COMP 4801 Final Year Project

An Easy-to-use Mobile Application For Personal Finance

Lee Shun Cheung - 2013541948
Abstract

To many people, receipts are just scraps of paper that can be immediately thrown into the trash can. However, some people need to keep them for purposes like warranty, tax deductions or reimbursements and to these individuals, it is vital to efficiently catalog as well as store the receipts. The digitising process can be manual or automatic, and the latter is achieved through software that offer Optical Character Recognition (OCR) functionality. OCR software transforms an image to software friendly representations, which can then be manipulated for other purposes. It must be applied to a digital image and while not everyone will have a scanner with them at all times, they will most likely carry around a smartphone. Unfortunately, existing solutions do not offer the convenience consumers expect and the accuracy of such products is not great.

As such, the proposed product to be created is an Android application that will automatically extract relevant information, such as date, time and amount purchased from images of receipts using 3 machine learning algorithms (K-Nearest Neighbours, Support Vector Machines and Neural Networks). Unlike currently available products, this application will perform optical character recognition in real time locally as well as convert extracted information to graphs and tables. While optical character recognition works for both written and printed text, this project will only focus on printed text as it is unlikely that receipts will be hand written. Furthermore, only the English language will be supported. At the time of writing, both the K-Nearest Neighbours and Support Vector Machine models have been created. Once the Neural Network model has been implemented, they will be migrated to the Android application.
Acknowledgements

I would like to thank Dr. C. Wu for her continued support and suggestions thus far. They have been very constructive and made it easier to create the Optical Character Recognition Android application. I would also like to thank Mr. Patrick Desloge for his critiques and comments on the initial Project Plan and Intermediate Report. They have allowed me to locate areas for improvement to write this Final Report.
# Table Of Contents

1. List of Abbreviations ................................................. p. 4
2. Introduction .............................................................. p. 5-6
3. Types Of Machine Learning Algorithms ......................... p. 6
4. Optical Character Recognition Research ......................... p. 7-8
5. My approach to Optical Character Recognition ................ p. 9-11
6. Scope of project ..................................................... p. 12
7. Deliverables ............................................................. p. 13
8. Overview of experiment setup ...................................... p. 14-22
   a. Part 1.1 Image Preprocessing .................................. p. 14-17
   b. Part 1.2 Training models ....................................... p. 17-21
   c. Part 2 Android application ..................................... p. 21-22
9. Current Status ....................................................... p. 22
10. Difficulties / Limitations encountered .......................... p. 22-25
11. Conclusion ........................................................... p. 25
12. List of Figures ....................................................... p. 26
13. List of Tables ....................................................... p. 26
14. References ........................................................... p. 27
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR</td>
<td>Optical Character Recognition</td>
</tr>
<tr>
<td>K-NN</td>
<td>K-Nearest Neighbours</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Networks</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open source Computer Vision</td>
</tr>
</tbody>
</table>
Introduction

This final year project relies on Optical Character Recognition, and so it would be important to begin by discussing its importance and history. OCR is the process of converting images of printed characters into machine-readable format and there are 3 main steps involved in this process:

1. **Document Layout Analysis** - Essentially figuring out where the text is located in the image.
2. **Character Recognition** - Determine what characters / symbols the text represent
3. **Information Extraction** - Group characters to words / sentences and convert text to meaningful representations such as graphs.

Traditionally, this process was performed by humans manually and OCR was created to improve both the speed and accuracy of digitising content. There are many practical applications, for example, old books can not only be preserved, but also shared to anyone with an internet connection. In turn, the digitised books can be converted to an audio file so that blind people can make use of it. Even the United States Postal Office have incorporated OCR to automate the mail handling process[1].

When OCR was first introduced in the 1930s, it was mainly used to help the visually impaired to read[2]. However, specific hardware had to be purchased and only a limited amount of fonts were supported[3]. Although the English alphabet only has 52 characters (lowercase and uppercase), the accuracy of such devices was quite poor. The main reason for that is because computers view images as binary.

![Human Vision VS Computer Vision](image)

**Figure 1:** Comparison of what humans and computers “see” when they are given an image
A single variation to any character, like a smudge or a crease, led to inaccurate predictions. In order to account for such variations, machine learning algorithms were integrated to OCR software. Machine learning is best explained using an example. Let’s assume there is a folder containing images of apples as well as oranges and a program needs to be able to recognize which category each image belongs to. It is impossible to account for all variations that may occur as factors like size, color and shape will all affect how the fruit is classified.

![Figure 2: Demonstration of the variety of oranges and apples](image)

Rather than coding countless lines just to distinguish whether an image contains an apple or orange, the program is given some images of apples and oranges respectively. Using those images, the program will associate patterns and extract other relevant information based on the choice of machine learning algorithm. This process forms the basis of knowledge for the program, which will allow it to be able to predict whether new images contains an apple or an orange. Much like how a child is able to recognize images of giraffes after a few examples, programs “learn” with the data, hence the name “Machine Learning”.

**Types of Machine learning algorithms**

There are many algorithms in Machine Learning but they can be classified into 3 main categories:

1. **Supervised Learning** - There is a specific label / answer for each data. Every time new information is provided to the program, it will estimate the best label / answer. An example would be classifying emails as spam or not spam.
2. **Unsupervised Learning** - The algorithm is expected to extract meaningful patterns and other important information from the data. Unlike supervised learning, there is no “correct answer” and the output has to be further processed manually to determine whether it is useful or not. Movie recommendation programs would make use of this type of algorithm.
3. **Reinforced Learning** - The algorithm is given specific instructions to follow based on certain data. The instructions serves as guidelines for the program and allow it to explore different aspects of the data. Programs that play board games are likely to use this type of algorithm.
Optical Character Recognition Research

OCR uses supervised learning algorithms. As there is no absolute silver bullet for this problem, researchers have used a variety of algorithms and evaluated the accuracy. Some popular machine learning algorithms used include decision trees and naive bayes classifier.

1. Decision Trees

Decision trees are a type of classifier that use a series of “questions” in order to determine which category a data belongs to.

![Decision Tree Diagram]

Figure 3: A demonstration of how Decision Trees work

Due to its simplicity, the models can be easily visualized, which allows humans to check the model for potential errors. However, this model is prone to overfitting, which is the phenomenal in which the model works very well for the training data set, but the accuracy declines dramatically when testing on other data. Furthermore, they can be unstable as slight variations of the data could modify the structure of the whole tree. Data must be carefully preprocessed in order to achieve acceptable results. Many online resources[4] provide datasets that have already been preprocessed and so this problem will not exist, hence its popularity with researchers writing academic papers. Unfortunately, in the real world data will not be so organized and it may lead to some classes dominating the model and creating inaccurate predictions.
2. Naive Bayes Classifier

Unlike Decision Trees, Naive Bayes Classifier works by estimating the probability that new input data belongs to a certain category based on already collected data. Naive bayes is order of magnitudes faster compared to other machine learning algorithms because it assumes that features of the data are independent. This means the model only has to do some basic multiplication and summation in order to produce the probability. Unfortunately, this classifier also has its limitations. If the selected features of the data are not independent, then the classifier will not produce accurate probabilities.

Irrelevant of which machine learning algorithm used in OCR, one common recurring theme is that they all only use one machine learning algorithm[5, 6] when implementing optical character recognition. There are papers that compare the performances of different machine learning algorithms [7, 8] yet they do not suggest an algorithm that stands out from the crowd. Due to this uncertainty, both companies and researchers will often select one algorithm and then optimize the code to create their unique algorithm.

My approach to Optical Character Recognition

While existing products are able to achieve decent accuracies for OCR, there is definitely room for improvement. To achieve this goal 3 machine learning algorithms, K-Nearest Neighbours, Support Vector Machines and Neural Networks, will be used. The 3 models will work independently and majority voting will decide what the final answer will be. In the event that all 3 models produce different results, Neural Network will be paired with either K-Nearest Neighbour or Support Vector Machine. A more detailed explanation will be provided below.

1. K-Nearest Neighbours

The K-Nearest Neighbours model works by first assigning a number to the variable “K”. All new data will be compared to the “K” nearest neighbours and majority voting will decide which category the data belongs to.
Figure 4: Visualization of K-NN. If K is set to 3, then the new data belongs to Class B. However, if K is set to 6, then the data belongs to Class A.

2. Support Vector Machines

Support Vector Machine models will only consider the data points closest between 2 classes. A boundary line will be created that ensures the gap between the two classes is as wide as possible. All new data points will be classified depending on which side of the line it falls into.
3. Neural Networks

Figure 6: A simplified visualization of neural networks.

Neural Network models have 3 main sections: input, hidden and output. Each input node represents one feature of the data, the pixel of an image, for instance. Each output node corresponds to a unique answer, like an alphabet character. The hidden nodes do not have meanings by themselves, as it is the connections between nodes that make them important. Every time the model is trained, the weights connecting the input and hidden nodes are updated. New data will trigger certain hidden nodes to activate, which will then affect the final output of the model. The main advantage for this model is that it outputs probabilities of each output node rather than providing a single answer like the previous 2 models. For example, if the output nodes correspond to the 10 digits from 0 to 9, the output will give the percentage that each output node is correct. The output can then be determined accordingly.

<table>
<thead>
<tr>
<th>Output Node</th>
<th>Percentage it is the correct answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3%</td>
</tr>
<tr>
<td>1</td>
<td>76%</td>
</tr>
<tr>
<td>2</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 1: Examples of the output from a Neural Network. The percentages are made up but it is important to note that they do not necessarily add up to 100%.
If all 3 models produce different results, then the answer with highest probability from either K-Nearest Neighbours or Support Vector Machine will be used. This ensures that at least 2 models are used to make a decision.

Scope of Final Year Project

Having discussed the proposed approach for OCR, it would be suitable to define the scope for this final year project. Only the Android version of the mobile application will be created, and it will only support the English language. This is due to the limited number of characters and symbols required for training (96 at time of writing).

![Figure 7: The characters that will be used for training the 3 models.](image)

Furthermore, only printed characters will be supported as most receipts will be printed rather than hand-written. The Android application itself will allow users to take photos or select one from the gallery of their phone. It is assumed that users will only use photos of receipts and the application will attempt to automatically extract relevant information from the images like date, time, amount spent and location. The data will be saved locally and can be converted to graphs for easier viewing.
**Deliverables**

Given the scope of the application, below is the list of the detailed deliverables for this project:

1. **Optical Character Recognition code in Python with OpenCV library**
   a. Image Preprocessing code
      i. Text location
      ii. Text separation
   b. K-Nearest Neighbour python implementation
   c. Support Vector Machine python implementation
   d. Neural Network python implementation

2. **Android Application**
   a. Convert Optical Character Recognition code (part 1 a-d) to Java
   b. Code necessary to display graphs in android application

---

**Figure 8: Graphical Representation of the deliverables.**
Overview of Experiment Setup

From the figure above, it can be seen that there are 2 main parts for the project. The following will outline the detailed steps involved in achieving them.

Part 1 of the deliverables can be visualized in the diagram below.

![Diagram of Part 1 of the deliverables]

**Figure 9: Part 1 of the deliverables**

All 4 components will first be tested using a computer so they will be coded in Python using the Open Source Computer Vision (OpenCV) library. OpenCV is an open-source library that is available in C++, C, Python and Java. This is useful because Python allows for rapid prototyping, while Java is compatible with Android. If necessary, C can also be used for IOS development in the future. While there are other machine libraries and frameworks available, OpenCV was chosen because not only has it been released for 16 years, but also it offers support for a wide range of machine learning algorithms, instead of focusing on only one or two. Although the latest version is OpenCV 3.1[9], version 2.4.13 will be used instead because it is the latest stable release (i.e. it has been tested and there are relatively less bugs).

Other than the 3 machine learning algorithms discussed previously, an additional image preprocessing script will be created. The reason for this is that the images will be taken using the built in camera of the smartphone. The image quality and the rotation of the image will need some modifications in order to optimize the accuracy of the models later on.
Part 1: Optical Character Recognition code in Python

1 Image Preprocessing

This step is actually made up of 2 subcomponents: Text location and Text separation.

1.1 Text Location

Text location is finding the text of the image. Since the final product will involve users taking a photo, it can be reasonably assumed that pictures will not be perfectly straight and aligned. To minimize the angle variations, the mobile application will have 4 predefined corners for the user to align the receipts with. Unfortunately, users may upload their own photos in which the corners are partially covered. In this case there will be a draggable component for the user to manually mark the corners of the receipt.

Figure 10: An example of the user interface for the user to manually mark out the corners of the receipt.
1.2 Text separation

After the text has been found, the program will rotate and scale the image so that it is in a top-down view. This step ensures all the text is horizontal and will make it easier for the program to recognize individual characters later on.

Figure 11: An example of how the image will be transformed to a top-down view.

Since OpenCV works best when searching for text in white against a black background so a process called thresholding will then be applied to the image. This is simply changing every pixel of an image to black or white.

Figure 12: Flipping the black and white colors of an image for easier recognition.
Before isolating individual characters for recognition, related text will be grouped together.

![Image](image_url)

**Figure 13: Extracting groups of text makes it easier to form words later on.**

This is necessary because the program will not know anything about the image. By grouping nearby characters, the information represented can be guessed later on. For example, groups of numbers will most likely represent a telephone number while the presence of the “at” symbol (@) should represent an email address. Groupings will then be further separated into individual characters for recognition.

This is the end of the image preprocessing step. Words have been identified and they have been separated into individual characters. Now it is time to train the 3 models.

![Image](image_url)

**Figure 14: Annotated diagram for part 1 of deliverables.**
2. Data for training the 3 models

As mentioned in the scope, the models will be able to recognize 96 characters. To train the models, different fonts of a character will be supplied to each model.

Figure 15: Examples of the fonts that will be used for training the 3 models.

Not only that, to maximize the accuracy of the training, some transformations will be applied to each character.

Figure 16a: Original Image

Figure 16b: Image after erosion

Figure 16c: Image after dilation
The transformations are applied to mimic real life scenarios where the characters are certain amount of noise attached. One last step is to

2.1 Training K-Nearest Neighbours

The accuracy of this model is affected by the value of “K” selected. However, other factors such as distance and weight will also affect the final decision.

If we set the K value to be 7, then the new data point should be a square based on majority voting. At the same time, it is extremely close to the 2 triangles and so it may be better to assign the data...
point as a triangle instead. The model parameters have to be tuned as more training data is introduced to ensure the model achieves maximum performance.

### 2.2 Training Support Vector Machine

In the previous introduction of support vector machines, it was mentioned that there are boundary lines that separate the classes. Each new data point will fall on either side of the line and be classified accordingly. However, other than straight lines, it is possible to use polynomial curves to separate the classes. The choice of boundary is also known as the kernel choice.

![Figure 18: Examples of the different kernels available for Support Vector Machines](image)

Each type of kernel will have a different effect on data separation and so they will all have to be tested to figure out which is best for the model.
2.3 Training Neural Network

The accuracy of neural networks depend on many factors, including number of hidden layers, number of input nodes and choice of features. The parameters will be varied independently to determine the optimal combinations.

At this point part 1 will be completed.

Figure 19: Annotated diagram of deliverables.

Part 2: Android application

Figure 20: Simplified diagram for Part 2 of deliverables
Part 2 begins by converting the code from part 1 to Java first. This is necessary as Android applications run using Java. Once the code has been converted the rest of the Android application can be completed. The application will have functionalities including but not limited to:

1. Take a photo
2. Select photo from gallery
3. Image preprocessing
4. Extract info from image
   a. Amount spent
   b. Category of receipt
      i. Add custom category
      1. Set icon of new category
   c. Items purchased
   d. Date
   e. Time
   f. Location
   g. Payment method
      i. Cash
      ii. Card
      iii. Bank
      iv. Octopus
      v. Others
5. Edit information extracted
   a. Add additional information like a note
   b. Add a tag for filtering
6. Manually input spending without receipt
7. Set budget for each category / all
   a. Receive alert when spending amount approaches budget
      i. Set custom percentage target
   b. Weekly, Monthly, Yearly
8. View spendings
   a. As list
   b. As a graph
      i. Pi chart
      ii. Bar chart
   c. Filtered
      i. By category
      ii. By time period
      iii. By location
iv. By time  
v. By date  
vi. By amount spent  
vii. By payment method  
viii. By custom tags  
d. Edit a spending record  
e. Delete a spending record  
f. Search existing receipts  
9. Save data into device (NoSQL)

The Android application is called “Receits”, spelt without the letter “p” because the motto of the application is “It’s just like ReceiPts… without the Paper hassle.”

People who download this application are tired of storing and managing paper receipts so it is aimed at providing an alternative for them. Every function of this application is done offline, so it was necessary to save user’s information locally. Realm, a NoSQL library was chosen instead of the standard SQLite because of 2 main reasons: package size and speed. NoSQL databases do not store the information in the standard table and work directly with the objects. This results in a smaller APK size, which is important if the user’s phone does not have a lot of storage. Furthermore, this direct manipulation of objects reduces the need to copy and save data to and from the database, and creates a more fluent experience for the user.
As the application has many functionalities, only some screenshots will be shown to illustrate the basic usage. The application itself has a detailed onboard tutorial for first time users and its interactive format will serve as a better guide than this textual description.

Users can decide between adding a new transaction with or without a receipt.

In the first installation the user needs to grant permissions in order for the application to work.
Once they grant the permissions they can either select a photo from the gallery or take a new photo.

Irrespective of what choice they decide, users will view this form where they can fill in relevant information. At the bottom is the optional “Smart Scan” whereby the application will attempt to automatically extract relevant information from the image to fill in this form. Of course, users can still modify the form before creating a new transaction.
Example of the date dialog that users use to select the date of transaction.

After creating a new transaction they can view them as a list, edit them or delete them.
Experiments and Results

The main functionality of this Android application is to allow users to take a photo of a receipt and it will automatically extract relevant information from the photo. As such, image preprocessing and information extraction were the 2 main points of focus. Image preprocessing involves enhancing the image so that it will be easier for the algorithm to analyze the image. The methods used included Gaussian Blurring, Erosion, Dilation and Adaptive Thresholding. Before applying these methods, the image was converted to a grayscale image. The reason is color images store 4 channels of data (Red, Green, Blue and Alpha) for each pixel whereas grayscale images only store 1 integer, a value between 0 (Black) and 255(White). As receipts are typically also black and white, it would be a waste of resources to analyze a colored image when a simple grayscale image is enough. After the conversion, Gaussian Blurring was first applied.

Gaussian Blurring is like adding a blurred screen on top of an image and it may be counter-intuitive to do this but it is useful because the receipt images will most likely be taken using a phone camera. This results in random specs due to dust, creases and lighting. Blurring the image will blend surrounding pixels together, which in turn significantly reduces the number of specs. Of course, blurring an image too much will make the image unusable, so different kernel shapes and sizes were experimented with.

```
# Rectangular Kernel
>>> cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)
```

```
# Elliptical Kernel
>>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)
```
The results were that a 3x3 rectangular matrix blurred images enough to remove most of the specs, without making the image unreadable. After blurring the next step was to erode the image. Receipts are printed on thermal paper and often companies employ the cheapest alternative to accomplish this task. This results in receipts printed with text that is blended together, like a smudge. Erosion separates the text by “shrinking” connected blobs of colors. This process will result in isolated characters but it may sometimes destroy the characters, for example turning the character “m” into 2 closely placed “n” characters. To solve this, dilation is applied. It is the opposite of erosion, which “expands” connected blobs. Hence, a combination of erosion and dilation was necessary to isolate characters without separating them too much. The final process used was

Erosion -> Dilation -> Erosion -> Dilation -> Erosion -> Erosion -> Dilation.

The first 4 erosions and dilations used a rectangular kernel of size 1. This ensured the image was modified little by little without affecting the image too much. The last 2 erosions and dilation used a cross-shaped kernel with size 3, to apply a final touch to the isolation. In some cases, the characters were still slightly touching or overlapping but this was okay because a secondary isolation process was to be applied later on. Once the characters were sufficiently separated, adaptive thresholding was applied to the image. Adaptive thresholding uses an integer called Block Size, as a reference for how many neighbour pixels to visit. Each neighbour pixel has a weight and an average value is calculated from the sum of their weight and pixel intensity. A constant is then subtracted from this average and if it is greater than a threshold value, the pixel is assigned the value 255 (black), otherwise 0 (white). The result is a binary image where each pixel is either black or white. At this stage the receipt image should look like a computer scan.

However, this is not enough because not every character or symbol is connected. The lower case letter “i” for example, would be considered as 2 characters: a dot and the number 1. A simple solution would be to perform a vertical dilation so that these characters would be merged together. However, as receipts are tightly packed, this would result in merging of characters between lines. The solution used was to first apply a horizontal dilation to the image. Each connected component

# Cross-shaped Kernel

```python
>>> cv2.getStructuringElement(cv2.MORPH CROSS, (5, 5))
array([[0, 0, 1, 0, 0],
      [0, 0, 1, 0, 0],
      [1, 1, 1, 1, 1],
      [0, 0, 1, 0, 0],
      [0, 0, 1, 0, 0]], dtype=uint8)
```
represented a row, and the sub-image was cropped from the original image. Each row could then be vertically dilated without worrying about them merging with other rows.

Modern smartphones take images at 12 Megapixels and it would be extremely slow to analyze each one of them during the OCR process. Thus, each row was downsampled before passing it to the OCR algorithm. The downsampling ratio was calculated by taking the minimum of the height to width ratio and then using OpenCV’s resize method to create a new image. Before downsampling, each receipt image took around 2 to 3 minutes to process. After downsampling, receipts with little information were processed between 10 and 42 seconds.

OpenCV works best with horizontally aligned images but it is impossible to guarantee that every image the user supplies is perfectly aligned. There may be missing corners, or the receipts may be angled. In order to align the images, either the edges or the corners of the receipt had to be found. Both of these methods were experimented with and the results will be discussed. First, the edges of the receipt could be found by determining the rate of pixel density variations. High rates of changes correlates to the presence of an edge. OpenCV offers 2 algorithms to achieve this: Sobel edge detector and Canny edge detector for this purpose. The Sobel Edge detector makes use of the first derivative of pixel intensities to determine edges. Edge pixels have high rates of change so they will be the local maximum of the first derivative.

To be more graphical, let’s assume we have a 1D-image. An edge is shown by the “jump” in intensity in the plot below:
By analyzing the graphs, edge locations can be approximated. However, this method was not useful for this application because the paper receipts were taken on top of other surfaces like wood. This created a lot of misleading data as the patterns of the background affected the location of the true edges of the receipt images. Similarly, the Canny edge detector uses derivatives to predict the location of edges but the difference is that each pixel’s gradient is compared to its neighbors. Each comparison uses 2 threshold values to determine if the pixel is rejected or accepted as an edge pixel. This method resulted in better results but still suffered from the same problems as the Sobel edge detector so it was necessary to try and find a receipt using corner detection.

Corner detection builds on top of edge detection in order to get more accurate results. First, the Canny edge detector was used to determine all the edges in the image. Straight edges were calculated using the probabilistic Hough Transform. Probabilistic Hough Transform is an algorithm used to determine the presence of any shape that can represented mathematically, even if the shape is broken or distorted. In this case the target shape was a straight line in the form \( y = mx + b \). Once straight lines were found, corners were determined by the intersection points. These corners were then used to approximate polygons so as to reduce the number of vertices. The remaining corners were sorted in a circular format (top left, top right, bottom right and bottom left) and if it had 4 corners, this was determined as the image of the receipt. Sometimes, the corners were missing, perhaps due to it being cut off the image. In those instances the edge of the image was assumed to be a corner to fill in the missing point. After approximating the location of the 4 corners, a perspective transform algorithm could be applied to the image. The perspective transform algorithm used the centroid of all 4 points as a reference and then the smallest width to height ratio was calculated. The image could then be transformed using OpenCV’s perspective transform api to get a top-down view of the image. Of course, it would be easier to let users to manually select the corners / edges of the image but this method was not reliable as users may crop too much or too little of the image. Instead, different settings were experimented with and in the end the transform function was sufficient to approximate the location of the receipt without having the user to manually mark the corners / edges.
If the original image is of decent quality, the isolation process should be able to pick up individual characters and perform OCR by now. Unfortunately, does not find contours in any particular order so the characters had to be sorted from left-to-right and top-to-bottom using a custom algorithm. This involved comparing the coordinates of the contour boxes and sorting them accordingly. As mentioned before, OpenCV contours are just lines that represented connected components.

![Image of Canny threshold and contours](image)

This information cannot be passed to the OCR algorithm as our original training data was in rectangular boxes. Each contour was thus analyzed and processed. First, its area had to be greater than a target threshold in order to weed out the noise pixels that were still on the image. An approximate distance between the 2 extreme left, right top and bottom points was calculated to represent the width and height of the contour. A bounding box was then created to wrap this contour. As a rectangle can be represented using only 4 points, the intermediate points found were discarded to save memory resources. Each box contour was then resized to 20x30 pixels to match the ones of our training data and finally flattened to a 1d array of 600 values. Each array can then be passed to the machine learning algorithms for OCR.

However, even if the accuracy is 100%, the data returned is generally useless because the structure of the data is unknown. Receipts do not have a standard format so each company’s receipt will be different. However, there are patterns to these receipts such as:

1. Company name or logo is always at the top of the receipt. The logo may not work well with OCR but most receipts include the company name beneath it.
2. The date is always separated using a forward slash “/” or a dash “-”. The format of the date may be different, with some companies using day/month/year while others used year/month/day.
3. The time is always separated using a colon “:”.
4. The keywords for the amount paid is something like: total or subtotal.

Thus, each row was sequentially analyzed to find patterns such as those listed above. Unfortunately, many shops in Hong Kong use a mix of Chinese and English in their receipts so additional data like “合計” and “總數” had to be incorporated into the machine learning models to recognize the keywords. The most difficult and inaccurate data was the category that the receipt belonged to. A receipt for “Food & Drinks” is practically indistinguishable from that of “Transportation” unless there are very obvious phrases that indicate the category. The temporary solution was to determine the name of the shop and use the Levenshtein Distance Algorithm to perform fuzzy string matching.

![Damerau-Levenshtein](image)

The Levenshtein distance algorithm calculates the minimum number of edits required before a string is identical to the target. This allows us to determine how “close” a string is to the target. A dictionary of common stores was stored in the application and if a match was found, the receipt was assigned to a certain category. Otherwise, the mobile application waited for the user to manually input the category and this was saved for future use. Granted, this is not the optimal solution as it limits the number of shop-to-category pairs the mobile application is able to recognize but at the time of writing this method was the “best” to solve the problem. Once the process was completed it was passed back to the application to display to the user for editing before saving.
Difficulties / Limitations encountered

The most difficult part of the project was image preprocessing as well as OCR. As there is no online database for receipts, all testing images had to be collected by myself. There are mock receipt generators online but they were not useful because the images had zero noise and the format was all identical. Those receipt images created false positive results as the OCR algorithm achieved an accuracy of over 98% every time. Unfortunately, the receipt collection was limited so I could only test in on around 200 receipts.

As mentioned before, the original OCR algorithm was written in Python for testing on the computer and it had to be translated in Java for use inside the the Android application. Whilst the documentation of OpenCV for python is very detailed, the documentation for Java was poor to the point of non-existence. Extensive research had to be conducted and often I had to post questions on sites like Stack Overflow to determine the equivalent function calls. Some apis in Java used extra parameters and others used less so it took some time to get use to the new syntax. For example, the data structure np.array could be used in OpenCV. The equivalent version of this in Java is not just a simple array but a Matrix with the data type CvType.CV_32FC2. The official documentation for this is just the line “CV_32FC2 public static final int CV_32FC2”, without any explanation as to what or how we should use this variable.

Another difficulty was that the Mac OS X version of OpenCV performs differently to that of the Android java version. Due to limited resources, there is an internal cutoff as to how much time the software will spend on an image. This is to reduce the amount of time, and in turn, resources spent so even with the exact same code, the desktop version produced different results compared to the Android version. This caused me to have to manually tweak the different parameters over again in order to ensure the OCR algorithm produced accurate results.

Regarding the Android application, there were also some problems encountered. The Android application uses the open source library OpenCV. This library is written in C++ and was originally meant for desktop PC as it was released more than a decade ago. When smartphones came out, this library was in demand for mobile developers so a java and objective-c binding version was released. However, in order to help save space, the mobile library was released as an external apk. The idea was that users had to download this OpenCV app once and then all applications that made use of the OpenCV library would just call the api. This sharing would significantly reduce storage space because no matter how many apps made use of it, there would only be 1 version of the library. Unfortunately, this was going to affect the project because in my personal opinion, it is extremely annoying to have to download 2 separate applications for 1 purpose. Nevertheless, after some research and code modification, I was able to extract the relevant source code and package them together as an internal library, thus eliminating the need to download the separate app.
However, another issue then presented itself. Machine learning models had to loaded from the smartphone’s disk, not from the apk itself. Luckily, this was only a minor issue as the application only had to seek the user’s approval before copying the model files to the disk. As the files are less than 20 kilobytes in size, it didn’t waste a lot of the user’s existing space. Permissions in Android traditionally worked by asking the user’s approval before installation. Users could only download an application if they accepted all the permissions and if they did not want to grant the permission anymore, they had to delete the application. Unfortunately, Android Marshmallow changed this setting such that permissions were granted runtime. The idea behind it was so that users can grant permissions on the go, and only access partial functionality if they disapprove the application. For example, this android application uses the camera but if users do not grant the permission, then they should be able to continue using its other functions such as transaction recording or budget tracking. While this is a good idea for consumers, it meant a lot more tracking code had to be implemented. Before certain code could execute, permissions had to be checked or else the application would crash.

Some other minor troubles encountered during development include naming conventions for translation files. The Android documentation suggests creating a folder named “zh_hans” for storing simplified Chinese string.xml files but apparently the files had to be named “zh_Hans” with a capital “H” in order to work. When switching from one language to another, the documentation regarding the activity refresh was not detailed nor did it provide any concrete examples. I had to use savedInstanceState to pass a boolean between activities in order to ensure the translation worked properly. Xml resource arrays also had to be referenced using TypedArray rather than a simple integer array. There were many “gotcha” places where further research had to be done in order to fix the problem.

**Conclusion and Future work**

The purpose of this final year project was to create an Android application that is capable of performing optical character recognition on images of printed receipts. Existing products, especially the free ones, do not provide the convenience that consumers expect and so this Android application will serve as a free alternative to the expensive OCR software offered by companies. Although the application is able to perform decent offline recognition, a few improvements can be made. The machine learning models were trained on characters in about 205 fonts yet the accuracy is not at 100%. The erroneous assumption made was that the more fonts the models were trained with, the more accurate the OCR would be. However, the phrase “Garbage in, Garbage out” succinctly summarizes the fatal flaw in the assumption. The images provided to the application were not computer scans, which meant there was a lot of deformation and noise. Although deformation and noise was applied to each training character, the initial characters were all “pure” in the sense that they were images of perfectly aligned characters on top of a perfect background. This meant the
OCR could not function to its full potential with the provided images. Future work could involve training the models using a mixture of perfect and noisy characters extracted from the photos.

Although the image preprocessing eliminated most of the noise, the preprocessing can also be improved to make the input better. The current image preprocessing does not follow any certain technique or algorithm, and it is the result of trial-and-error. This means there should be more advanced and accurate methods of image preprocessing that can be integrated into the application.

Also, the use of 3 machine learning algorithms to perform majority voting was fairly accurate but in the future other methods such as deep learning could be explored. Although each model was trained and the parameters were tweaked to improve accuracy, they are nonetheless still standard algorithms. Future work could involve implementing a custom algorithm that extends one of these base models in order to significantly improve the accuracy of predictions. Apart from modifying the models used for predictions, a future version of the application could use a cloud server for recognition. Using a cluster of backend servers provides the necessary computing power to perform complex parallel calculations. Not only will the accuracy increase, but also the speed of recognition could be reduced. The backend servers could also accumulate knowledge from all the users to further improve the accuracy of the overall application.

Not only that, the current application only supports the English language. Although some common Chinese keywords were integrated into the models, it nonetheless only supports English as the main medium. Different languages could be integrated into the models so that users of different countries could also benefit from the application.

Of course, the other obvious extension would be to create an IOS application and a mobile-friendly website to accompany this Android version. This would require some sort of backend framework in order to support cross device synchronization as well as account creations.

A final minor point of improvement involves the user interface (UI) of the application. This project’s emphasis was on the functionality of the OCR inside the app, so relatively less time was placed on the UI. A little bit of animation was included but overall the application could benefit from some “makeup” as standard UI components were used. Although many users only care whether the application works and benefits them in some way, it is always important to consider the aesthetics as well. The look and feel of the application shapes the experience of the application, such that users should not feel a sense of chore when using it.
This final year project has provided me with a lot of practical experience in building a mobile application as well as machine learning. Although most academic projects will be forgotten or removed after graduation, I feel that this is one that I will continue to enhance as a side project. After all, it is an application that I would be needing in the near future.
List of Figures

Figure 1  Comparison of how computers and humans see images  p. 4
Figure 2  Examples of oranges and apples  p. 5
Figure 3  Example model of a decision tree  p. 6
Figure 4  Example model for K-Nearest Neighbours  p. 8
Figure 5  Example model for Support Vector Machines  p. 8
Figure 6  Visualization for neural networks  p. 9
Figure 7  Characters used for training the models  p. 10
Figure 8  Diagram of deliverables  p. 11
Figure 9  Diagram for part 1 of deliverables  p. 12
Figure 10 Example UI for Android application  p. 13
Figure 11 Transforming text to top-down view  p. 14
Figure 12 Applying thresholding to an image  p. 14
Figure 13 Isolating words in an image  p. 15
Figure 14 Annotated diagram for part 1 of deliverables  p. 15
Figure 15 Example fonts used for training the models  p. 16
Figure 16 Transformations applied to individual characters  p. 16
Figure 17 Demonstration of K-Nearest Neighbour model  p. 17
Figure 18 Support Vector Machine with different kernels  p. 18
Figure 19 Completed part 1 of deliverables  p. 19
Figure 20 Part 2 of deliverables  p. 19
Figure 21 Dollar sign with 2 strokes  p. 20
Figure 22 Image with lots of noise  p. 21
Figure 23 Image with bad lighting after transformation  p. 21
Figure 24 Receipts with text in curved logos  p. 22

List of Tables

Table 1  Example output for neural network  p. 9
References


3. Optical Character Recognition Software Sources - ABBYY [Internet]. [cited 2016Nov1]. Available from: http://www.abbyy.co.il/?categoryId=63424


