CMPS446 – Fall 2024

(Image Processing & Computer Vision)

Project Submission (Phase 2)

FawryVision - Team 23

Presented To:

Dr. Youssef Ghattas

Prepared By:

Ahmed Tarek AbdelAal	1200088
Basel Hossam Abdelmoula	1200462
Heba Gamal	1200076

Table of Contents

Project Overview

Fawry Receipt OCR for extracting important information such as Balance recharge code

Project Idea

FawryVision is an image processing system designed to help users quickly and accurately extract a Fawry 16-digit code from receipts, Balance Recharge Code typically used to recharge phone credit. This system aims to reduce the challenges associated with manually reading and entering these codes, which can be difficult for individuals with vision impairments or prone to human error. By leveraging image processing techniques, FawryVision automatically identifies, isolates, and displays the code for easy input, minimizing the potential for misreading digits due to poor print quality, low lighting, or user oversight.

Additional Comments

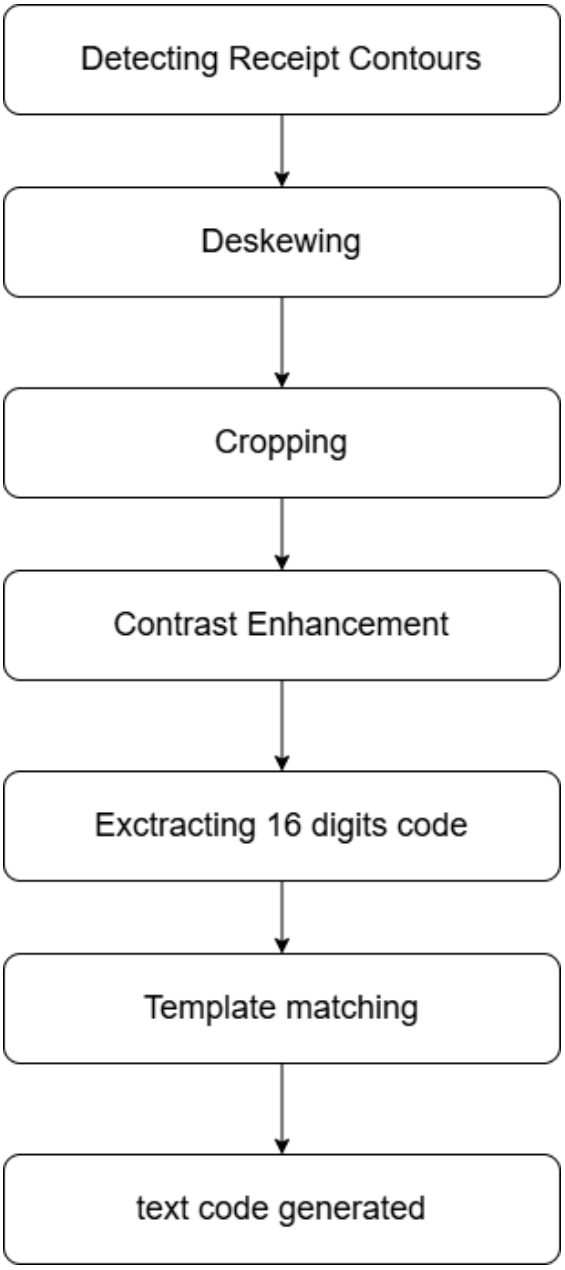
- This project emphasizes classical image processing techniques to maximize efficiency and avoid reliance on deep learning.
- The project is mainly focusing on the case where you bought a single charging code, got a Fawry receipt for it, and we will extract this code.
- Extracted Information is the 16 digits code and the total amount to be charged,

Proposed Block Diagram before implementation

Input:

- Camera Input: The phone or tablet camera captures the Fawry Receipt.

Processing:



Output:

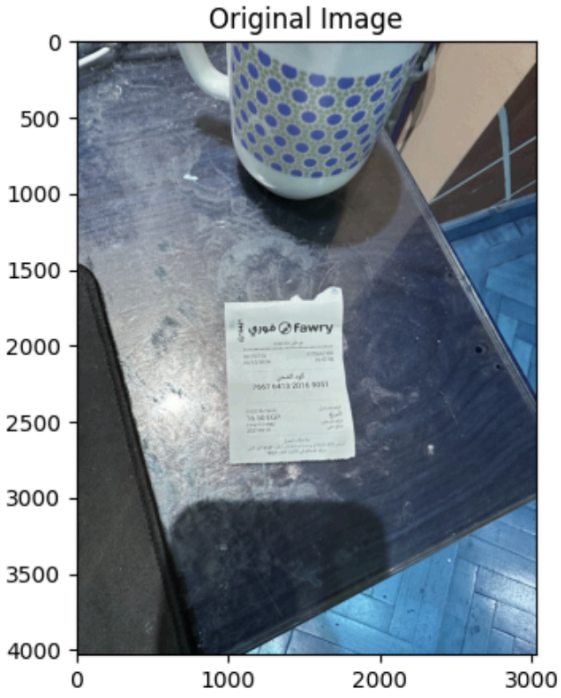
- Balance Recharge Code: Displayed a 16-digit number + Price

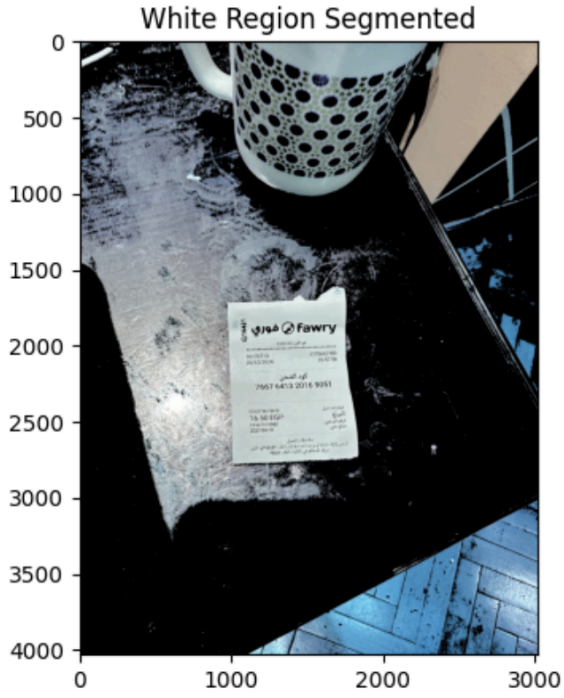
Approach 1

Colour based segmentation

This technique tries to use the fact that the receipt is white to perform the very initial segmentation, this section will contain the exact details:-

1- As a starting point, K means is performed on the whole RGB image, number of clusters used is 2, this ensures that the brighter colours will be grouped together (including the receipt), and the darker parts will be grouped into another cluster, The result is something like this one





2- One of the main problems of this approach is the white noise present on the surface, unfortunately since it shares the same colour as the

receipt, it gets extracted as well from Kmeans, now in order to reduce it , we need to perform some morphological operations, so as a result, image is first converted to greyscale

And here appears one of the biggest challenges, what threshold should be used

Choosing a low threshold, ex: minimum_threshold function, will be helpful for the right image, as we want as much bright as possible to be obtained, to not be affected by the shadow, however, for a photo like the one on the left, you need i high threshold, to be able to get rid of most of the white noise just from binary conversion.

I kept on trying out the thresholds like otsu and minimum_threshold, they were case specific, so I kept on trying out many thresholds, for instance 0.4,0.5,0.6, 0.6 proved to be the one the generalizes the best.

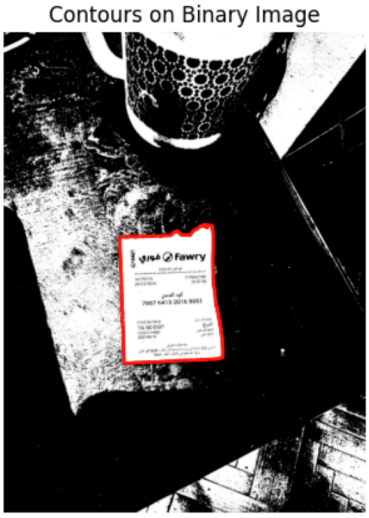
3- The next step was also really hard, and I was in a loop as something works with a threshold on an image, but does not work on the other one and so on, the step was morphological operations.

I kept on trying out all different sequences ,sizes and iterations of binary opening, closing , erosion, and after so many trial and error, the one that generalized best, surprisingly, was just applying binary opening once, with a 2x2 window and 15 iterations

It had the balance of not applying over erosion thus destroying the image, and also barely keeping out white noise from the receipt.

4- Now that we an image where the white region of the receipt is separated from the rest of the image (as much as possible), it is time to find the biggest contour surrounding this receipt, this is simply done using the function find_contours

But which contour should we select, i.e, should we select the longest contour (max len) or contour with max area, based on experimentation, max area was always the best option, even in later sections



5- Now that we obtained the contour, hopefully a rectangular one (as much as possible), we need to start approximating it to a rectangle, the purpose from this is, the 4 corners of the rectangle can be for performing perspective transformation, obtaining a bird's eye view on the receipt

This step was done using cv2.approxPolyDp, by iteratively decreasing the epsilon passed to the function, until the approximated polygon has exactly 4 points, meaning it is a rectangle

The polygon approximated is the one in green in the image on the left

It is worth noting that, sometimes this polygon cannot be obtained, if the contour is very noisy and not taking any rectangular shape, this occurs in cases of too much white noise

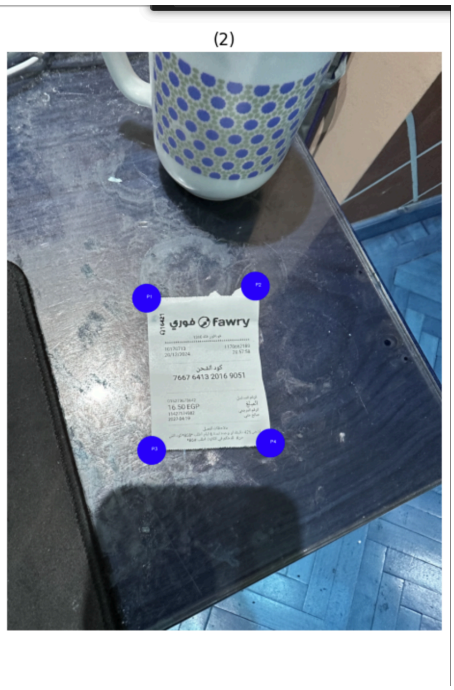
6-Now the 4 corners of the polygon are obtained, but there is one final trick, sometimes the polygon obtained has 3 corners aligning with the receipt, but a final corner way off, outlier detection and correction

7- Outlier detection is formed by: calculating the mean (x,y) of all the 4 points, then the outlier is the one furthest from it, now that we detected it, to correct it, I calculate the unit vector form the outlier to the center, then moving the outlier in this direction by a factor of (distance of outlier - mean distance)

I personally think that using something like prediction the point by:

If it is top left, the top left = bottom left + (top right - bottom right), as they should have same inclination in most cases, but the applied method actually works really well and I did not need to change it

8- Now finally it is time for the perspective transformation



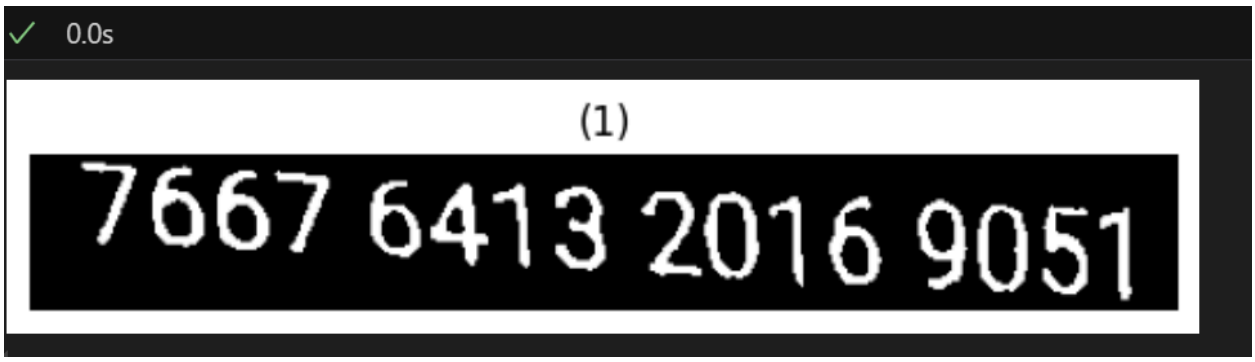
This step works so well when all the previous ones work in tandem , with all of their thresholds and heuristics

9- Now for the important part, code and price extraction, we will be using some dilation and then finding contours with greatest area in order to find our code and price, but to do this inverted binary conversion needs to be applied

Inverted Binary Image

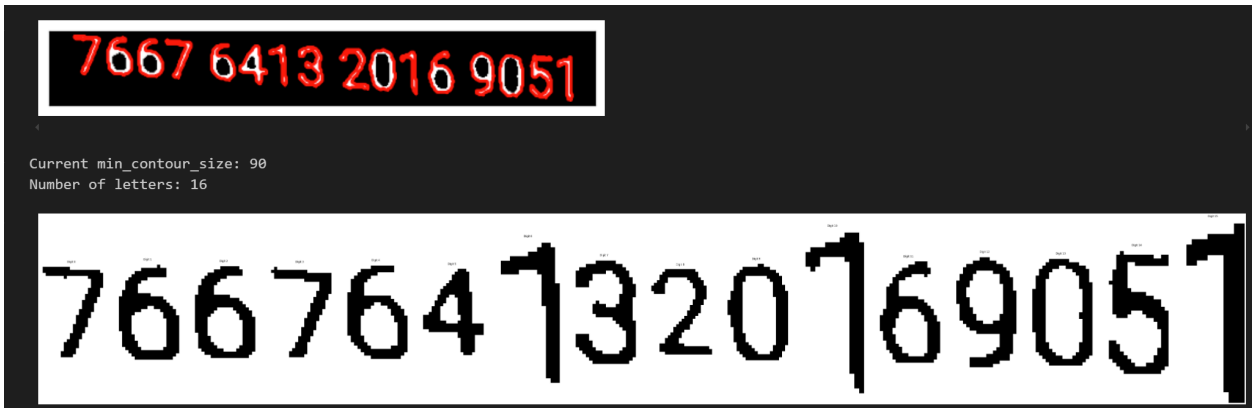
10- Now the next part is tricky, how to obtain proper contours around our targets, one approach is dilation, then checking for aspect ratios, this unfortunately did not work very well, it usually fails either due to the second form of the receipt (with QR code) having different ratios, or the perspective transformed image is not that good.

The other technique is, we know that the code is somewhere around the middle of the image on the right, and the price is around the last ⅓ of the image on the left, and code will have the largest contour area in its corresponding region, same for price. Now choosing dimensions to crop on is tricky, since as I mentioned we have 2 types of receipts, with different lengths, however, I managed to find proper dimensions by trial and error that generalize very well

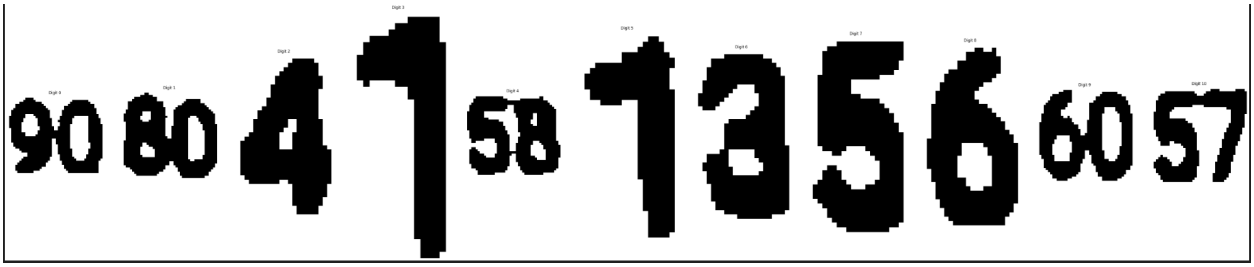


11- Now that we have our code, it is time for prediction, my approach was using template matching, template matching requires having some predefined templates, of a certain size, then comparing images of the same size to it, before getting to the headache of where to get templates from, we already now we need templates for each number, so let us start cropping digits!

12- Cropping digits was done by first finding contours in the image, the largest 16 contours to be specific, This was done by setting a large size for contours, finding contours of the size, then iteratively decreasing size until 16 contours are detected, not sharing or overlapping on the x-axis, the reason a large size is initially set, is that for instance numbers like 9 and 0 have 2 contours each, outer one, and one for inner circle,, so size needed to be limited, Finally, the image is cropped by getting the bounding box of each contour.



13- Things are not usually that perfect though, sometimes numbers are too close horizontally, the same contour is having 2 digits, for

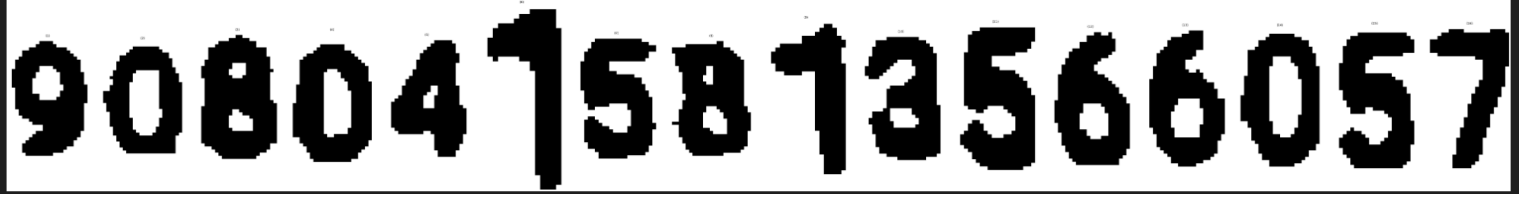


instance here

As you can see, 90, 80, 58, 60, 57 each couple was detected as one contour, now to counter this, one proposed solution would be erosion before contour detection, unfortunately this sometimes destroys numbers, even with low amounts.A better approach is, we know that the couple-digits detected have width, around the double of single digits, we also know that by maximum there would be 8 of them (2 * 8 = 16), now how about we iterate on the all the cropped numbers, check on width, and if it satisfies the double condition cut it in half, this was actually one of the most satisfying moments in the project for me, as you will see in the next pictures

Number of images after splitting: 15

Number of images after splitting: 16



This little trick neatly solved the problem, from trial and error, this barely fails!

13- Now for the elephant in the room template matching, how can I obtain a dataset with the correct dimensions? Is template matching even the solution? Why not use something like KNN or SVM on SIFT or HOG features from the image? The latter methods required a large dataset, so I opted for template matching, but still, how can I obtain proper templates??

The answer was simply, saving the cropped images, using them as templates, now I know, this is a data leak right? I cannot predict digits in the images using those exact digits! Yes for the first case, but actually the obtained templates easily generalized for all test cases it never saw before, even if the number is damaged, if the damage is not extremely severe, it can be matched, for instance number 7 has this skewed line to it, if the input is only the skewed line template matching knows it is 7, it comes from the fact that numbers on receipt do not change, they are PRINTED, which is the optimum case for template matching + the templates and cropping are neatly obtained, leaving very little margin for error.

14- A very similar procedure is carried out for price detection, but this time I added templates for the letters EGP as they exist in the cropped, also getting dimensions for cropping was quite tricky to generalize, but it worked.

As the dot it always before the last 2 digits i did not need to detect


```
    print("Predicted Price:", "".join(predictedPrice))
    print("Predicted Digits:", "".join(predicted_digits))
3]  ✓ 0.0s

Predicted Price: 25.00EGP
Predicted Digits: 90804158135660 57
```

Difficulties with colour segmentation pipeline

- Hard to determine thresholds that generalize well in different lighting conditions
- It is the most prone to white noise, as receipt and noise have same colour, making receipt detection harder
- Determining the correct combination of morphological operations to remove white noise
- Obtaining the correct rectangle around the receipt
- Choosing the right dimensions to crop on, before detecting the code or price
- Cropping the digits without getting a contour on 2 digits at the same time

Approach 2: Edge based Segmentation

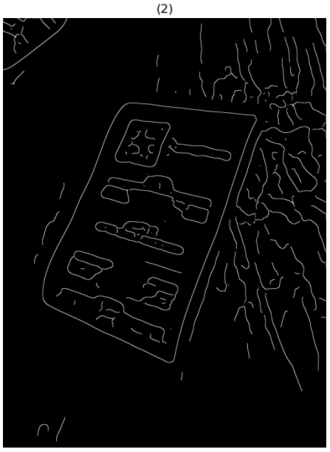
1) Initial Receipt Extraction

(1)

a) Convert the image to grayscale and apply contrast enhancement using Gamma corr .

b) Apply minimum_thresholding to get the optimal threshold for obtaining a binary image

(2)



c) Apply Canny edge detector with high sigma value to reduce the edges detected from the background noise

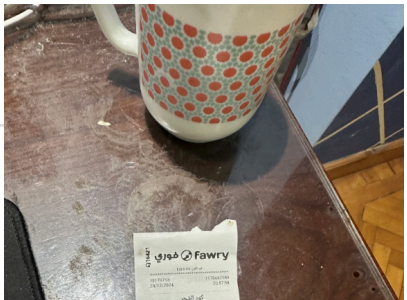
d) Dilate the edges to connect the receipt edges if it's disconnected , so that find_countours can detect the receipt contour .

e) Apply Thinning to the dilated image before the corner detection step because thick edges resulting from dilation produce false positive corners . The dilation Step's only usage was connecting the receipts edge if disconnected .
After it's connected we can return it back thin .

f) Find The largest Contour area

g) Use Approximate Polygon to get exactly 4 corners . If more than 4 corners are detected then try more iterations with tuning parameters to get exactly 4 . However , the default parameters are experimented to handle most of the different test cases , Iterations is done for extremely noisy images

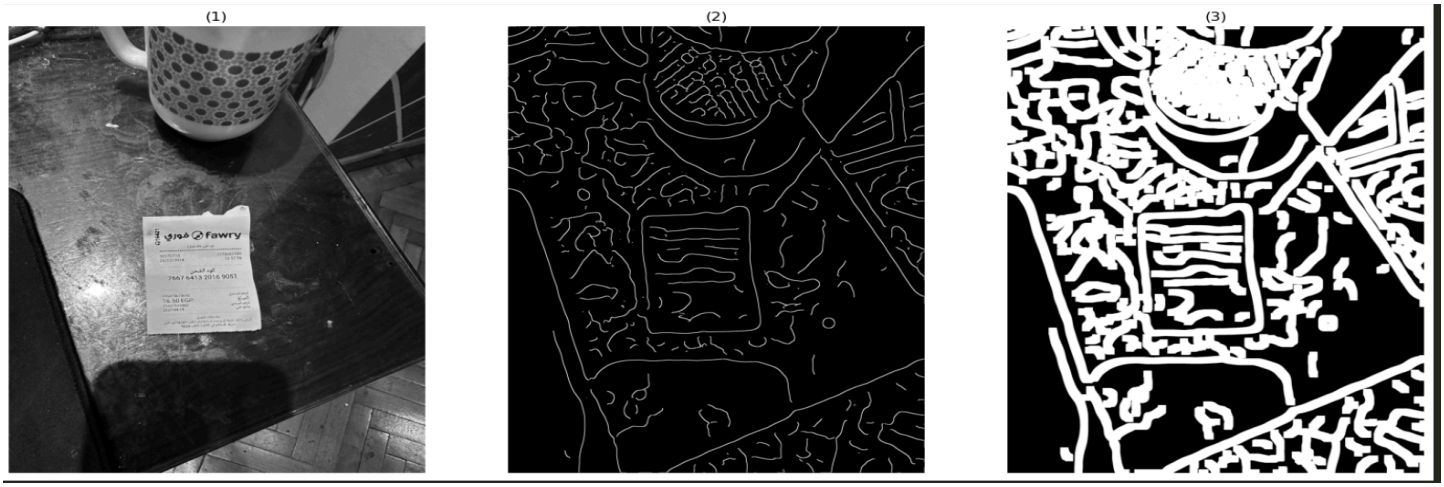
h) Using the four corners , apply Perspective Transformation that fixes the perspective view of the receipt as well as fixing the slight skews in the card .



Problem

Depending only on the edge detection to find the receipt contours was experimented to be greatly affected by noise , for example if the background contains other objects or lightning , the sigma in canny's edge detector wouldn't be sufficient to remove the noise .

For the following image , notice the detected edges after the canny's edge detector in image (2) . The receipt's contour is not connected which will never be found by finding contours , and applying dilation here to connect the edges resulted in a more noisy image (3) which we cannot process to the next step to find the card contour.

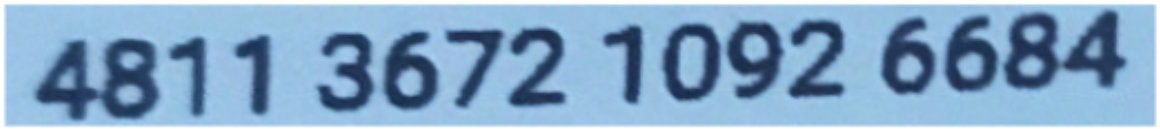


Enhancement

we used the fact that the receipt is a white area and used k means for segmentation, the one used in approach 1, before which decreased the percentage of error for noisy images and the contours are much better

2) Cropping the 16 digits Number

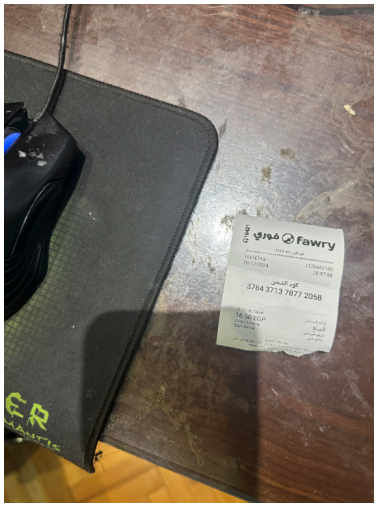
a) Apply Erosion and Dilation to the transformed image to get the 16 as digits a single contour
b) get all contours and filter by the aspect ratio in addition to the the contour width to get only the correct contour
c) Crop the image and return the 16 digits as one single image
d) Find Contours again on the resulted image to get each digit image separately for the template matching module



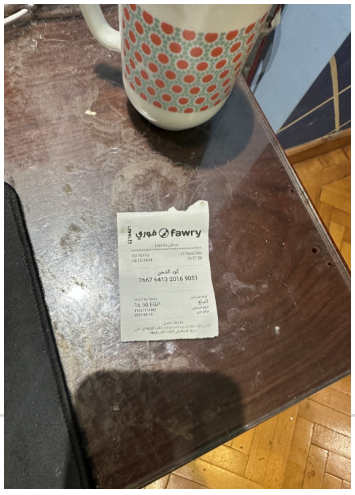
Performance & accuracy

1) The selected approaches effectively handle standard test cases without noise or minimal skewing. However, they encounter challenges with severely skewed images or those affected by intense flash lighting.
2) The segmentation techniques accurately extract the 16-digit code but occasionally struggle with the price, as the transformed image sometimes crops part of the price, preventing successful template matching.
3) These corner cases were intentionally introduced to evaluate the robustness and accuracy of our methods under extreme conditions. In practical applications, users aiming to extract numbers from a card are unlikely to capture photos that are heavily rotated or taken from extreme perspectives.

here's an example of our two failing test cases :

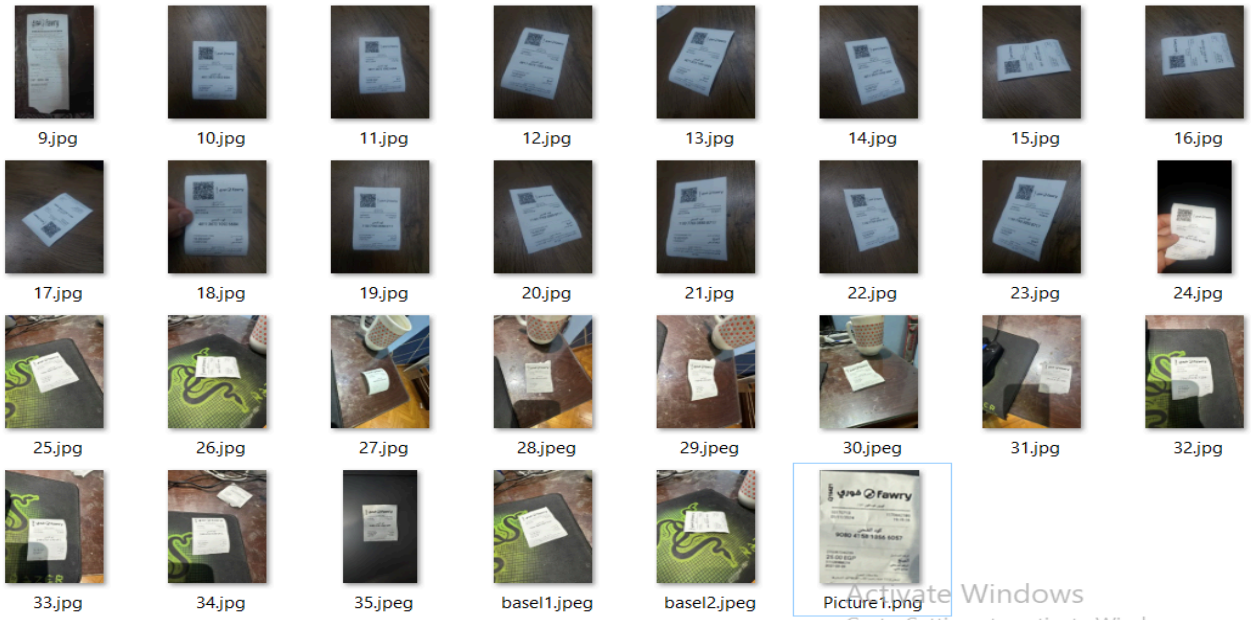




However , our code successfully passes these test cases :


Measuring accuracy

On a given dataset of 30 receipts taken from different perspectives , we have runned out project against Pytesseract's pretrained OCR and calculated recall , accuracy and precision



Results of this test set :

- Total image considered : 15
- Skipped Images: 15, 16, 17, 18, 24 → > 90 degrees rotated images
- Error Images: 27 → couldn't find card contour.
- **Correct Predictions**:
  - **Image 11**: Predicted = Expected
  - **Image 12**: Predicted = Expected
  - **Image 13**: Predicted = Expected
  - **Image 14**: Predicted = Expected
  - **Image 19**: Predicted = Expected
  - **Image 20**: Predicted = Expected
  - **Image 21**: Predicted = Expected
  - **Image 22**: Predicted = Expected
  - **Image 25**: Predicted = Expected
  - **Image 28**: Predicted = Expected
  - **Image 32**: Predicted = Expected
  - **Image 35**: Predicted = Expected

Total Correct Predictions: **12**

Results:

- **Total Processed Images**: 15
- **Correct Predictions**: 12
- **Accuracy**: **80%**

	Our Implementation	Pytesseract
Accuracy	80 %	90%

How to operate

Method 1 :

1) You can use the GUI by running this command *:*

   streamlit run .\code\gui\app.py in the root project directory

   (detailed packages needed to run is provided in the readme file)

2) Drag your receipt image
3) wait for a couple of seconds until the image is processed and you will get the 16 digits code and the price . (the price is a plus , it's not always gets extracted in noisy images)

Method 2 :

Open Fawry.ipynb notebook and add the path to the image you want to test in the testing cell provided .

Work Division

Basel Hossam

- Extracting receipt using colour segmentation (approach 1)
- Cropping receipt to obtain 16 digit number using M1
- Cropping receipt to obtain price using M1
- Applying template matching to detect price and 16 digits

Ahmed Tarek / Heba Gamal

- Extracting receipt using edge detection (approach 2)
- Cropping receipt to obtain 16 digit number using M2
- GUI
- Applying pretrained model and testing against it

M1 and M2 are 2 different techniques used for cropping, meaning there exist 2 different codes

Research Papers, References and links

- pyimagesearch.com: credit-card-ocr-with-opencv-and-python
- Template Matching Advances and Applications in Image Analysis: https://arxiv.org/abs/1610.07231
- An Overview of the Tesseract OCR Engine: 33418.pdf
- https://stackoverflow.com/questions/7263621/how-to-find-corners-on-a-image-using-opencv
- https://stackoverflow.com/questions/64860785/opencv-using-canny-and-shi-tomasi-to-detect-round-corners-of-a-playing-card

Needed non-Primitive Functions

Libraries

- Numpy , Skimage, PIL, CV2, Matplotlib, OS

Deep Learning

- Pytesseract OCR

Functions Proposed to be used in phase 1

- Binary_erosion() , Binary_dilation()
- Cv2.equalizehist()
- Cv2.resize()
- Find_contours()

- Logical_and()
- Rgba2rgb() , Rgb2gray()
- Imread() , Imshow() , Imsave()
- Cv2.dilate()
- Cv2.findContours()
- Cv2.wrapPrespective()
- Convolve2d()
- Canny() , Gaussian() , bilateralFilter()
- cv2.minMaxLoc()
- cv2.matchTemplate()

Extra needed Functions

- cv2.kmeans()
- cv2.getPerspectiveTransform()