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 scripts. I would also like to thank Mr. Patrick Desloge for his critiques and comments on the initial Project Plan. They have allowed me to locate areas for improvement to write this Interim 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

OCR - Optical Character Recognition
K-NN - K-Nearest Neighbours
SVM - Support Vector Machines
NN - Neural Networks
OpenCV - Open source Computer Vision
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.

![Figure 3: A demonstration of how Decision Trees work](image)

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.

Figure 5: Example of a Support Vector Machine boundary line.
3. Neural Networks

![Neural Network Diagram]

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).

![Characters](image)

**Figure 7:** The characters that will be used for training the 3 models.

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 deliverables](image)

*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 has 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.

**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.

**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

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
   c. Filtered
      i. By category
      ii. By time period
      iii. By location
      iv. By time
Current Status

To date, everything in Part 1 has been completed. All models have been trained and the image preprocessing script has been implemented. This code is being converted to Java to be used inside the Android application. Once this code has been converted other functionalities of the Android application listed above can be created. Prototypes of the Android application will continually integrate functionalities and be verified with Dr. Wu to ensure it suits the intended requirements. The expected finalized mobile application is expected to be completed by the end of February / early March and the remaining time will be spent fixing bugs and modifying other aspects before releasing it to the Google Play Store.

Difficulties / Limitations encountered

Unlike handwritten digits, there is no repository of letters and symbols to just download and use. As a result, all training and testing data had to be self generated. Some special symbols like the dollar sign with 2 strokes (Figure 21) was omitted as they are not common in receipts.

Figure 21: Dollar symbol with 2 strokes.
For each symbol, different fonts were then generated using TextEdit in order to account for as many variety of receipts as possible.
Other than generating fonts, the image preprocessing step was much more difficult than anticipated. As images are taken using a smartphone, there will be a certain level of blurriness. This significantly decreased the quality of the image, which in turn made optical character recognition very inaccurate. Apart from blurry images, there was a lot of noise in images captured, due to dust, creases and other environmental factors.

![Example of an image taken using an Android Smartphone and processed using the image preprocessing script.](image1.png)

Figure 22: Example of an image taken using an Android Smartphone and processed using the image preprocessing script.

Lighting of the images also posed a problem, as images taken in dark environments were unrecognizable after transformations.

![Image after applying an OpenCV transform function.](image2.png)

Figure 23: Image after applying an OpenCV transform function.
Not only that, some receipts also had text embedded in logos or curved images. So far, those type of text had the worst accuracies and alternative measures will have to be explored later on.

![Figure 24: Example of receipts with text inside icons and logos.](image)

In addition to image preprocessing, there were problems with isolating individual characters. Text in OpenCV is found by creating bounding rectangles, called contours, around each individual character. These contours are found in a random order so it was necessary to manually sort them from left to right, and top to bottom. However, there is not default function for this and so a custom method had to be created for sorting. Moreover, characters like ‘i’ were sometimes not correctly isolated as they were separated into two characters that resembled the full stop (‘.’) and digit 1.

With regards to individual character recognition, some characters were often mixed together. Upper case “O” is similar to the digit 0, and the upper case B is similar to the digit 8. Even humans may have some difficulty separating these characters looking at images and so more data will have to be provided to the model to improve its accuracy. Sometimes, the characters extracted were actually actually random geometry shapes and graphics. This led to inaccurate predictions as the data collected was not useful at all.

Furthermore, a lot of shops in Hong Kong provide receipts with a mix of English and Chinese. Users may not necessarily find the application useful if they are only able to scan a portion of the
receipts. Handwritten text is also not supported, which means if the receipts contain written text, the mobile application may not produce enough useful information for the user.

Conclusion

The purpose of this final year project is 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 the final application created will be a free alternative to the expensive OCR software offered by companies. The Android application will make use of 3 machine learning algorithms, namely: K-Nearest Neighbours, Support Vector Machines and Neural Networks, which will hopefully increase the accuracy of the OCR process.
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


