The end-user rather than the system developer conducts this test.

Most software developers has a process called “Alpha and Beta test” to uncover those, that only the end user seems able to find. This is the final step in system life cycle. Here we implement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion.

**4.2 Unit Testing**

It verifies the smallest unit of software design module. This is known as “Module Testing”. The modules are tested separately. This testing is carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

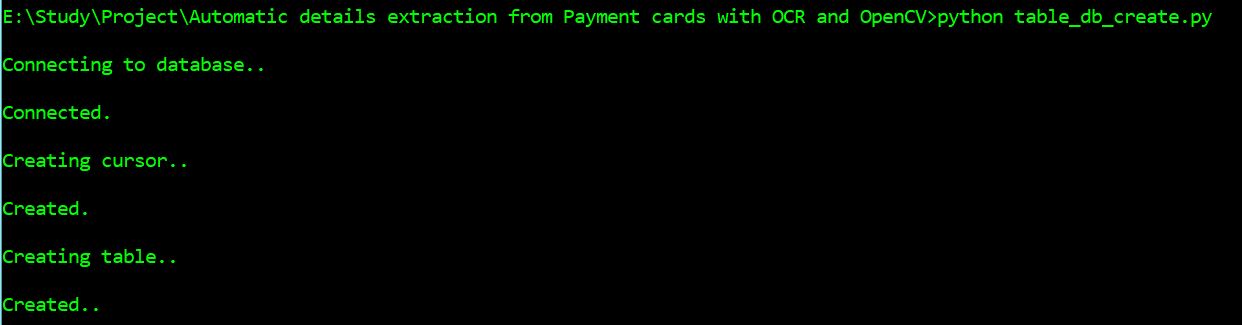
**4.3 Integration Testing**

It is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire program is tested as a whole. In the integration-testing step, all the errors uncovered is corrected for the next testing steps.

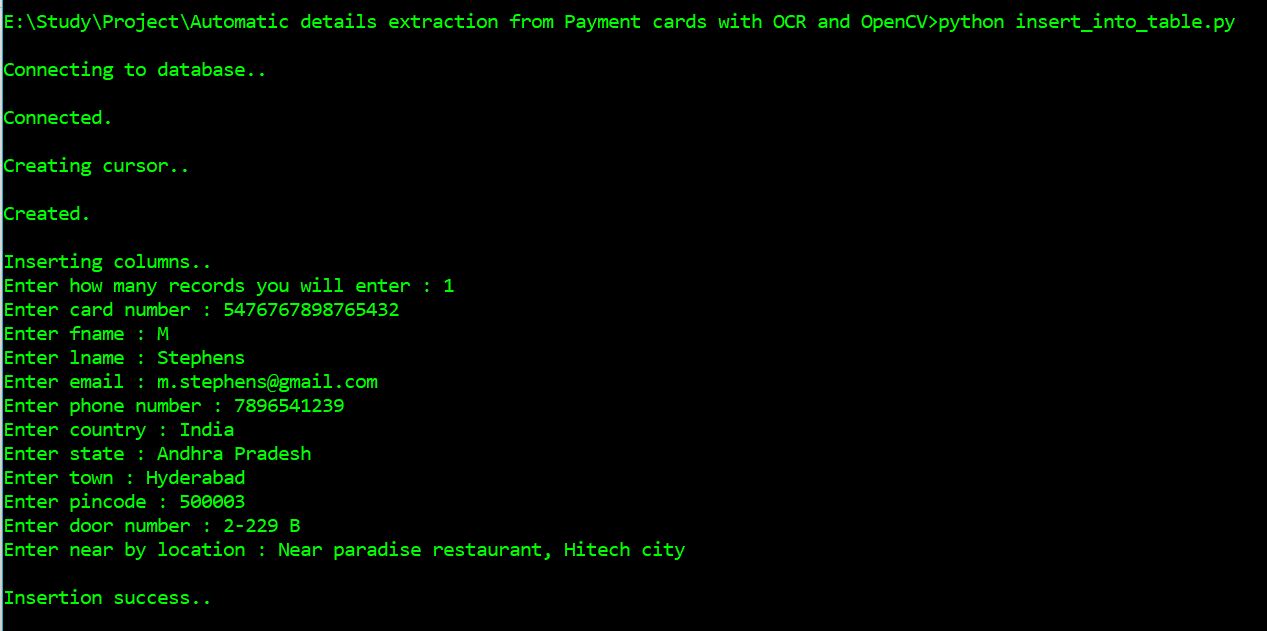
**4.4 Validation Testing**

This testing is done to uncover functional errors, that is to check whether functional characteristics confirm to specification or not.

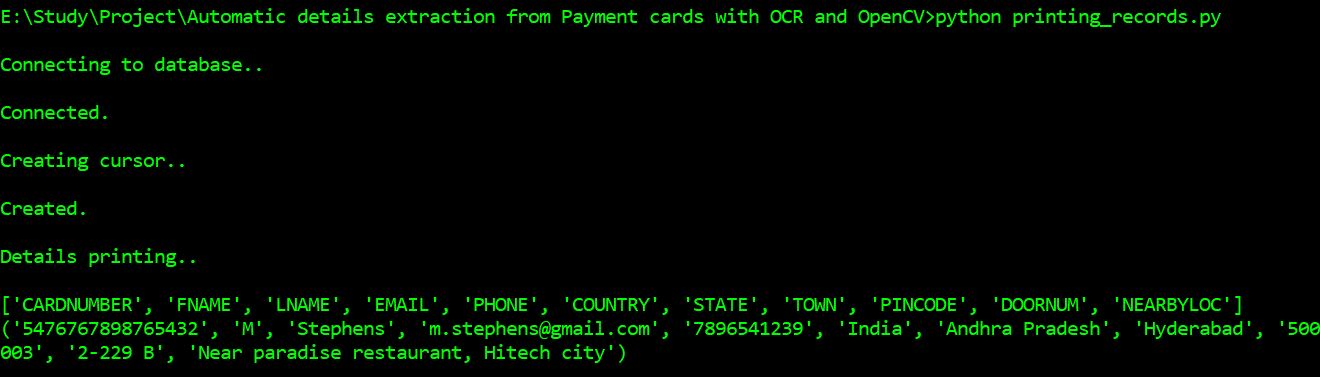
**5. SCREENSHOTS**



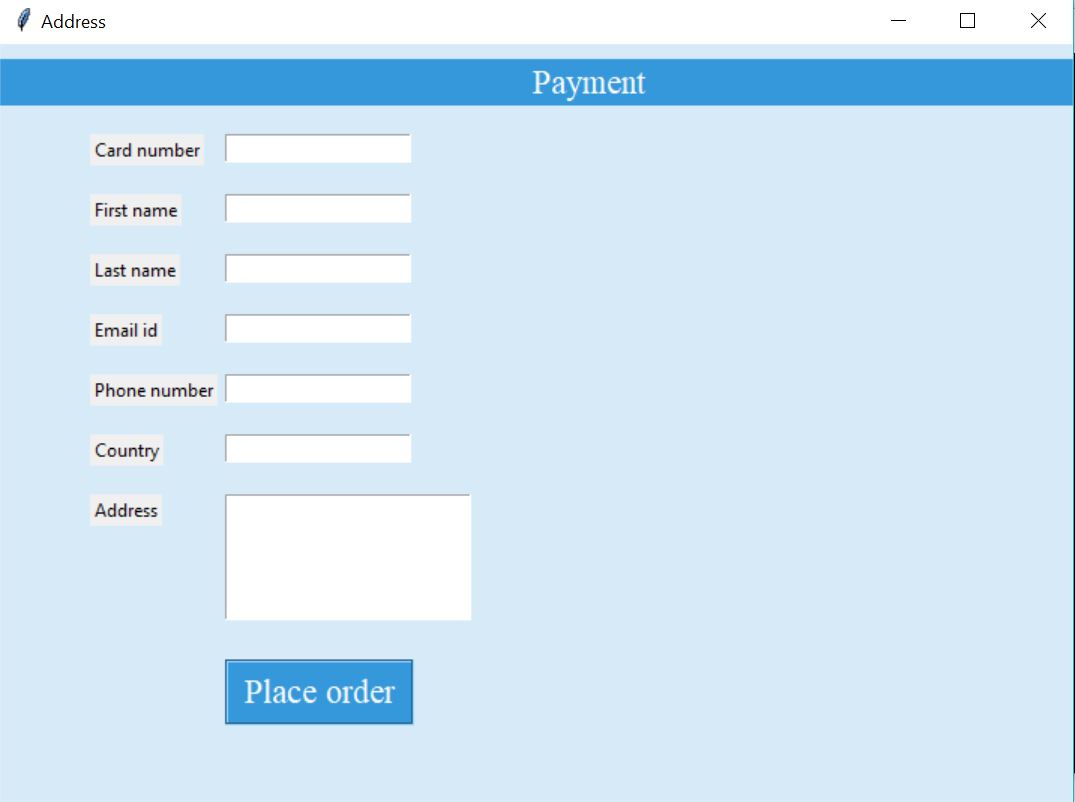
**Figure 5.1:** Screenshot that creates table of owner details.



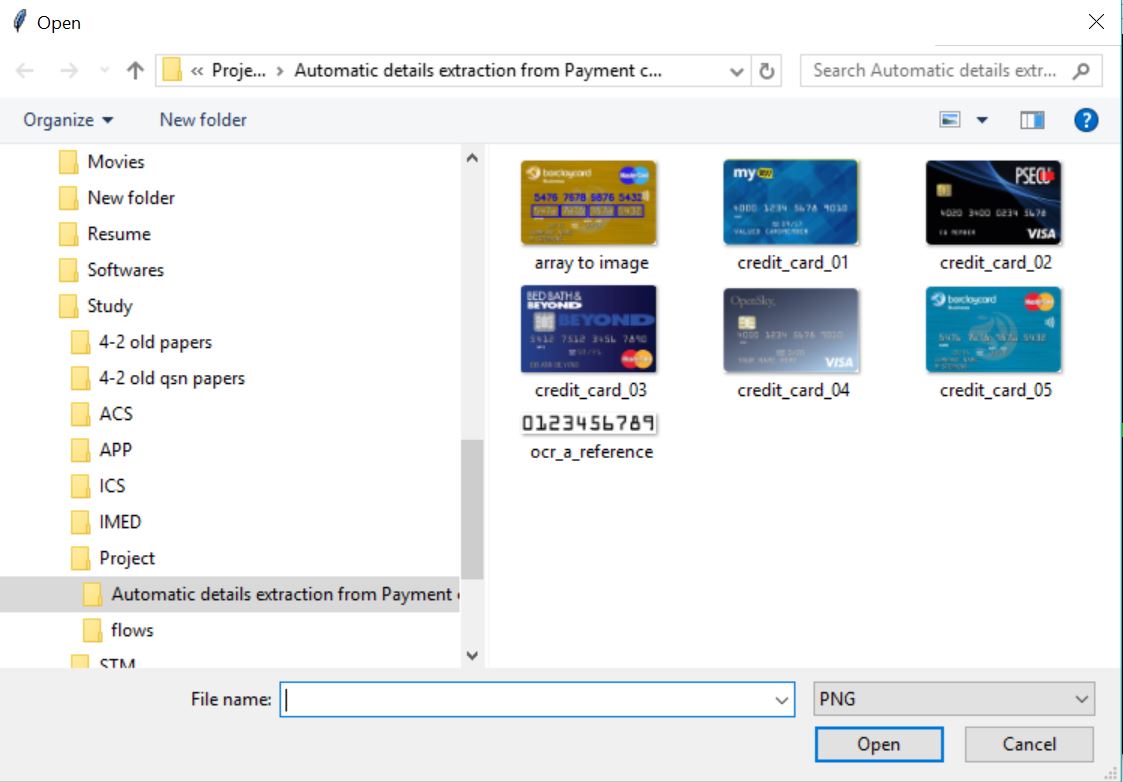
**Figure 5.2:** Screenshot that inserts owner details into table.



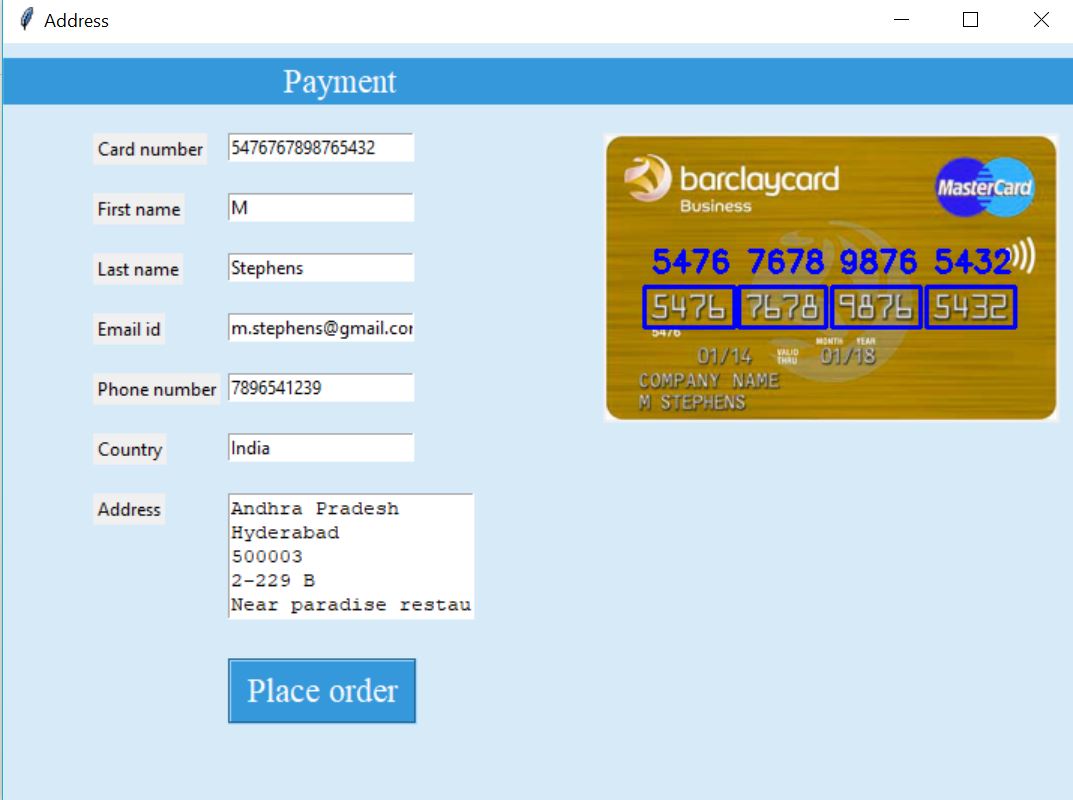
**Figure 5.3:** Screenshot that check all data from table.



**Figure 5.4:** tkinter window with empty form.



**Figure 5.5:** Browse window to select payment card.



**Figure 5.6:** Number is retrieved from payment card and details are selected from database.

**6. CONCLUSION**

As this application Increases the efficiency and effectiveness, Instantly search through details quickly, Accurately and Ensuring with no waste of time in typing whole details in payment activities in relaxed alignment in mobiles. There is an immediate need of such kind of payment cards details retrieval system in India as there are some issues of time killing payment forms, lost cards etc. Government should take some interest in developing this system as this system is very economical and eco-friendly, if applied effectively . This change will help in the progress of the payment activities in websites, android applications etc.

**APPENDIX : REFERENCES**

1. https://en.wikipedia.org/wiki/OpenCV

2. https://en.wikipedia.org/wiki/Optical\_character\_recognition

3. https://en.wikipedia.org/wiki/SQLite

4. http://www.sqlitetutorial.net/sqlite-python/

5. https://www.tutorialspoint.com/sqlite/sqlite\_python.htm

6. https://www.learnopencv.com/deep-learning-based-text-recognition-ocr-using-tesseract-and-opencv/

7. https://opencv-python tutroals.readthedocs.io/en/latest/py\_tutorials/py\_ml/py\_knn/py\_knn\_opencv/py\_knn\_opencv.html

8. https://www.educationumbrella.com/ocr-textbooks

9. www-cs.ccny.cuny.edu/~wolberg/capstone/opencv/LearningOpenCV.pdf

10. https://stackoverflow.com/questions/7542194/how-to-use-opencv-for-document-recognition-with-ocr

11. https://github.com/goncalopp/simple-ocr-opencv

12. https://docs.opencv.org/3.4/d7/ddc/classcv\_1\_1text\_1\_1OCRTesseract.html

13. https://coderwall.com/p/en\_zza/simple-digit-recognition-ocr-in-opencv-python

14. https://ampersandacademy.com/tutorials/python-data-science/recognise-text-and-digit-from-the-image-with-python-opencv-and-tesseract-ocr