

Jin Jin

# A Flash Card Android Application Development Applied with OCR technology

---

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

26 May 2014

Author(s) Title Number of Pages Date	Jin Jin A Flash card Android application development applied with OCR technology 34 pages, 26 May 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Mobile Application Development
Instructor(s)	Olli Alm, Lecturer
<p>The aim of the project was to build a flashcard Android application to help people memorize words. The major functions of the application include creating card books, adding or removing words from the created book by typing or Optical character recognition (OCR) and translating all the words automatically.</p> <p>The application was developed on the Android platform with Android SDK and tesseract OCR engine. An android word-storing application and a graph scanning application were combined to implement the project. In order to call the Tesseract OCR library, it is necessary to use Android NDK and Java Native Interface (JNI) to call the functions from the Tesseract library. Microsoft translation API was used to help users translate all the entered words to the expected language.</p> <p>The results of the project proved that using the OCR technology to digitize paper-based document was a feasible way to simplify the user input and it made archives searchable, editable and machine readable. With the increase of the camera's imaging quality, the accuracy of the OCR technology will perform more reliable results.</p>	
Keywords	Android, OCR, NDK, JNI, mobile application

## Contents

1	Introduction	1
2	Optical Character Recognition Theories	2
2.1	History of OCR	2
2.2	Primary OCR Workflow	3
2.2.1	Preprocessing	3
2.2.2	Feature Extraction	4
2.3	OCR in the Daily Life	4
2.4	Tesseract OCR Engine	6
2.4.1	Tesseract OCR Architecture	7
2.4.2	Line and Word Finding	8
2.4.3	Word Recognition	9
2.4.4	Training Data	10
3	The Flash Card Project	13
3.1	The Use Case	13
3.2	The Application Workflow	14
4	Implementation of the Application	16
4.1	Development Platform and Tools	16
4.1.1	Android	16
4.1.2	Tess-two Project	18
4.1.3	Bing Translation APIs	19
4.2	Development Environment Configuration	19
4.3	Class Diagram Of The Application	22
4.4	Implementation of the Components	24
4.4.1	User Interface	24
4.4.2	SQLite Database	27
4.4.3	Data Transfer	27
5	Project Evaluation	29
5.1	Application Testing and Result	29
5.2	Further development and possible improvements	29
6	Conclusions	31
	References	32

## 1 Introduction

This final year project is a mobile word-collection solution for my personal interest. It is a trial and practice of my developing skills in the open source community. During the past few years, the trend of digitizing paper-based material has emerged. Data and information would be much more valuable if they were available in digital form. Digitization technology has been applied in many fields in people's daily lives, such as retail businesses, post offices, insurance and aircraft companies[3]. The optical character recognition (OCR) is one of the most commonly used digitization technologies.

The OCR concept was firstly invented to help the blind to read [1]. Nowadays OCR software engines have been developed for various applications, such as receipt, invoice, check and legal billing [3]. The optical character recognition helps to reduce the human's jobs of manual handling of data.

In the final year project, I integrated the Tesseract OCR engine into the application, as it is open-source and free to use, released under the Apache License. The Tesseract OCR engine can read a wide variety of image formats and convert them to texts in over 60 languages [1]. The Tesseract OCR engine was initially developed at HP Labs and it has been extensively improved by Google [1].

Since there is very few of OCR solutions for a flash card application in the market, I chose this technology as the topic for my final year project. Flash cards are a group of cards with hint information on one side and answer on the other side. The thesis describes the development of the flash card Android applications and discusses the theories of the OCR technology, the process of software designing and developing, and the result of the project.

## 2 Optical Character Recognition Theories

Most people start to learn reading and writing during the first years of education. As long as they have finished the basic education, people should have acquired writing and reading skills. Regardless of the following situations such as: fancy font styles, misspelled words, fragmented parts and figurative or artistic design, most people are able to read and understand the contents by making use of experience and context. On the contrary comparing to the human being's reading skills, there's still a long way to go for machines to produce human-competitive text recognition.

### 2.1 History of OCR

Nowadays computer intelligence is designed to stimulate the human's behavior, such as reading, writing, listening and talking. In the early 1950s, scientists firstly tried to capture images of characters and texts by mechanical means, as shown in Figure 1. Then optical means of rotating disks and a photomultiplier, a flying spot scanner with a cathode ray tube lens, were followed by photocells and arrays of them [2]. Subsequently, the drum and the flatbed scanner were invented, which enabled OCR machines to scan the full page.



W. S. Pike, RCA engineer who helped develop electronic reader, moves its eye over text.

Figure 1. Early OCR Reading Machine Copied from Cheriet (2007) [2]

With the invention of the digital-integrated circuit, the scanning speed and conversion speed were highly accelerated [2].

In the early 1960s, various errors in the OCR were made in the case of poor print quality, wide variations in fonts and rough surface paper [2]. A leap happened in 1970s for the OCR applications. The American National Standards Institute (ANSI) and the European Computer Manufacturers Association (ECMA) designed fonts custom-built for OCR such as OCRA and OCRB [2]. The International Standards Organization (ISO) soon adopted the customized fonts [2]. Consequently, higher recognition accuracy was achieved. With all these accomplishments, the cost of high speed accurate OCR scanning was sharply lowered.

## 2.2 Primary OCR Workflow

Currently, in the OCR, it takes three primary steps for a captured image to be recognized, as shown in Figure 2: preprocessing stage, feature extraction stage and classification stage [2].

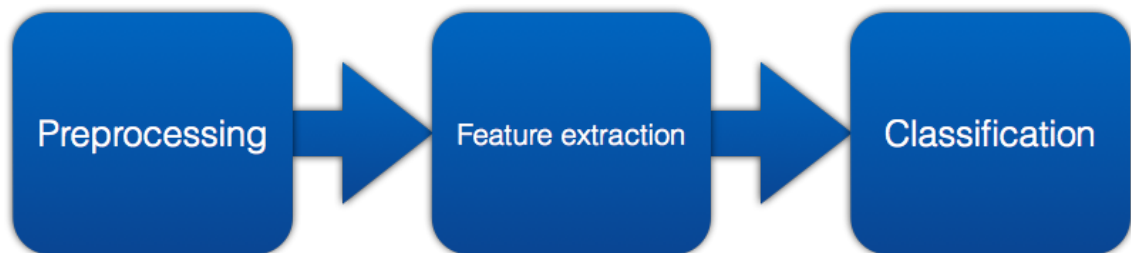


Figure 2. OCR workflow

The OCR stages are discussed in more detail below.

### 2.2.1 Preprocessing

After a raw image is taken, image preprocessing will be the first recognition stage. It is responsible for eliminating the uninterested data, positioning the interested regions, are enhancing the images to make the data efficiently used by the next step [2].

Normally, the following operations are performed in the preprocessing phase:

1. Thresholding: Converting the raw image into binary format.
2. Noise reduction: Applying proper techniques to connect or disconnect pixels.
3. Normalization: Normalizing the image to standard matrix.

### 2.2.2 Feature Extraction

The feature extraction stage is used to extract the most relevant information from the text image, which helps us to recognize the characters in the text [3]. The algorithms and techniques applied in this stage for selecting representative feature is the core of the whole recognition process and greatly affects the recognition quality. The most popular feature extraction methods can be categorized into geometric features (moments, histograms, and direction features), structural features (registration, line element features, Fourier descriptors and topological features) and feature space transformation methods (the principal component analysis (PCA), linear discriminant analysis, kernel PCA) [2].

The classification stage utilizes the features that were extracted in the previous step. It determines the feature space in which an unknown pattern falls. The classification step is performed by comparing feature vectors corresponding to the input character with the representative of each character class [4].

## 2.3 OCR in the Daily Life

The OCR has been extensively developed for over 60 years, and it has been applied in various fields [1]. OCR has helped human beings to reduce a huge amount of manual typing work. With the flourishing of cloud computing services, smart mobile devices and the release of the OCR engine in the past few years, The OCR is not only playing its role as saving human labors, but also providing better user experience. Now OCR is making its way to our daily lives. Some examples are shown as follows.

Google Translate camera input

In 2012, an OCR feature was added to the Google translate service, as shown in Figure 3, so that users could translate text using only camera lens.

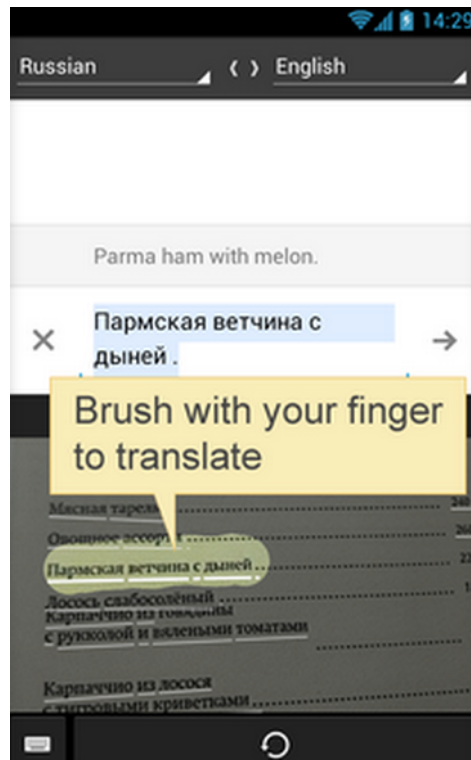


Figure 3. Google Translate for Android added OCR features

It provides users the convenience of quick input when they do not have an adaptive keyboard or in case when the text is too long.

Automatic License Plate Recognition (ALPR)

OCR has been helping the policeman for years [6].



Figure 4. An Example of ALPR User Interface copied from David (2012) [6]



The scanner above is a vehicle-mounted ALPR system, as shown in figure 4. It helps officers to be alerted when a vehicle on the wanted lists has been observed in the vicinity [6].

### OCR Passport Reader

OCR has been widely used in airport passport scanning [7].



Figure 5. OCR full-page multi-illumination desktop readers copied from OCR640 full-page multi-illumination desktop readers for ePassports and eID Cards (2014) [7]

It takes less than one second for the passengers and the security personnel to finish the identity check in the airport, as shown in figure 5 [7]. The whole process of checking and recording are automated. Since the data is digitalized, it provides the informationized way of further usage, such as movement record tracking [7].

## 2.4 Tesseract OCR Engine

One of the core difficulties in my final year project was to find an efficient and economical OCR solution for the application. There are numerous OCR engine/cloud service providers, such as ABBYY Cloud OCR SKD, Bing OCR and Alternatives, offering programmers easy approaches to develop the OCR application for many different fields. However, most of the OCR engines or service providers are commercial and closed-source [7].

Ver	Set	Character			Word		
		Errs	%Err	%Chg	Errs	%Err	%Chg
HP	bus	5959	1.86		1293	4.27	
2.0	bus	6449	2.02	8.22	1295	4.28	0.15
HP	doe	36349	2.48		7042	5.13	
2.0	doe	29921	2.04	-17.68	6791	4.95	-3.56
HP	mag	15043	2.26		3379	5.01	
2.0	mag	14814	2.22	-1.52	3133	4.64	-7.28
HP	news	6432	1.31		1502	3.06	
2.0	news	7935	1.61	23.36	1284	2.62	-14.51
2.0	total	59119		-7.31	12503		-5.39

Figure 6. The results of current and old Tesseract copied from Ray (2014)[8]

The Tesseract OCR engine is an open-source and the most accurate free-to-use OCR engine created by HP labs and extensively enhanced (as shown in Figure 6) by Google at present. The recent version was shown as 2.0 and the original version in 1995 was shown as HP. At the time this project was initialized, it was the only free Android toolkit available in the market.

#### 2.4.1 Tesseract OCR Architecture

The OCR process of Tesseract follows a staged pipeline, as shown in Figure 7.

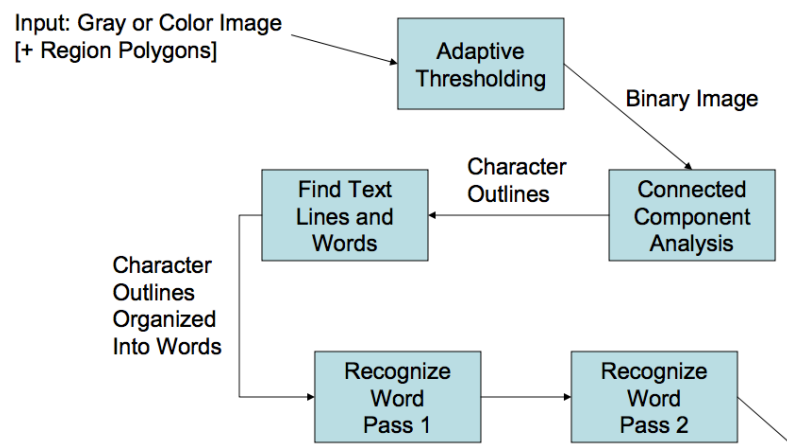


Figure 7. Tesseract OCR Architecture copied from Ray (2007)[9]

Unlike Google Translate smearing, Tesseract assumes that the input image is in a polygonal binary area. The outlines of the component are gathered into Blobs [8]. Blobs

are arranged into text lines [8]. (In OCR, Blobs refers to those areas on the digital image that are detected to be different from the surrounding regions in color or brightness.) Tesseract detects text area by its line finding algorithm.

#### 2.4.2 Line and Word Finding

Tesseract firstly filters out blobs that are smaller than some fraction of the median height, which could possibly be punctuation or noise. Then Tesseract processes the filtered blobs horizontally and assigns them into a text line. After blobs have been assigned, the filtered-out blobs will be put back into the appropriate position.

Once text lines have been detected, the baselines are fitted more precisely using a quadratic spline. The baselines are fitted by partitioning the blobs into groups with a reasonably continuous displacement for the original straight baseline [8]. A quadratic spline is fitted to the most popular partition, (assumed to be the baseline) by at least squares fit [8]. The quadratic spline keeps the calculation result at a stable level. However, multiple splines lead to discontinuities as a disadvantage. As shown in figure 8, Tesseract uses the baseline (green), the descended line (blue), the mean line (pink) and the ascender line (cyan) to locate the text line.

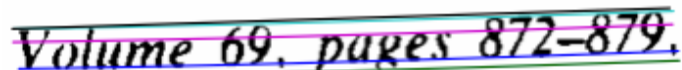


Figure 8. An example of the curved fitted baseline copied from Ray (2014)[8]

After the text area has been detected, Tesseract will segment the words into character pieces by analyzing if they are width equivalent, as shown in Figure 9.



Figure 9. A fixed-pitch chopped word copied from Ray (2014)[8].

When the fixed pitch text is found, the chopper on these words would be disabled for the word recognition step [8].

### 2.4.3 Word Recognition

Tesseract improves the unsatisfactory result by chopping the blob. It attempts to find candidate chop points from concave vertices of a polygonal approximation of the outline [8]. At least three pairs of chop points assure that a separate character is from the ASCII set, [8]. Figure 10 shows how Tesseract recognizes the character when the character 'r' touches the character 'm'.

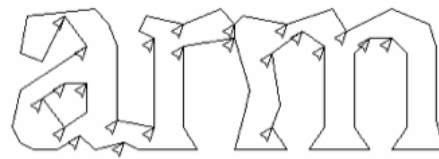


Figure 10. Candidate chop points copied from Ray (2014)[8]

If the chops are all used up and the result is still not satisfying, the analysis task would be taken over by the associator. The associator performs the best-first search and makes the graph into candidate characters. Figure 11 shows an example of the broken character.

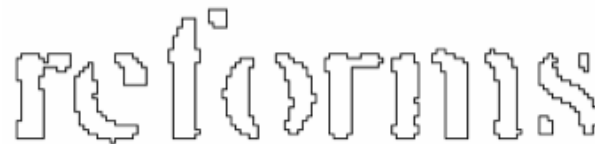


Figure 11. A broken word to be recognized copied from Ray (2014)[8]

The features of the broken character are extracted from the outline fragment by the static classifier.

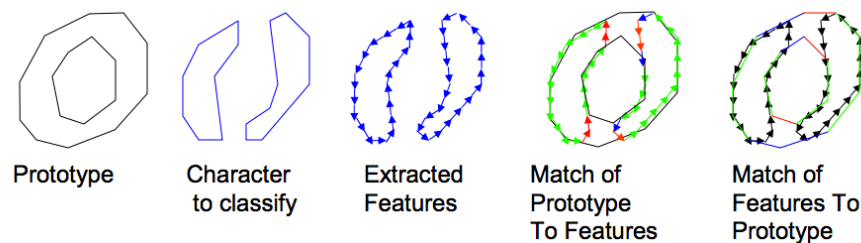


Figure.12 Features and matching copied from Ray (2007) [9]

The many-to-one matching helps Tesseract easily recognize the broken characters, (as shown in Figure 12) [9].

#### 2.4.4 Training Data

To improve the accuracy and extend the language limitation, the Tesseract engine is designed to be fully trainable. After character segmentation, a two-pass process is performed. In the first pass, an attempt is made to recognize each words in turn [8]. The satisfactory character is passed to the dynamic classifier as training data [8]. In the second pass, words that were not satisfyingly recognized are recognized again.

Users can train Tesseract to improve the accuracy of recognition for their own cases. Figure 13 is a digital image waiting for recognition.

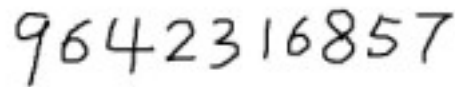
A digital image of the handwritten number 9642316857.

Figure 13. Sample Image to be recognized

Some of the characters do not match the expectation. Figure 14 shows that the result of recognition is not perfect.

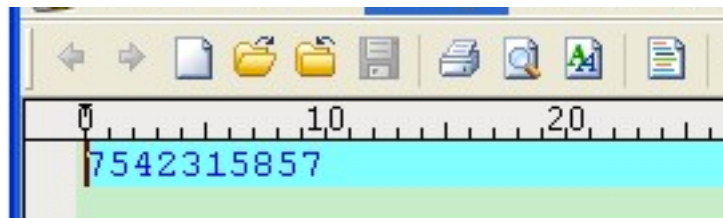


Figure 14. Recognition result without training

Hereby, I used a tool called jTessBoxEitor (<http://vietocr.sourceforge.net/training.html>) developed for Tesseract OCR training automation to illustrate the basic idea of Tesseract training. To improve the result, I took five text samples (the more the better) for training, as shown in Figure 15.

|234567890  
 |234567890  
 |234567890  
 |234567890  
 |234567890

Figure15. Five samples

With the sample image, I used the Tesseract shell command to generate a '.box' file, which is for recognizing the character and position of the words. Then I provided the file as an input to the jTessBoxEditor. The output is as follows in Figure 16. In this step, some of the character recognition results (the number '1' and number '5') were not what were expected.

In this case, I revised the wrong character and saved it to the result. After that I exported the language-training file and saved it in the Tesseract data library .

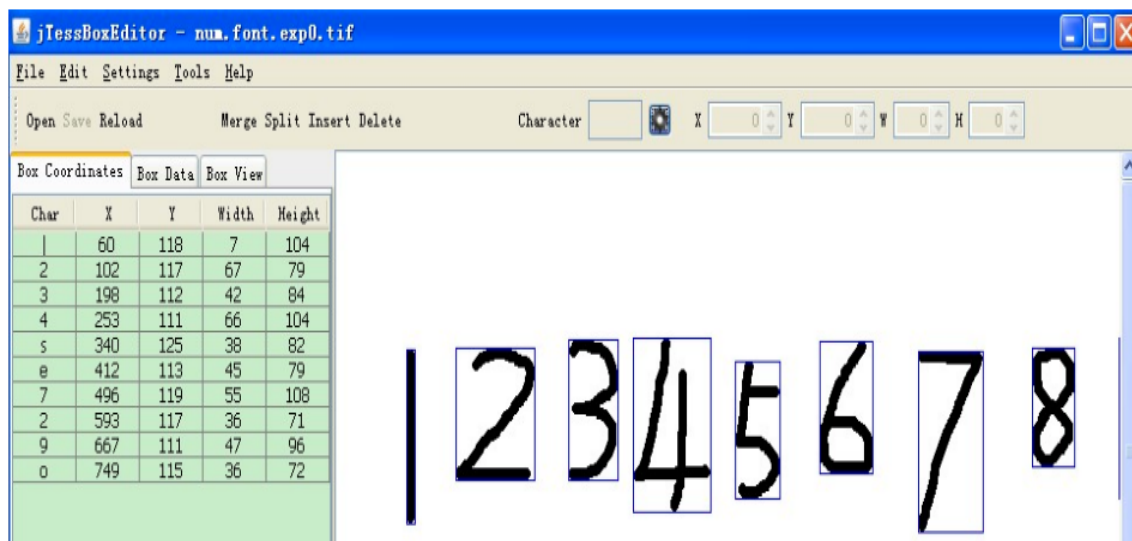


Figure 16. Revision of the recognition result for training

Thus, Tesseract was trained by being told the mistakes it made so that it would not make the same mistakes in later recognition.



Figure 18. The recognition result after training

Figure 18 shows the recognition result after Tesseract was trained. The result perfectly matches the expectation.

### 3 The Flash Card Project

Flash cards are a group of cards that containing hints information on one side and answers on the other side manufactured for study purpose. Flash cards are widely used for helping memorization. In this project, the flash card application acted a role as the word-retention tool. There are numerous flashcard applications in the market, but the motivation of this project was that I could barely find any flashcard application containing OCR input function. The most similar application I could find was in Google Translate, but it did not fit the use case of flash cards. Since the translation process of the Google Translate OCR process does not contain the word segmentation stage, the expected words can not be spited into a customized word list. So I decided to develop an Android application, which would both fit the Flash Card use case and be with OCR technology for input convenience.

#### 3.1 The Use Case

Figure 18 illustrates the steps that users follow in the Flash card application use case. When a user starts the application, he/she can either choose to enter words by a camera lens, or by manual typing, or to look back at the previous recorded words. Once the word-input step is finished, a new word list is generated. All the words on the list will be translated to the expected language automatically.

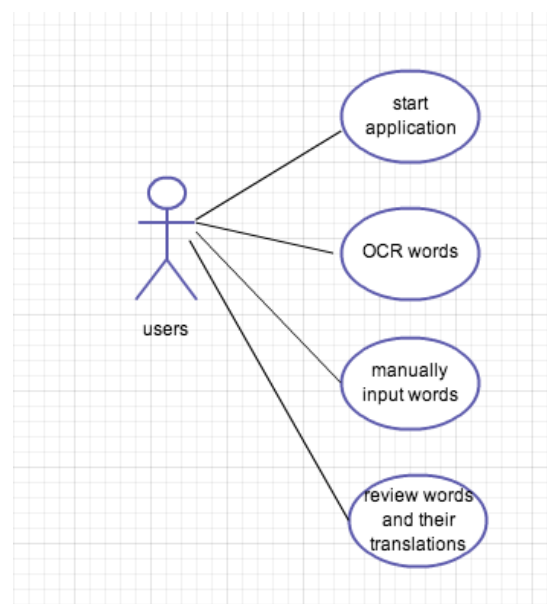


Figure 18. Use case diagram of the application



Thus, the Finnish words could be regarded as the hints information and the translation could be regarded as the answer.

### 3.2 The Application Workflow

This section discusses about the primary workflow of the application from the developing perspective. As I have mentioned, the OCR task utilized the Tesseract OCR engine. After words being captured by the camera, the string will be passed to the word list. In this stage, all the punctuation characters and numbers are filtered out. If multiple words are detected, the application will split them into a new row of a list. All the items of the list contain a check box for the purpose of choosing the words. When the word list has been confirmed, a new set of 'flash cards' will be generated. Every item of the list can be pulled down and closed up for the purpose of word memorizing. The translations of each word are settled in the pulled down containers.

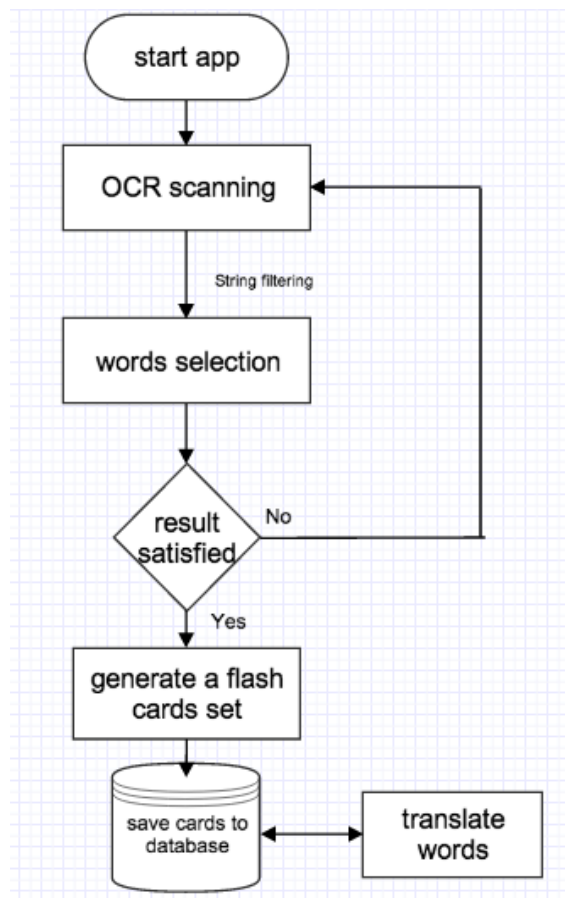


Figure 19. Flash Card Generation Process

Figure 19 describes the basic process with the flowchart. When the words scanning or inputting words manually has been done, the flash card data is stored into the database inside the application.

## 4 Implementation of the Application

### 4.1 Development Platform and Tools

Over the past few years, the blooming of smartphones has exerted a subtle influence on people's daily customs. As a fresh IT program student five years ago, the change aroused my profound curiosity of mobile software specialization option main curriculum of the degree. I spent most of my spare time on android development study. Thus I adopted the Android system as the platform for the final year project.

#### 4.1.1 Android

As the project started, there were four mobile operating systems performing noticeably in the market. As shown in Figure 21, they were Android, Blackberry OS, IOS and Microsoft Windows phone OS. Among them, Android was dominating the market with a share of over 80% [7]. One and a half million Android devices are activated every day in more than 190 countries [11]. Among the four mentioned operating systems above, Android is the only open-source system. It is developed on the Linux kernel, providing possibilities for the developers to perform lower level implementation [11].

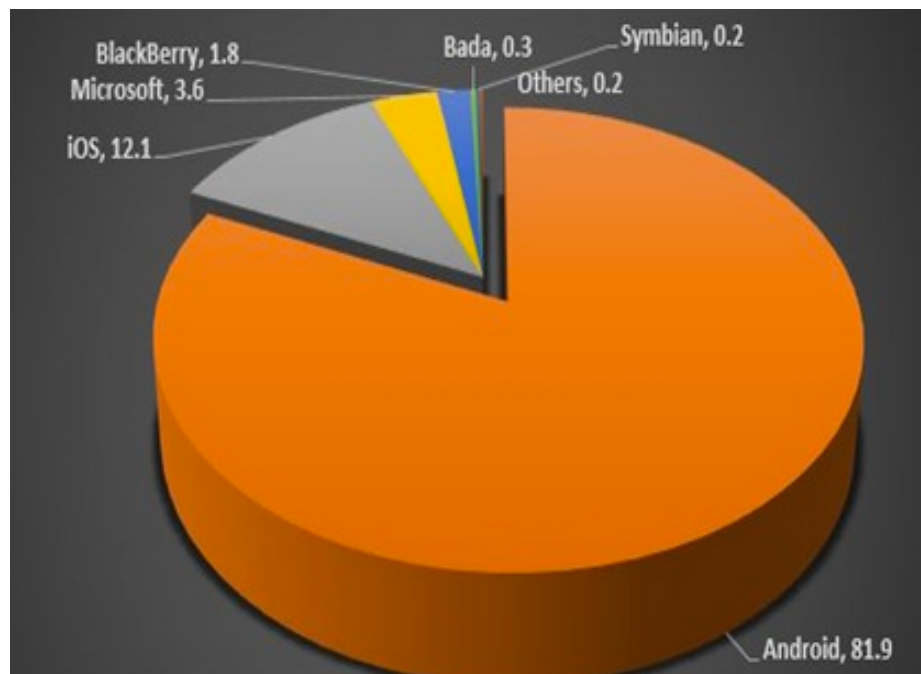


Figure 21. The smartphone OS Market Share copied from Android Software Development Kit (2014)[11]

Android OS consists of four major layers. There are four different layers in the Android stack (as shown in Figure 21):

- The Application Layer as the topmost layer interacts with system users.
- The Application Framework Layer provides the application developing access.
- The Libraries Layer includes a group of C/C++ libraries, primarily relating to the graph, audio processing and data saving.
- The Android Runtime Layer is located in the same layer as libraries' layer. It includes core Java libraries to enable programmers develop android application using Java programming language.
- The Linux Kernel Layer is the core of the system. It acts as a hardware abstraction at the bottom of the Android stack.

Such a stack structure design ensures the loose coupling between layers. When an update happens to a lower layer, the upper application layer will not need to be changed.



Figure 21. Android System Architecture Android Software Development Kit (2014)[11]

To develop an Android application, Android SDK (Software Development Kit) is the main tool needed for building, testing and debugging. It is composed of Android API libraries (SDK tool packages), ADT Plugin for Eclipse, system images, mobile the emulator and others.

Since Tesseract is implemented in C++, Android NDK (Native Development Kit) will be needed to encapsulate the native code as Java API, so that Tesseract functions can be called from the Android application.

#### 4.1.2 Tess-two Project

Tesseract officially provides a tool called “tesseract-android-tools” for Android developers. There is a fork of it called “tess-two” (<https://github.com/rmtheis/tess-two>) on github providing some additional functions. For the moment I launched the project, tess-two was more updated and often maintained. In this case, I adopted tess-two as the OCR engine to my project. The tess-two library requires Android 2.2 or higher version.

The tess-two project is runnable as a single project on Android.



Figure 22. Tess-two character recognition

As shown in Figure 22, the rectangle in the middle is the words choosing area. The recognized result is on the left upper corner. .

#### 4.1.3 Bing Translation APIs

At the final step of the application workflow, words need to be translated into the expected language automatically through the network. Both Google Translate and Microsoft Bing provide machine translation service. The only reason I chose Bing was that it was free into use the translation service. Microsoft does not provide the service in Java language. However, there is a Bing Translator Java wrapper project called 'microsoft-translator-java-api' held on Google Code. The Microsoft translation service demands an access token, so users need to register the translation service from Azure Datamarket (<http://datamarket.azure.com>).

## 4.2 Development Environment Configuration

### Building the native library

Since the tess-two project uses JNI (Java Native Interface) to implement the Tesseract C++ interface, before referencing tess-two, it will be necessary to use the NDK tool to build the project. Figure 23 shows how to build the tess-two project by using the NDK command in the shell.

```
cd <project-directory>/tess-two
export TESSERACT_PATH=${PWD}/external/tesseract-3.01
export LEPTONICA_PATH=${PWD}/external/leptonica-1.68
export LIBJPEG_PATH=${PWD}/external/libjpeg
ndk-build
android update project --path .
ant release
```

Listing 1. NDK building command

By building the project, NDK creates the native library in the lib/armabi and libs/armabi-v7a directories [12]. The library files are built as '.so' format, as shown in Figure 23.

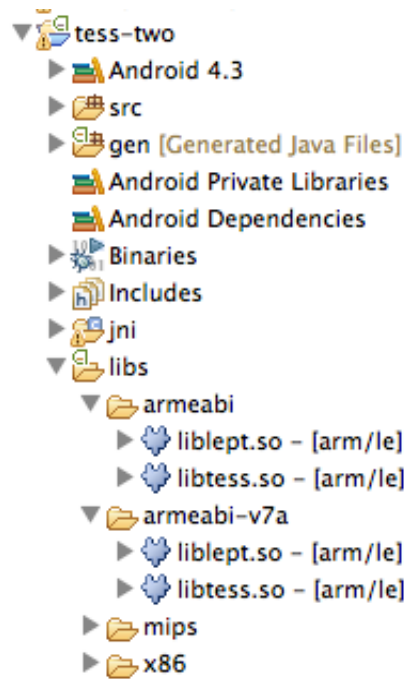


Figure 23. Building the native library

### Importing library project

During the Android development process, in the case there is more than one developer in the developing team, or one developer needs to take advantage of others work. The way of code reuse is necessary to be taken into consideration. If the common-used module is only simply copied to team members' projects, any small modification towards the common-used module will bring trouble to the team project combination and synchronization.

There are two ways of implementing code reuse in Android development. One way is to adopt one project as 'main' and export the rest as jar files. Then add those external jar files to the build path. The jar wrapping sets the code invisible. In order to use the Bing Translator API, download the 'microsoft-translator-java-api-0.6-mod.jar' file from the Google code holder and import the library file by right clicking the project, then choose properties -> Java build path -> Add External JARs ,as shown in Figure 25.

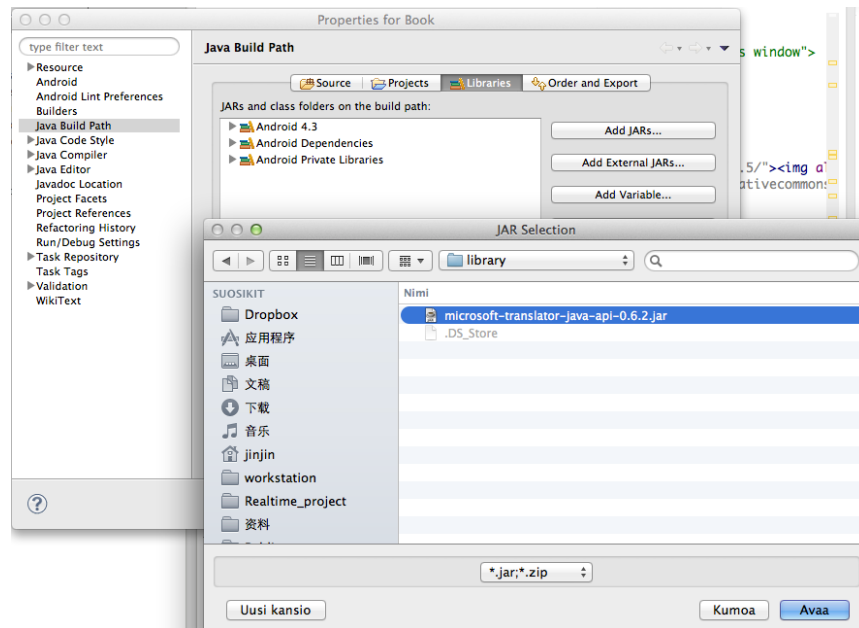


Figure 24. Add a JAR file as library

The other way is to set the referenced project as a library project. To import the tess-two project, I adopted the second way. To do that, firstly right click the tess-two project in the Eclipse project explorer and click properties. Choose the Android tag and check the 'is library' option, as shown in Figure 25.

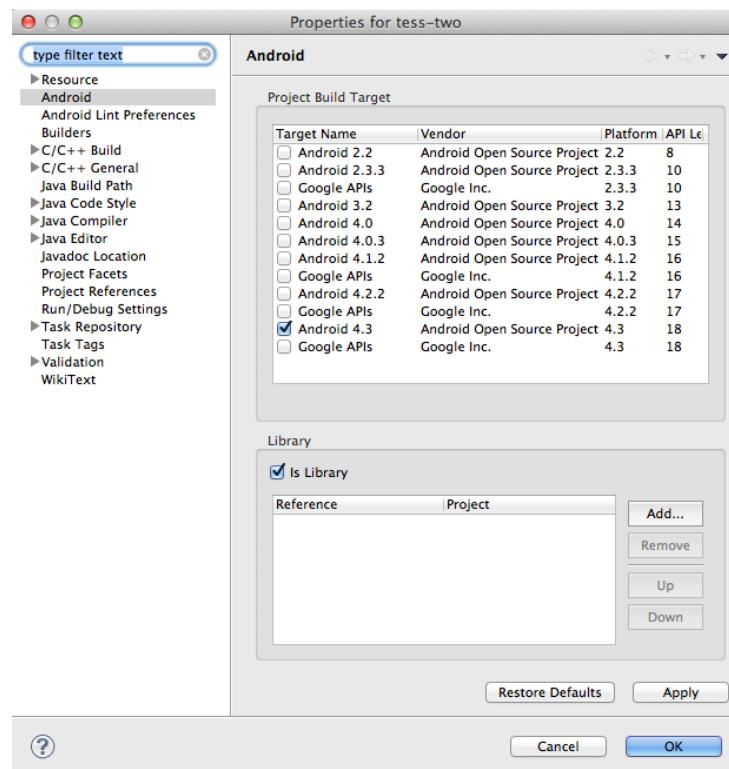


Figure 25. Library Project



To reference the tess-two project in the main project, right-click the flash card project and click properties, as shown in Figure 26.

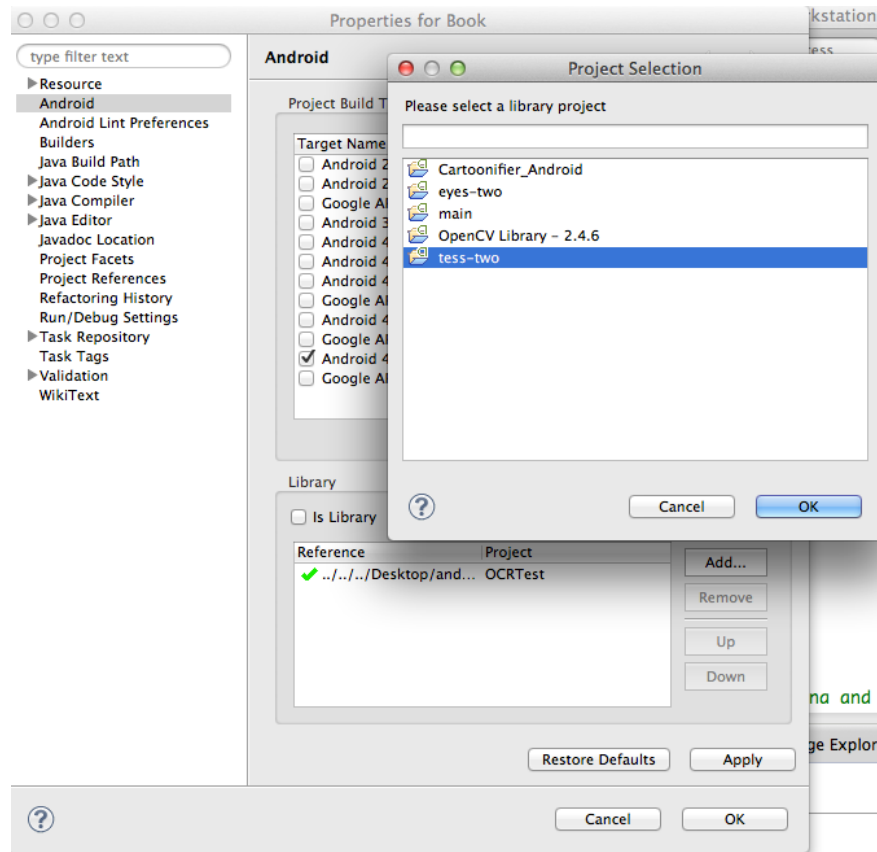


Figure 26. Import library project

Thus, the project environment has been setup.

#### 4.3 Class Diagram Of The Application

The code pattern was designed in a way that Models and Views were strictly isolated. Each package acts in a different role. The relationships are illustrated in the following class diagram Figure 27 and Figure 28. Figure 27 demonstrates the class implemented for the views at every use case stage.

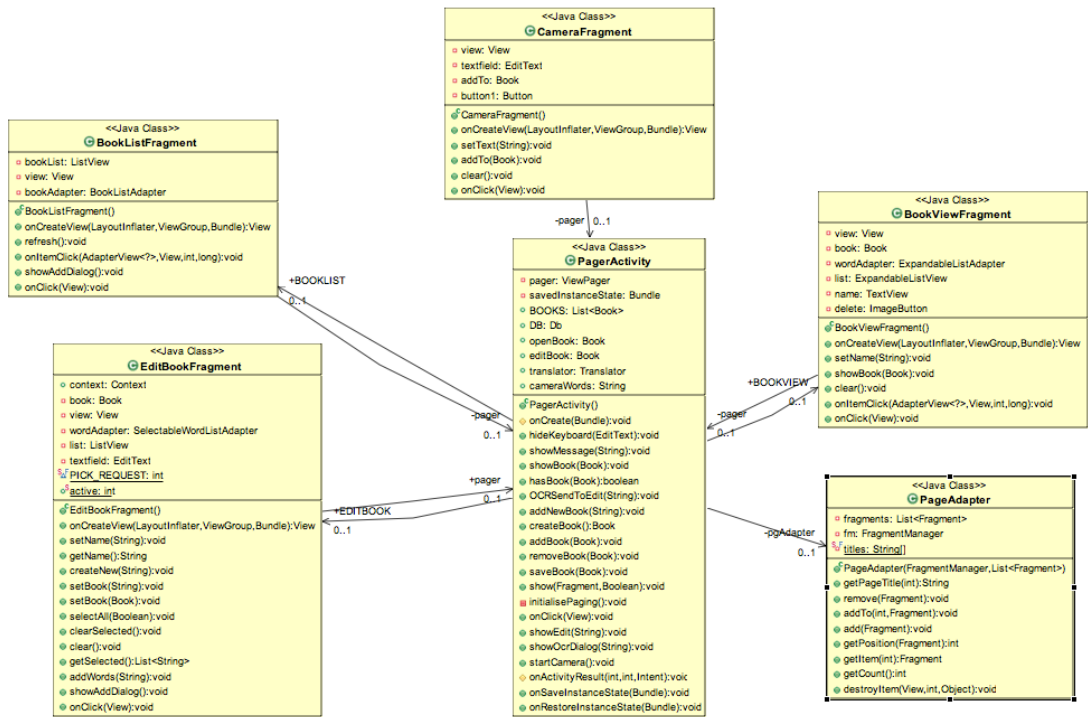


Figure 27. Application Views class diagram

When users enter the application, PageActivity with three fragments contained is created.

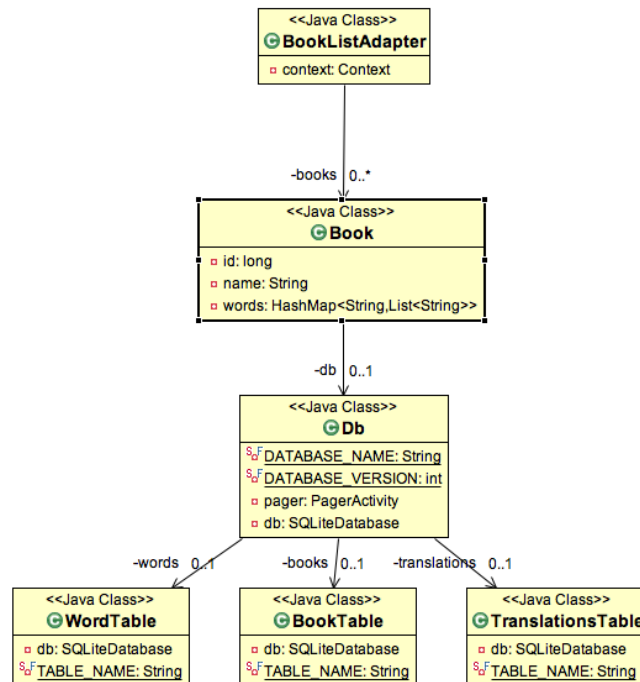


Figure 28. The data table class diagram

Figure 28 describes the data structure containing words, word lists and translations of words and their implementations.

#### 4.4 Implementation of the Components

##### 4.4.1 User Interface

The original intention of building an OCR Flash Card is to improve the user experience when people reciting words and interacting with the application. Other than a faster way of input, building a clear user interface with the smooth learning curve becomes another critical topic for the project. In the perspective of designing, nowadays various applications are built in a way that is over task-focused and simply with features piled up. This section discusses the user interface design of the flash card application.

In the user interface, there are four primary views in the application: camera view, edit view, browser view and book view in the flash card application. When the application starts, a user can add a set of words by means of the OCR (tapping the camera button) or by means of manual typing (tapping the plus button), as shown in Figure 29. Figure 30 shows how the application implements the OCR function in the camera view.



Figure 29. The initial interface

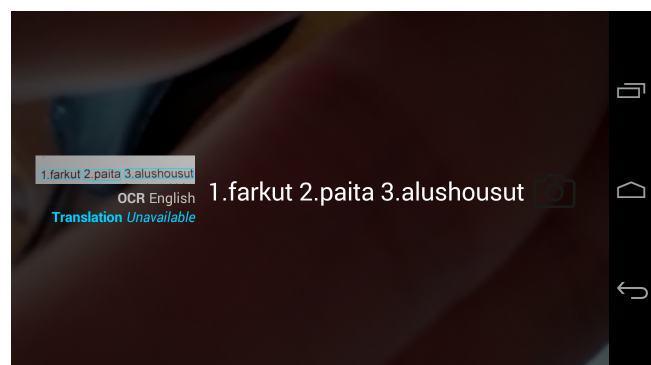


Figure 30. Camera Interface

In the camera view, one can OCR text and then see them in the edit view. If doesn't want to use OCR, one can manually input words in the edit-view. Whenever one use the application, one has to create a container (=list) for a set of words.

After the strings are captured, the strings will be segmented as words into the rows of a list. A user needs to select the expected words and type a name for the list of words. They can be browsed in the 'browse' view. By having named lists, the user can gather meaningful sets of words and utilize them later on. By tapping the list, the words show up in the book view. Their corresponding translations are shown in the unfolded items by tapping the words for the words reciting purpose, as shown in Figure 31. A flash card (metaphorically) is a card where on one side there is a word, and on the other side the translation of the word. In the book view, the user can hide the translation for learning the language.

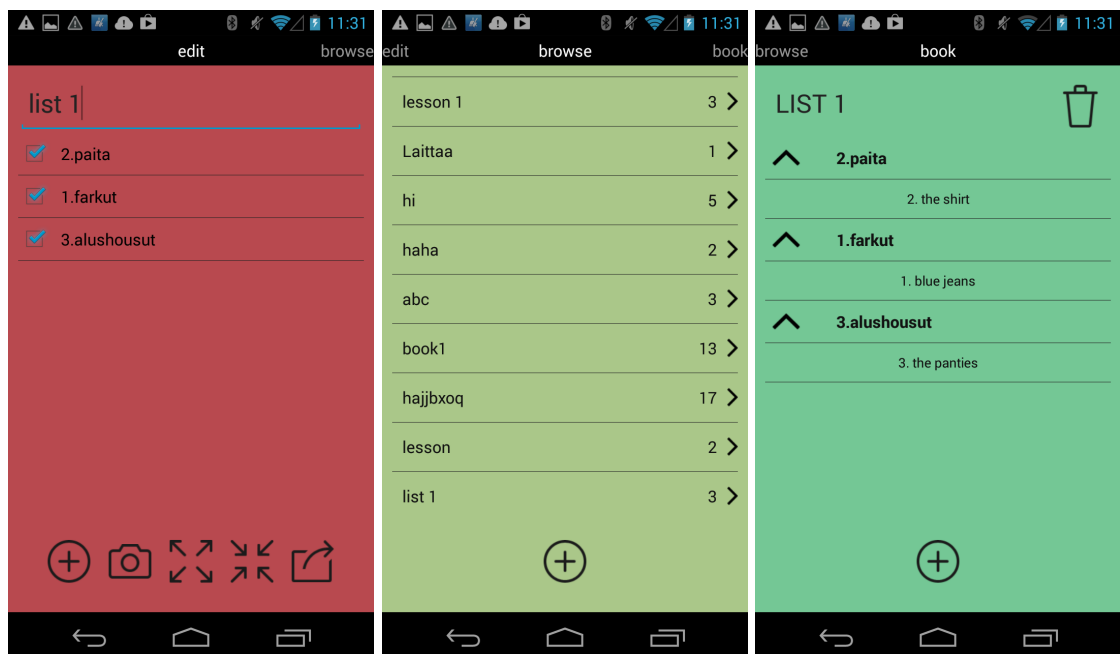


Figure 31. User interface of flash cards

To implement the user interface, I adopted the Android ViewPager, which was used in conjunction with three fragments. As the standard adapter, the FragmentPagerAdapter class was adopted for using fragments with the ViewPager [12]. Rather than building all the views with Activity, this way provided convenience to supplying and managing the life cycle of each page [12].

An Activity containing fragments has a direct impact on the fragments life cycle that it owns. Every callback to the Activity life cycle leads to a callback to the fragments' life cycle. For example, when the Activity receives the onPause() callback, every fragment contained in the Activity will also receive the onPause() callback. There are a few more callback functions in the fragment life cycle than in the Activity for the purpose of interacting with the Activity, generating and destroying the fragment UI. Figure 32 shows the relationship between the Activity life cycle and the Fragment life cycle.

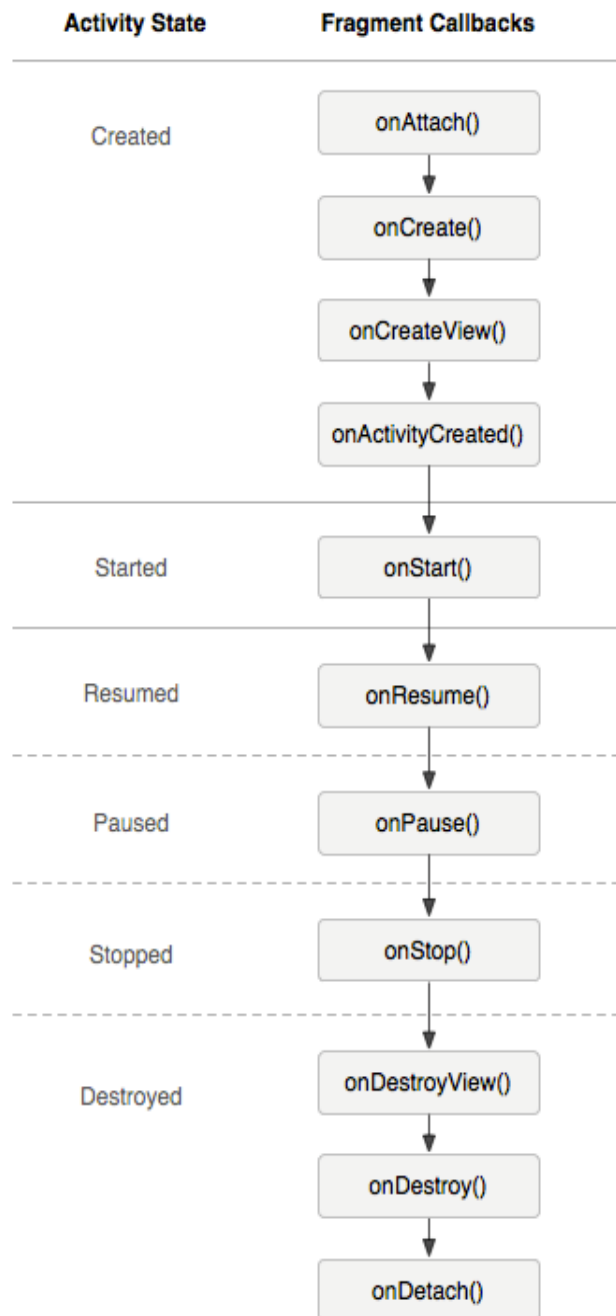


Figure 32. The Activity life cycle directly impacting on the Fragment life cycle

The fragment is not inherited from the View. It has its own life cycle, which makes it easy to manage when it is necessary to change its status. On the other hand, the Activity is independent of the Fragment life cycle. Using PageView with fragments improves the reusability and extensibility of the code.

#### 4.4.2 SQLite Database

The SQLite database is an open-source database embedded in every Android system. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements [13].

ID	Name

Table 1. The word list table

ID	Book_id	Word

Table 2. The word table

ID	Word	Translation

Table 3. The translation table

The SQLite database was adopted to store the word list, words and their translations in this project. The table 1,2 and 3 above illustrate the saved words' data structure and their dependency.

#### 4.4.3 Data Transfer

There are two main Activities built for the project. One was the PageView Activity for wrapping the words related view. The other was the camera view activity called Cap-

tureActivity, which was built in the Tess-two project. To transfer the captured string data from one activity to another, Intent was adopted.

An Intent is an object that provides runtime binding between separate components [14]. By calling putExtra() function, the result string is extended to the intent. Then I called the setResult() function to return to its caller Activity, which is the ViewPager with the intent transferred back. From the side of PagerView, the onActivityResult() function is called back when the camera view Activity, PagerView, launched exits. Intent is returned as the third parameter of the callback function. Finally I called getStringExtra() function to get the data, which is bound to the intent.

Once the OCR data is transferred to the ViewPager successfully, the workflow of the flash card will be initialized, as mentioned in section 4.4.1.

## 5 Project Evaluation

### 5.1 Application Testing and Result

The quality of OCR was primarily evaluated by manual testing. The test devices include LG G2 and Samsung Nexus 4 Android mobile. By applying the Tesseract OCR engine in the mobile camera character input, a few drawbacks emerged in different test conditions.

An ideal OCR object is that the word list is printed on a plate piece of paper with engine-familiar font that has been trained or stored. When opening a book, part of the paper close to the middle gutter line will become twisted. The twisted paper leads to distorted words, which lower the accuracy.

For the limitation of the mobile screen, the rectangle selection area may not wrap all the expected words in a whole line or a whole bar. The target paragraph may appear in some polygon other than the rectangle. Therefore, unrelated information may also be adopted. Other than the two most notable defects, recognition output is fast generated and accurate.

### 5.2 Further development and possible improvements

To improve the experience of word-selection, finding a way to replace the rectangle selection was the first idea that came to my mind. The most flexible way of choosing images is the Google Translate smearing. However, Google doesn't provide the OCR service APIs to third-party developers.

In this case, another way of recognizing words in OCR can be reasonably designed. First, build an activity similar to a drawing application with transparent background. Set the drawing track thick enough to wrap the words. The drawing track is regarded as the selection area instead of the rectangle. Cut the image of the selection from the background. Then deploy the rest of the work to the Tesseract engine. Android Canvas could be adopted to implement the drawing function.

Since the computer vision such as optical character recognizing task is still quite heavy for mobile processors nowadays, running OCR as a cloud-service on a remote server



could be another further development plan. In this case, Image would not be processed locally, users would need a network to upload the captured image and only the string from the image would be returned.

## 6 Conclusions

With decades of development and the emergence of new conferences, the OCR technique has made remarkable progress. Moreover, the computers speed is much faster than before. The way computers learning and getting information gravitates towards stimulating human beings. Optical character recognition is a technology that not only lowers the cost of manual labor with a faster computer input solution, but also provides users better user experiences.

In this project, OCR technology was applied to build a flash card Android application for memorizing new words. Users should be able to recognize and record the words without Internet access. The project was started in January 2014, and was finished at the beginning of March 2014. After two months of effort, the application was successfully built and it ran as expected.

The goal of the project was successfully accomplished, since the primary stages of the flash card workflow were implemented, and it bringing users better experience of selecting new words. On the current form of the application, considering it was developed the application within two months, I am satisfied with the project achievement.

## References

1. Tesseract OCR [online]. Google Developers; March 2013.  
URL: <https://code.google.com/p/tesseract-ocr/>. Accessed 10 April 2014
2. Cheriet M. Character Recognition System: A Guide for Students and Practitioners. New Jersey, United States of American: John Wiley & Sons, Inc; 2007.
3. Singh P, Budhiraja S. Feature extraction and Classification Techniques in O.C.R. System for Handwritten Gurmukhi Script-A Survey [online]; 2014.  
URL: <http://www.ijera.com/papers/Vol%201%20issue%204/BQ01417361739.pdf>  
Accessed 10 April 2014
4. Verma R, Ali J. A-Survey of Feature Extraction and Classification Techniques in OCR Systems [online]; 2014.  
URL: <http://www.ijcait.com/IJCAIT/index.php/www-ijcs/article/download/140/86>  
Accessed 10 April 2014
5. Cabebe Jamar. Google Translate for Android adds OCR [online]; 2014.  
URL: <http://www.cnet.com/news/google-translate-for-android-adds-ocr/> Accessed 10 April 2014
6. Roberts D. Automated License Plate Recognition Systems: Policy and Operational guidance for Law Enforcement [online]; 2012.  
URL: <https://www.ncjrs.gov/pdffiles1/nij/grants/239604.pdf>  
Accessed 10 April 2014
7. Access Ltd. OCR640 OCR640 full-page multi-illumination desktop readers for ePassports and eID Cards [online]; 2014.  
URL: <http://www.access-is.com/ocr640-desktop-full-page-passport-id-reader.php>  
Accessed 10 April 2014
8. Smith R. An Overview of the Tesseract OCR Engine [online]  
URL: <http://static.googleusercontent.com/media/research.google.com/zh-CN/pubs/archive/33418.pdf> Accessed 10 April 2014

9. Smith R. Tesseract OCR Engine [online]; 2007.  
URL: <http://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf> Accessed 10 April 2014
10. Mohul G. Anroid Has 82% Market Share [online]; November 2013.  
URL: [http://trak.in/tags/business/2013/11/15/smartphone-market-share-3q2013-lenovo-3rdandroid/?utm\\_source=feedburner&utm\\_medium=email&utm\\_campaign=Feed%3A+trak.in+%28India+Business+Blog+%21%29](http://trak.in/tags/business/2013/11/15/smartphone-market-share-3q2013-lenovo-3rdandroid/?utm_source=feedburner&utm_medium=email&utm_campaign=Feed%3A+trak.in+%28India+Business+Blog+%21%29) Accessed 10 April 2014
11. GOOGLE INC. (Hrsg.): Android Software Development Kit(SDK) [online]; 2014.  
URL: <http://developer.android.com/sdk/index.html>. Accessed 10 April 2014
12. GOOGLE INC. (Hrsg.): ViewPager Class Overview [online]; 2014.  
URL:  
<http://developer.android.com/reference/android/support/v4/view/ViewPager.html>  
Accessed 10 April 2014
13. AsyncTask [online]. Android Developers; August 2012.  
URL: <http://developer.android.com/reference/android/os/AsyncTask.html>.  
Accessed 1 May 2013.
14. AsyncTask [online]. Android Developers; August 2012.  
URL:<http://developer.android.com/training/basics/firstapp/starting-activity.html> Accessed 1 May 2014



