جامعة اليرموك الخاصة

كلية هندسة المعلوماتية والاتصالات

قسم هندسة البرمجيات

Word Finder

مشروع ما قبل التخرج في معالجة الصور

للطالب: وائل حمادة

بإشراف المهندس: إسماعيل الجبّة

الفصل الدراسي:

2013 –2

ACKNOWLEDGEMENTS

This project is made possible through the help and support from everyone, including: parents, teachers, family, friends, and in essence, all sentient beings. Especially, please allow me to dedicate my acknowledgment of gratitude toward the following significant advisors and contributors:

First and foremost, I would like to thank Eng. Ismail Ibrahim Aljubbah for his most support and encouragement. He kindly followed me through this project and offered invaluable detailed advices on many critical aspects.

Second, I would like to thank Dr. Ahmad Webbi and Dr. Nabil Hamed for the kind courses in neural networks and computer vision.

Finally, I sincerely thank to my parents, family, and friends, who provide the advice and financial support. The application of this project would not be possible without all of them.

ABSTRACT:

For this project, we propose and implement a reading tool on Andriod platform that can be used to identify keywords on paper-based media. We breakdown the image processing after receiving an image of the media into three steps. In the first step, we pre-process the image using binarization, de- skew and de-noising. The second step involves using OCR (in particular, Tesseract OCR engine) to recognize the text and find the keywords. Finally, we highlight the keywords by rectangle it with bounding boxes.

نقوم من خلال هذا المشروع بتطوير أداة تعمل على نظام الهاتف المحمول "أندرويد" ليتيح للمستخدم البحث عن كلمة ما في نص مكتوب ومطبوع مثل الصحف، مجلات او كتب مما يساعد المستخدم بعملية إيجاد الكلمة المطلوبة بسرعة. تقوم العملية على عدة نقاط أساسية هي: تحضير أولي للصورة (النص المطبوع) يما يتضمن تحويل الصورة الى الأبيض والأسود وإزالة الشوائب والقيام بتدوير الصورة للشكل الصحيح ثم القيام بتقطيع الصورة وفحص الكلمة لمطابقتها مع الكلمة المطلوبة وتحديدها للمستخدم بإطار.

Table of Contents

Introduction:

Have you ever read a long paper-based article and find yourself unable to locate keywords? With the advancement of digital media sometimes we take basic word search for granted. But the world is still full of media printed on paper and it would make our lives much simpler if we can automate this basic reading tool for good old fashioned books. We propose a mobile application that will be able to find a word that the user specified through phone's viewfinder. The phone highlights keywords detected in the viewfinder, saving the user a lot of time manually searching for word. For example, we want to search for "YPU" in this paper-based declaration of independence. We only need to use our smartphone to scan over the paper. It will be immediately circled in green bounding boxes. As shown in Figure 1

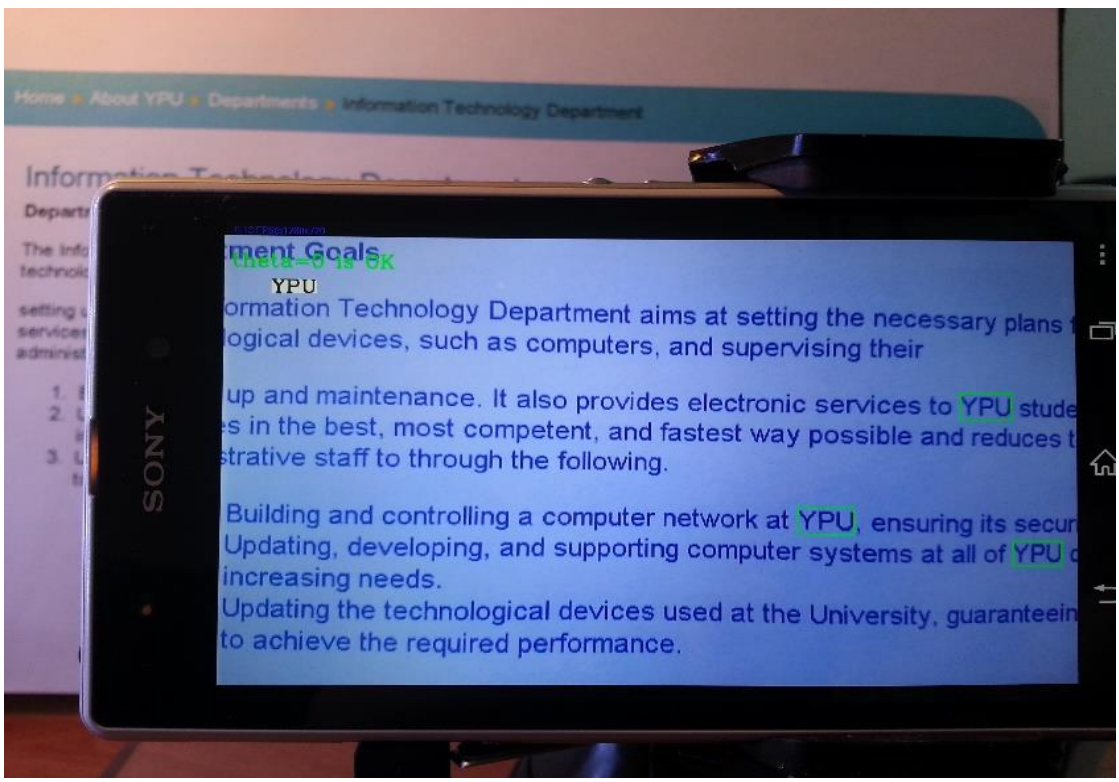


Figure 1(Word Finder)

Chapter 1:
State of Art

1-1 The History of optical character recognition (OCR)

Methodically, character recognition is a subset of the pattern recognition area. However, it was character recognition that gave the incentives for making pattern recognition and image analysis matured fields of science.

1-1-1 The very first attempts

To replicate the human functions by machines, making the machine able to perform tasks like reading, is an ancient dream. The origins of character recognition can actually be found back in 1870. This was the year that C.R.Carey of Boston Massachusetts invented the retina scanner which was an image transmission system using a mosaic of photocells. Two decades later the Polish P. Nipkow invented the sequential scanner which was a major breakthrough both for modern television and reading machines. During the first decades of the 19'th century several attempts were made to develop devices to aid the blind through experiments with OCR. However, the modern version of OCR did not appear until the middle of the 1940's with the development of the digital computer. The motivation for development from then on, was the possible applications within the business world.

1-1-2 The start of OCR

By 1950 the technological revolution was moving forward at a high speed, and electronic data processing was becoming an important field. Data entry was performed through punched cards and a cost-effective way of handling the increasing amount of data was needed. At the same time the technology for machine reading was becoming

sufficiently mature for application, and by the middle of the 1950's OCR machines became commercially available. The first true OCR reading machine was installed at Reader's Digest in 1954. This equipment was used to convert typewritten sales reports into punched cards for input to the computer.

1-1-3 First generation OCR.

The commercial OCR systems appearing in the period from 1960 to 1965 may be called the first generation of OCR. This generation of OCR machines were mainly characterized by the constrained letter shapes read. The symbols were specially designed for machine reading, and the first ones did not even look very natural. With time multi-font machines started to appear, which could read up to ten different fonts. The number of fonts were limited by the pattern recognition method applied, template matching, which compares the character image with a library of prototype images for each character of each font.

1-1-4 Second generation OCR.

The reading machines of the second generation appeared in the middle of the 1960's and early 1970's. These systems were able to recognize regular machine printed characters and also had hand-printed character recognition capabilities. When hand-printed characters were considered, the character set was constrained to numerals and a few letters and symbols. The first and famous system of this kind was the IBM 1287, which was exhibited at the World Fair in New York in 1965. Also, in this period Toshiba developed the first automatic letter sorting machine for postal code numbers and Hitachi made the first OCR machine for high performance and low cost. In this period significant work was

done in the area of standardization. In 1966, a thorough study of OCR requirements was completed and an American standard OCR character set was defined; OCR-A. This font was highly stylized and designed to facilitate optical recognition, although still readable to humans. A European font was also designed, OCR-B, which had more natural fonts than the American standard. Some attempts were made to merge the two fonts into one standard, but instead machines being able to read both standards appeared.

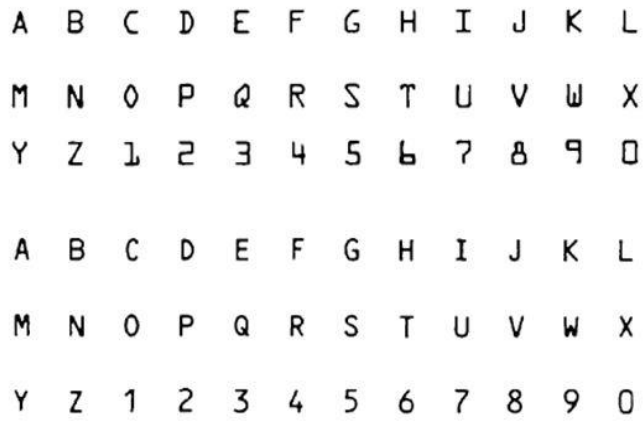


Figure 2 OCR-A (top), OCR-B (bottom).

1-1-5 Third generation OCR

For the third generation of OCR systems, appearing in the middle of the 1970's, the challenge was documents of poor quality and large printed and hand-written character sets. Low cost and high performance were also important objectives, which were helped by the dramatic advances in hardware technology. Although more sophisticated OCR-machines started to appear at the market simple OCR devices were still very useful. In the period before the personal computers and laser printers started to dominate the area of text production, typing was a special niche for OCR. The uniform print spacing and small number of fonts made simply designed OCR devices very useful. Rough drafts

could be created on ordinary typewriters and fed into the computer through an OCR device for final editing. In this way word processors, which were an expensive resource at this time, could support several people and the costs for equipment could be cut.

1-1-6 OCR today

Although, OCR machines became commercially available already in the 1950's, only a few thousand systems had been sold world wide up to 1986. The main reason for this was the cost of the systems. However, as hardware was getting cheaper, and OCR systems started to become available as software packages, the sale increased considerably. Today a few thousand is the number of systems sold every week, and the cost of an omnifont OCR has dropped with a factor of ten every other year for the last 6 years.

1870	The very first attempts
1940	The modern version of OCR.
1950	The first OCR machines appear
1960 - 1965	First generation OCR
1965 - 1975	Second generation OCR
1975 - 1985	Third generation OCR
1986 ->	OCR to the people

Table 1 A short OCR chronology.

1-2 Feature detection and matching

Feature detection and matching are an essential component of many computer vision applications. Consider the two pairs of images shown in Figure 3. For the first pair, we may wish to align the two images so that they can be seamlessly stitched into a composite mosaic. For the second pair, we may wish to establish a dense set of correspondences so that a 3D model can be constructed or an in-between view could be generated.

Figure 3 (Two pairs of images to be matched)

1-2-1 Feature detectors

Look image pair shown in Figure 4 and at the three sample patches to see how well they might be matched or tracked. As you may notice, texture-less patches are nearly impossible to localize. Patches with large contrast changes (gradients) are easier to localize, although straight-line segments at a single orientation suffer from the aperture problem.

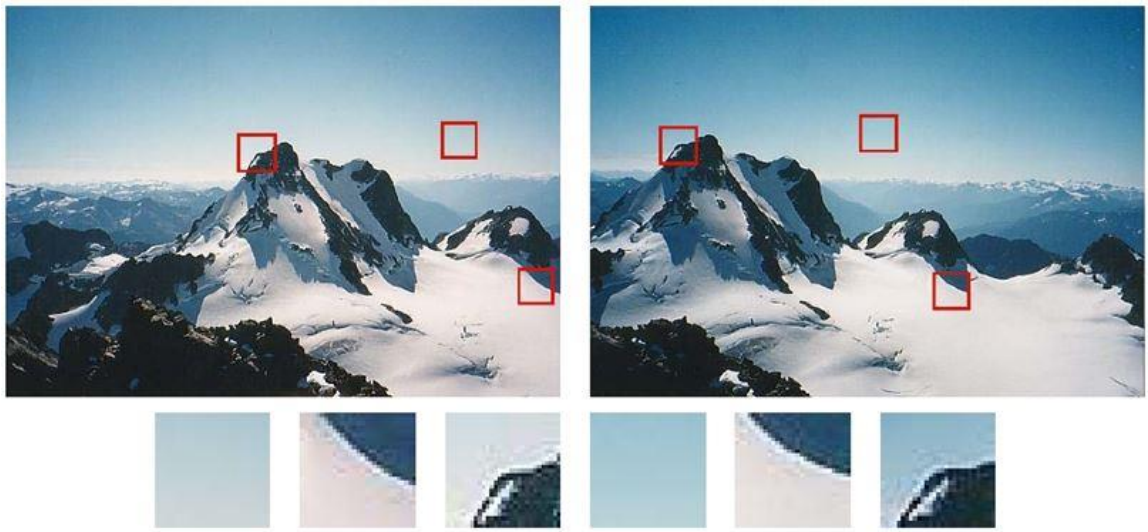


Figure 4 (image Paris with extracted patches below. notice how some patches can be localized or matched with higher accuracy than others)

1-2-1-1 List of some most common Feature detectors

- Hessian/ Harris corner detector
- Laplacian of Gaussian (LOG) detector
- Difference of Gaussian (DOG) detector
- Hessian/ Harris Laplacian detector
- Hessian/ Harris Affine detector
- Maximally Stable Extremal Regions (MSER)
- Many others…

1-2-2 Feature descriptors

After detecting the features (key-points), we must match them, i.e., determine which features come from corresponding locations in different images. In some situations, e.g., for video sequences or for stereo pairs that have been rectified, the local motion around each feature point may be mostly translational. In this case, the simple error metrics such as the sum of squared differences or normalized cross-correlation can be used to directly compare the intensities in small patches around each feature point. Because feature points may not be exactly located, a more accurate matching score can be computed by performing incremental motion refinement, but this can be time consuming and can sometimes even decrease performance.

In most cases, however, the local appearance of features will change in orientation, scale, and even affine frame between images. Extracting a local scale, orientation, and/or affine frame estimate and then using this to resample the patch before forming the feature descriptor is thus usually preferable.

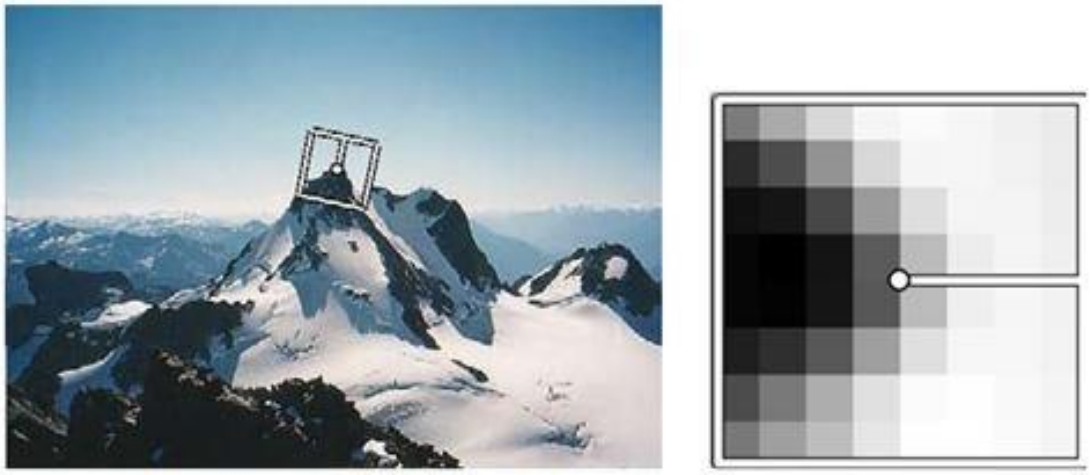


Figure 5 (MOPS descriptors are formed using an 8x8 sampling of bias/gain normalized intensity values, with a sample spacing of 5 pixels relative to the detection scale . This low frequency sampling gives the features some robustness to interest point location error, and is achieved by sampling at a higher pyramid level than the detection scale)

1-2-2-1 List of some most common Feature descriptors

- Scale Invariant Feature Transformation (SIFT)
- Speed-Up Robust Feature (SURF)
- Histogram of Oriented Gradient (HOG)
- Gradient Location Orientation Histogram (GLOH)
- PCA-SIFT
- Pyramidal HOG (PHOG)
- Pyramidal Histogram of visual Word (PHOW)
- Others … (shape context, Steerable filters, Spain images).

1-2-3 Feature Extraction for OCR

One of the main areas of application of any feature extraction algorithm is optical character recognition (OCR) and automatic document reading. Up to now, mathematical morphology is not often applied in these areas, although it offers interesting tools for solving some of problems.

From the large variety of morphological operators, directional operators seem to be useful for that purpose. They can operate in particular directions and find directional features of objects. Features are understood here as a set of information which allows us to identify characters and to classify it to one of the pre-defined classes.

For example of using directional morphology operators to extract features is to only consider two directions (horizontal – vertical). We use directional dilations in several steps of feature extraction. Firstly we use it for defining the character area – the smallest square including the character. This can be easily and fast obtained by taking intersection of horizontal and vertical dilation (figure 6a-6d) Similar procedure can be applied to extraction of characters from the whole line of text, as well as extraction of lines from whole page. But the main problem behind not using this approach for my project that it's very intensive operation which would not be done on a mobile phone

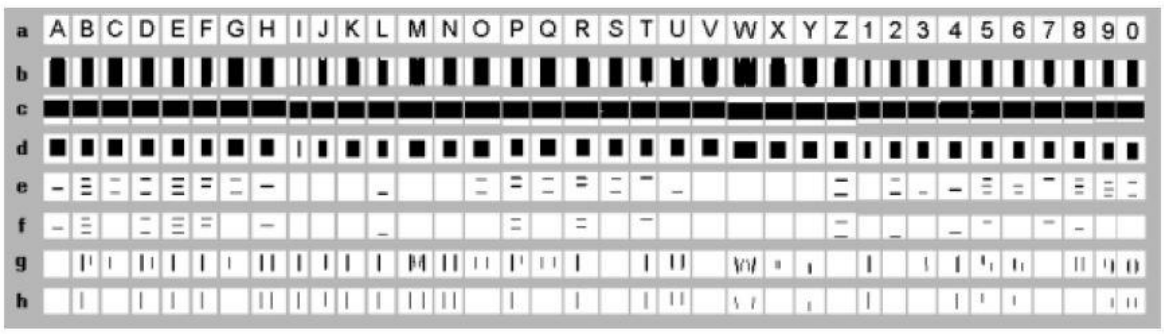


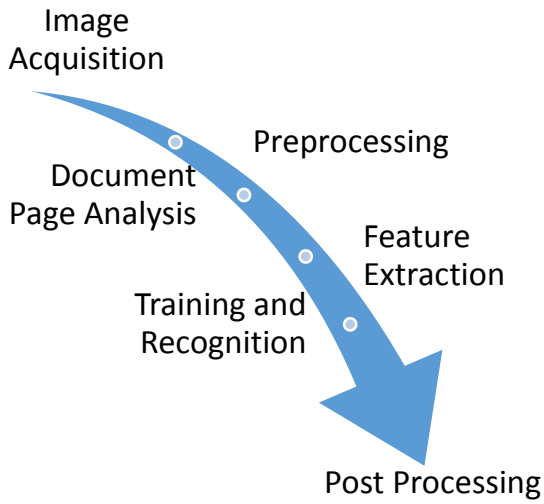
Figure 6 (Set of 36 characters of Arial font(a), after vertical dilation (b), after horizontal dilation (c), character area – intersection of vertical and horizontal dilation (d), extraction of horizontal lines – horizontal opening (e), filtered horizontal line (f), extraction of vertical lines – vertical opening (g), filtered horizontal lines (h).

Chapter 2: Algorithms

2-1 OCR using ANN Algorithms:

Optical Character Recognition (OCR) Using Artificial Neural Network (ANN) is basically in the field of research. To gain better knowledge, techniques and solutions regarding the procedures that we want to follow, we studied the various research papers on existing OCR systems. All these studies helped us with clarifying our target goals. The basic steps involved in Optical Character Recognition are:

1. Image Acquisition
2. Preprocessing
3. Document Page Analysis
4. Feature Extraction
5. Training and Recognition
6. Post Processing



2-2 Different Areas of Character Recognition:

Optical Character Recognition deals with the problem of recognizing optically processed characters. Optical recognition is performed off-line after the writing or printing has been completed, as opposed to on-line recognition where the computer recognizes the characters as they are drawn. Both hand printed and printed characters may be recognized, but the performance is directly dependent upon the quality of the input documents. The areas of character recognition are shown in Figure 7.

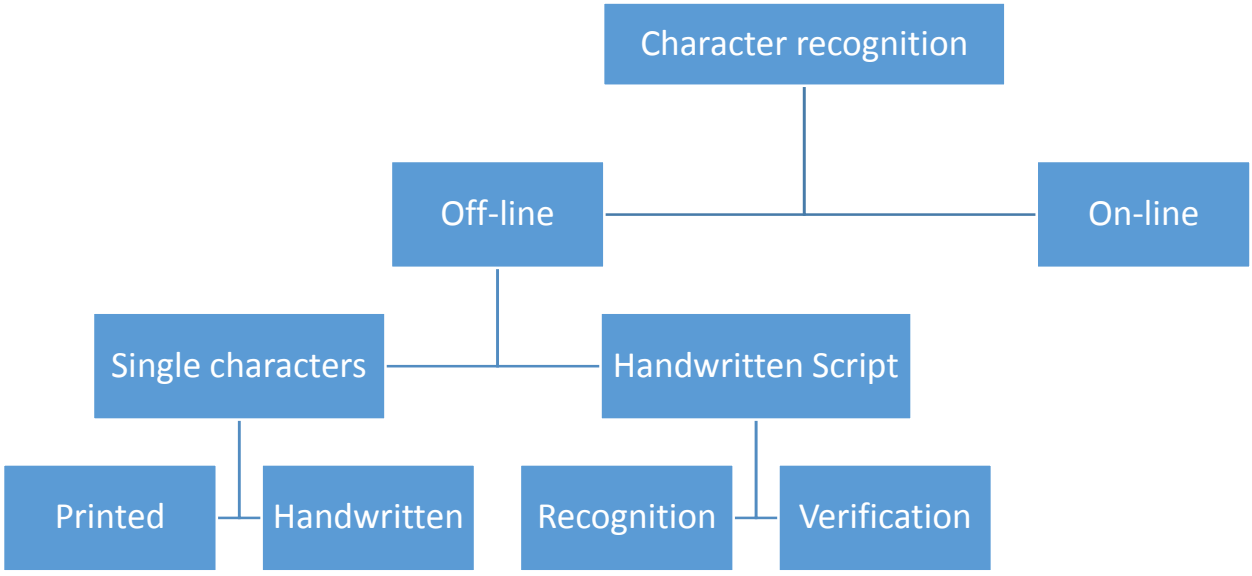


Figure 7 (The areas of character recognition)

The more constrained the input is the better will the performance of the OCR system be. However, when it comes to totally unconstrained handwriting, OCR machines are still a long way from reading as well as humans. However, the computer reads fast and technical advances are continually bringing the technology closer to its ideal.

2-3 Algorithm Used For Optical Character Recognition:

One of the most typical problems to which a neural network is applied is that of optical character recognition. Recognizing characters is a problem that at first seems extremely simple, but it is extremely difficult in practice to program a computer to do it. And Yet, automated character recognition is of vital importance in many industries such as banking. Moreover, shipping. The U.S. post office uses an automatic scanning system to recognize the digits in ZIP codes. We may have used scanning software that can take an image of a

printed page and generate an ASCII document from it. These devices work by simulating a type of neural network known as a back propagation network.

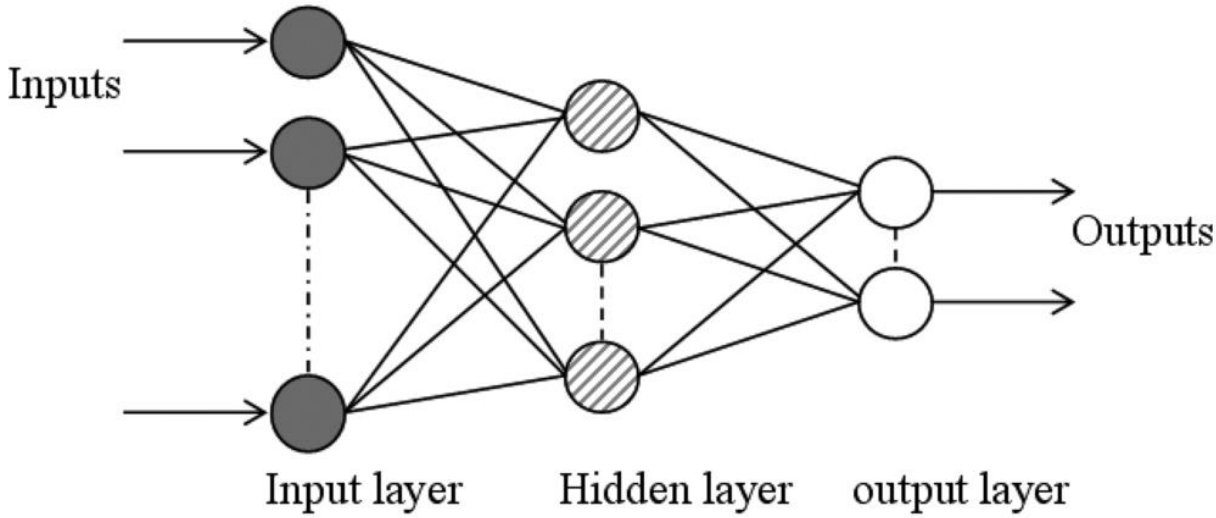


Figure 8 (Back Propagation network)

2-3-1 Back propagation Algorithm

The Back propagation algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set). Back propagation consists of the repeated application of the following two passes:

2-3-1-1Forward pass:

In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.

2-3-1-2Backward pass:

In this step, the network error is used for updating the weights (credit assignment problem). This process is more complex than the LMS algorithm for Adaline (Adaptive Linear Neuron or later Adaptive Linear Element), because hidden nodes are linked to the error not directly but by means of the nodes of the next layer. Therefore,

starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.

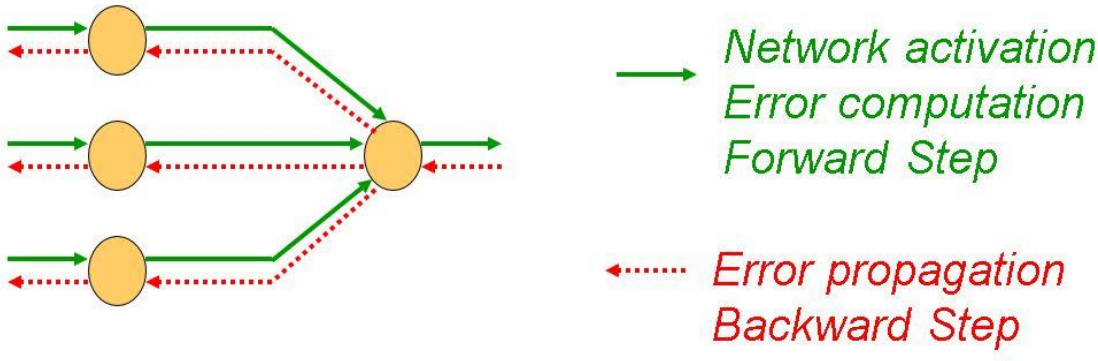


Figure 9 (Back Propagation Training phases)

2-4 Training using SunnyPage:

Figure 11 (SunnyPage Logo)

Because it's difficult to implement the Back Propagation algorithm, we found a program called SunnyPage which uses a mixture of training algorithms to provide the most accurate calculation of weights for each neuron in the neural network.

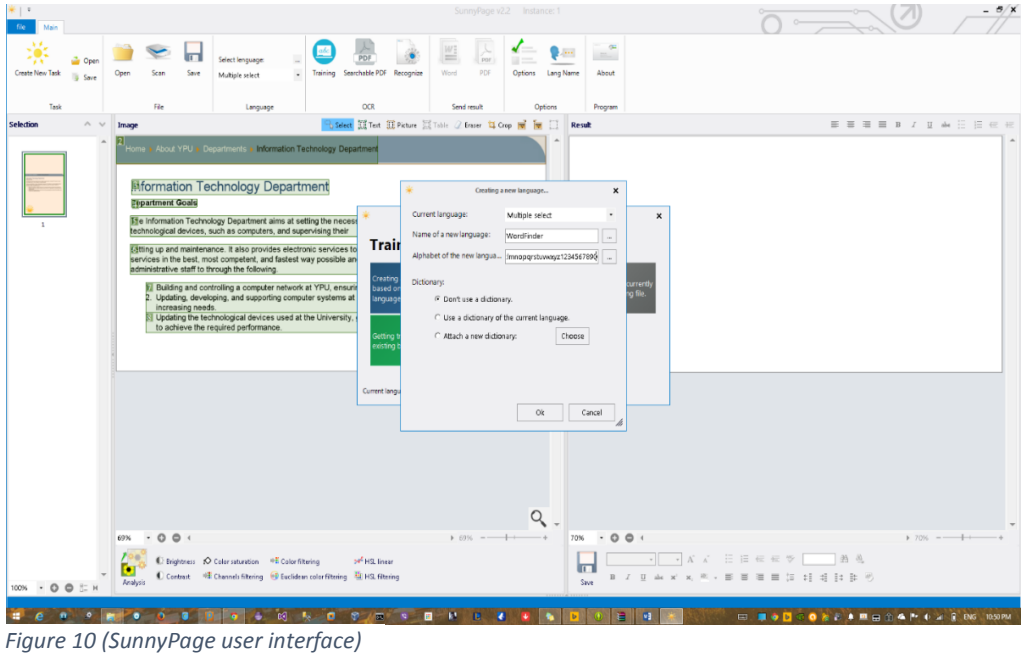


Figure 10 (SunnyPage user interface)

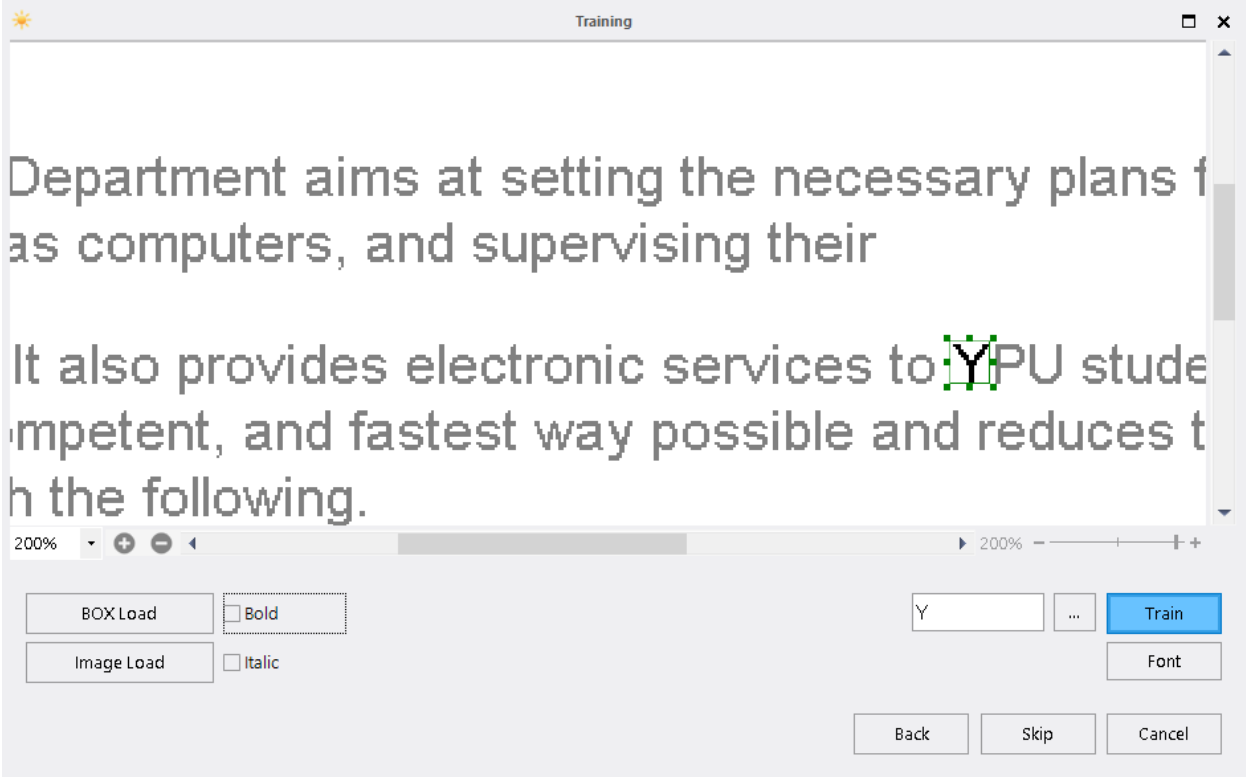
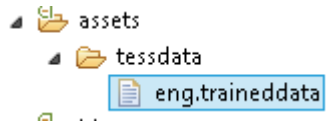


Figure 12 (SunnyPage Training GUI)

After the training, we result having a trained data for the language we've created and now we can use it in the recognition step within the implementation



Chapter 3:

Implementation

3-1 Android:

3-1-1 Brief Overview

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

Figure 13 (Android Logo)

Google wanted Android to be open and free; hence, most of the Android code was released under the open source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors. This has been especially true for companies affected by the phenomenon of Apple's iPhone, a hugely successful product that revolutionized the smartphone industry. Such companies include Motorola and Sony Ericsson, which for many years have been developing their own mobile operating systems. When the iPhone was launched, many of these manufacturers had to scramble to find new ways of revitalizing their products. These manufacturers see Android as a solution — they will continue to design their own hardware and use Android as the operating system that powers it.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain. Device manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

3-1-2 Required device features:

A. a 5+ Mega Pixel camera (Autofocus is optional for better recognition)

B. LED Flash (optional)

3-1-3 Android Camera API:

Classes	8 Callback interfaces
Camera	Camera.AutoFocusCallback
Camera.CameraInfo	Camera.ErrorCallback
Camera.Parameters	Camera.FaceDetectionListener
Camera.Size	Camera.PictureCallback
Camera.Face	Camera.PreviewCallback
Camera.Area	Camera.ShutterCallback
	Camera.AutoFocusMoveCallback
	Camera.OnZoomChangeListener

3-1-3-1Camera Class:

- Open & Release

- Access to the Camera Controls

- Preview:

    - Direct Live Preview to the Display or a texture.

    - Get Preview Frame in a Callback.

- Capture:

    - Callbacks: Shutter, JPEG, RAW, “PostView”

- Lock & Unlock

- Actions: StartAutoFocus, StartSmoothZoom & StartFaceDetection.

- Camera.Parameter: This class have some parameters to control the Camera

    - Auto White Balance

    - Sence Modes

- Focus Modes

- Preview Sizes

- Picture Sizes

- Thumbnail Sizes

- Preview Format

- Picture Format

3-2 Android Software Development Kit (SDK):

3-2-1 Definition:

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. The Android SDK includes sample projects with source code, development tools, an emulator, and required libraries to build Android applications. Android Applications are written using the Java programming language and run on Dalvik, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.The officially supported integrated development environment (IDE) is Eclipse (which I used) using the Android Development Tools (ADT) Plugin, though IntelliJ IDEA IDE fully supports Android development out of the box. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Android applications are packaged in .apk format and stored under /data/app folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called Dalvik executables), resource files, etc.

3-2-2 Advantages of Android App Development:

3-2-2-1 Low Investment and High ROI

Android comparatively has a low barrier to entry. Android provides freely its Software Development Kit (SDK) to the developer community which minimizes the development and licensing costs.

3-2-2-2 Open Source

Get the open source advantage from licensing, royalty-free, and the best technology framework offered by the Android community. The architecture of the Android SDK is open-source which means you can actually interact with the community for the upcoming expansions of android mobile application development. This is what makes the Android mobile application development platform very attractive for handset manufacturers and wireless operators, which results in a faster development of Android based phones, and better opportunities for developers to earn more.

3-2-2-3Easy to Integrate

The entire platform is ready for customization. You can integrate and tweak the mobile app according to your business need. Android is a good mobile platform between the application and processes architecture. Most of the platforms allow background processes helping you to integrate the apps.

3-2-2-4Easy Adoption

Android apps are scripted in Java language with the help of a rich set of libraries.

3-3 Android NDK:

3-3-1 Definition:

Android is put together of about equal part Java and C. So, no wonder that we need an easy way to bridge between these two totally different worlds. Java offers Java Native Interface (JNI) as a framework connecting the world of Java to the native code. Android goes a step further by packaging other useful tools and libraries into a Native Development Kit, or NDK. NDK makes developing C/C++ code that works with an Android app much simpler than if one was to do it by hand.

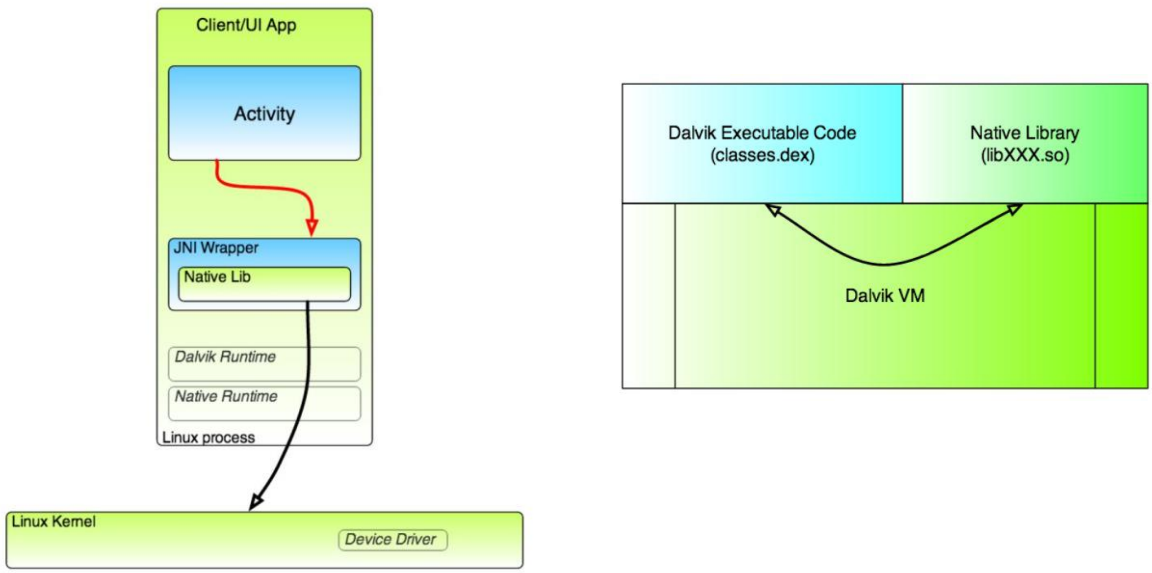


Figure 14 (Using NDK to connect Activity to native code)

3-2-2 NDK facts:

- NDK is tool-chain : Cross-compiler, Linker, What you need to build for ARM, x86, MIPS, etc.

- NDK Provides a way to bundle lib.so into APK: The native library needs to be loadable in a secure way.

- NDK Standardizes various native platforms: it provides headers for libc, libm, libz, liblog, libjnigraphics, OpenGL/OpenSL ES, JNI headers, minimal C++ support headers, and Android native app APIs.

3-2-3 Advantages of Using NDK:

-Performance: sometime native code still runs faster.

-Legacy support: you may have that C/C++ code you would like to use in your app.

-Access to low-level libraries: in a rare case when there is no java API to do something.

-Cross-platform development: C is the new portable language.

3-2-4 Disadvantages of Using NDK:

-Adding JNI to an app will make it more complex, which makes it harder to debug.

-Java often much richer APIs, Memory protection, OOP, Productivity.

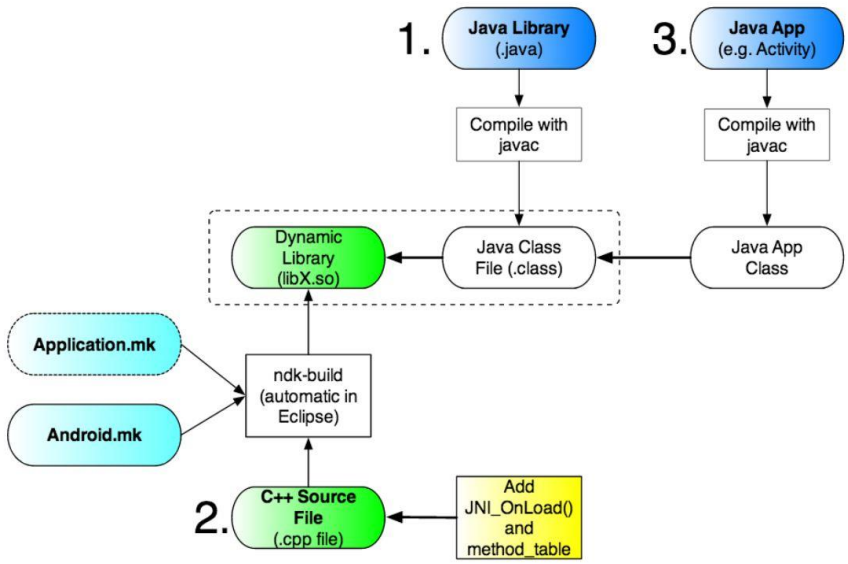


Figure 15 (NDK Process Using C++)

3-4 Open Computer Vision (OpenCV):

3-4-1 Definition

Open Source Computer Vision (OpenCV) Library is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate

and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

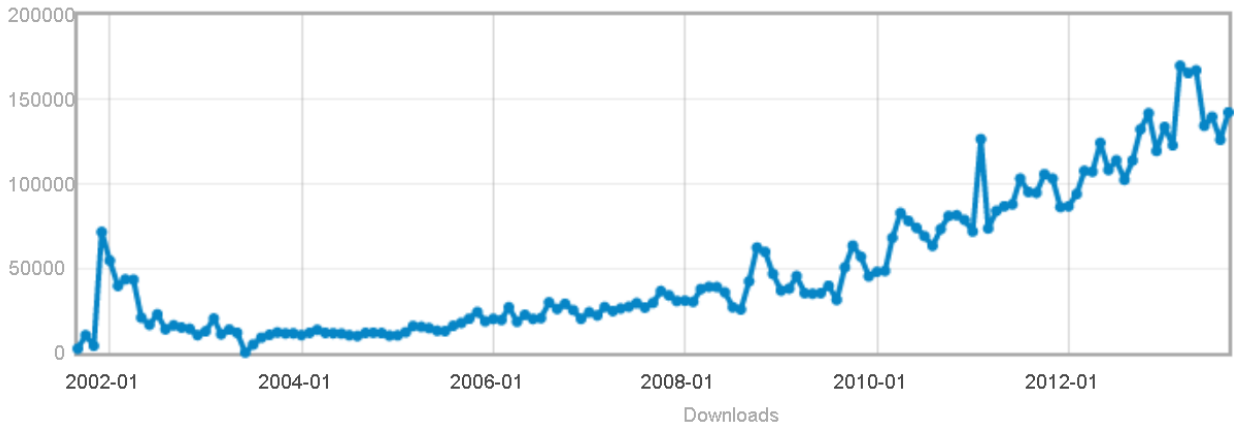


Figure 16 (OpenCV Lib Downloads)

3-5 Tesseract OCR Engine:

3-5-1 History:

The Tesseract engine was originally developed as proprietary software at HP labs in Bristol, England and Greeley, Colorado between 1985 and 1994, with some more changes made in 1996 to port to Windows, and some migration from C to C++ in 1998. A lot of the code was written in C, and then some more was written in C++. Since then all the code has been converted to at least compile with a C++ compiler. Very little work was done in the following decade. It was then released as open source in 2005 by Hewlett Packard and the University of Nevada, Las Vegas (UNLV). Tesseract development has been sponsored by Google since 2006.

3-5-2 Features:

Tesseract was in the top three OCR engines in terms of character accuracy in 1995. It is available for Linux, Windows and Mac OS X, however, due to limited resources only Windows and Ubuntu are rigorously tested by developers.

Tesseract up to and including version 2 could only accept TIFF images of simple one column text as inputs. These early versions did not include layout analysis and so inputting multi-columned text, images, or equations produced a garbled output. Since version 3.00 Tesseract has supported output text formatting, OCR positional information and page layout analysis. Support for a number of new image formats was added using the Leptonica library. Tesseract can detect whether text is mono-spaced or proportional.

The initial versions of Tesseract could only recognize English language text. Starting with version 2 Tesseract was able to process English, French, Italian, German, Spanish,

Brazilian Portuguese and Dutch. Starting with version 3 it can recognize Arabic, English, Bulgarian, Catalan, Czech, Chinese (Simplified and Traditional), Danish, German (standard and Fraktur script), Greek, Finnish, French, Hebrew, Hindi, Croatian, Hungarian, Indonesian, Italian, Japanese, Korean, Latvian, Lithuanian, Dutch, Norwegian, Polish, Portuguese, Romanian, Russian, Slovak (standard and Fraktur script), Slovenian, Spanish, Serbian, Swedish, Tagalog, Thai, Turkish, Ukrainian and Vietnamese. Tesseract can be trained to work in other languages too.

Tesseract is suitable for use as a back-end, and can be used for more complicated OCR tasks including layout analysis by using a front-end such as OCRopus.

Tesseract's output will be very poor quality if the input images are not preprocessed to suit it: Images (especially screenshots) must be scaled up such that the text x-height is at least 20 pixels, any rotation or skew must be corrected or no text will be recognized, low-frequency changes in brightness must be high-pass filtered, or Tesseract's binarization stage will destroy much of the page, and dark borders must be manually removed, or they will be misinterpreted as characters.

3-5-3 Architecture of Tesseract:

Tesseract OCR works in step by step manner as per the block diagram shown in figure 17 First step is Adaptive Thresholding, which converts the image into binary images. Next step is connected component analysis, which is used to extract character outlines. This method is very useful because it does the OCR of image with white text and black background.

Tesseract was probably first to provide this kind of processing. Then after, the outlines are converted into Blobs. Blobs are organized into text lines, and the lines and regions are

analyzed for some fixed area or equivalent text size. Text is divided into words using definite spaces and fuzzy spaces. Recognition of text is then started as two-pass.

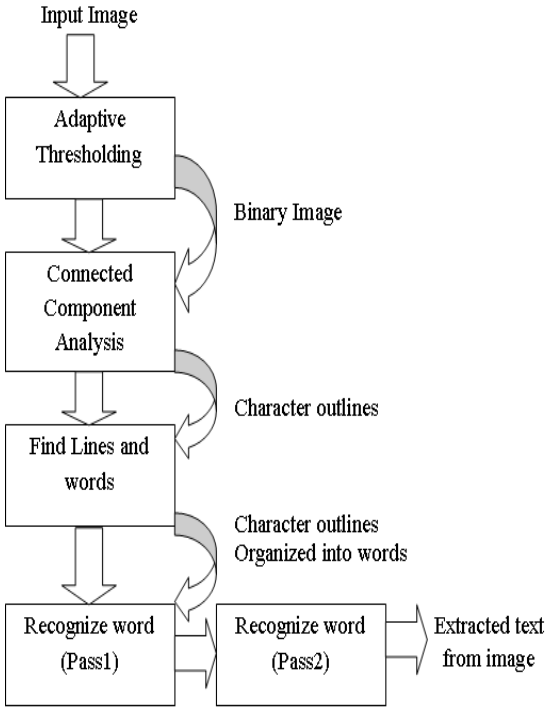


Figure 17 (Architecture of Tesseract OCR)

Process as shown in figure 17. In the first pass, an attempt is made to recognize each word from the text. Each word passed satisfactory is passed to an adaptive classifier as training data. The adaptive classifier tries to recognize text in more accurate manner. As adaptive classifier has received some training data it has learn something new so final phase is used to resolve various issues and to extract text from images. More details regarding every phase.

3-2 Step by Step Processing:

3-2-1 Pre-processing

3-2-1-1 Binarization

The first step is to binarize the image in order to separate text from background in grayscale image (which can be derived from RGB). Two methods were evaluated to be successfully applied in this project, the method is locally thresholding. The reason we choose locally adaptive thresholding instead of global threasholding is that the lighting/brightness of the image in not uniform, which will cause global thresholding to perform poorly in the extreme bright/dark regions. The idea of locally adaptive thresholding is divide the image into blocks/windows. For each block, use grayscale variance to determine whether the block is non-uniform (high variance), apply Otsu's method/global thresholding to the block; if the block is uniform (low variance), then classify the entire block as all black or all white based on the mean grayscale value. The reason not to apply Otsu's method to every block is that if some blocks are background with a number of noise pixels, Ostu's method will keep the noise pixels while classifying the entire block as background will eliminate noise.

OpenCV function adaptive Threshold is applied for binarization. It is observed that a blockSize of 51 yields a good trade- off between efficiency and thresholding effect. See Figure 18, and Figure 19 for an example of image before and after binarization with different light sources.

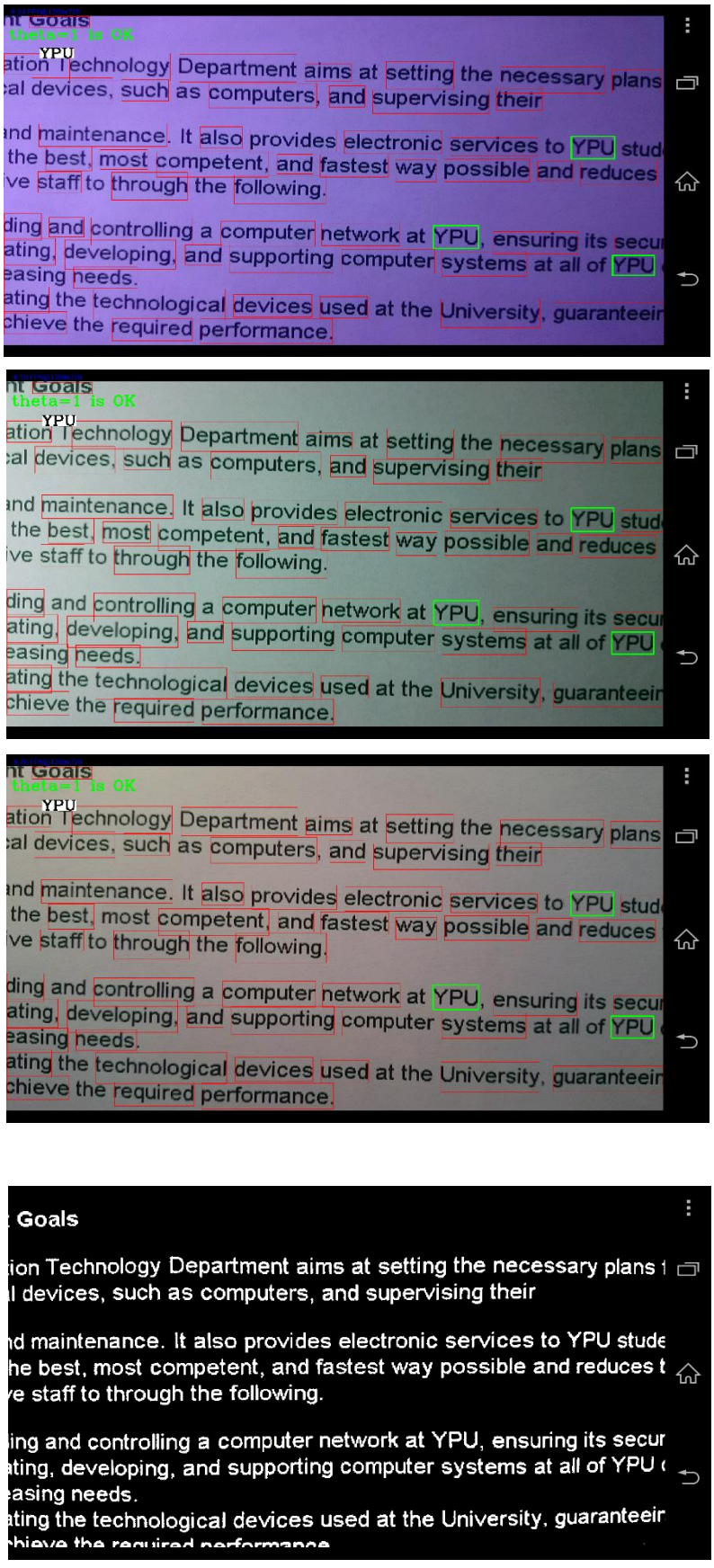


Figure 18 (colored image with different light sources)

Figure 19 (Binary image)

3-2-1-2 Rotation

The input image from the camera frame can be taken at an angle, so the lines of text may not be horizontally aligned. This affects on the following OCR step performance and not be as expected, so the image should be rectified. First of all, Hough transform is used to detect (text) lines in the image. Then the binary-image is rotated by the mean of the rotation angles which is calculated form each line. Observing that Tesseract does a decent job for skewed images by less than 5 degrees; We have included logic to bypass de-skew of rotation for such images. This improves the accuracy because the rotation decreases the image quality after interpolation. By skipping rotation, we essentially preserve more details in the image, and therefore boost performance for images with a small skew

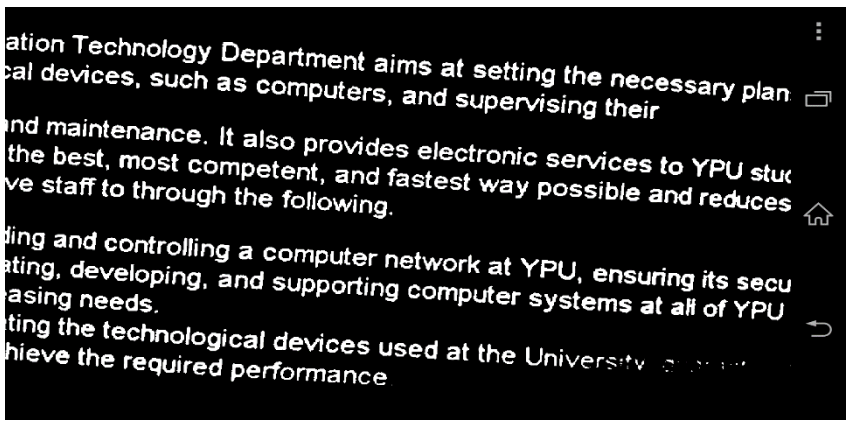


Figure 20 (Skewed binary Image)

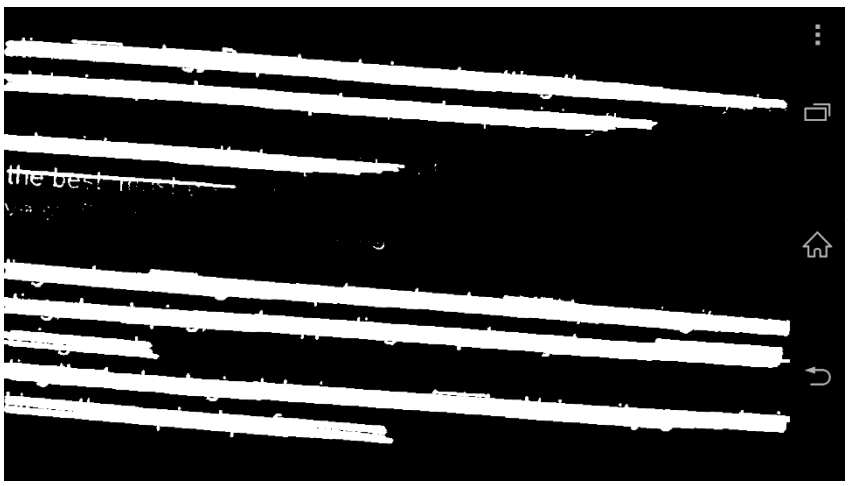


Figure 21 (Hough Lines)

However, we decided to let the user adjusts the tilt by his hands for the slow rotation of the matrix (as we explained above), the figures below show how the application gently asks the user to adjust the phone with the text to provide the most accurate results with a threshold of 5 degrees.

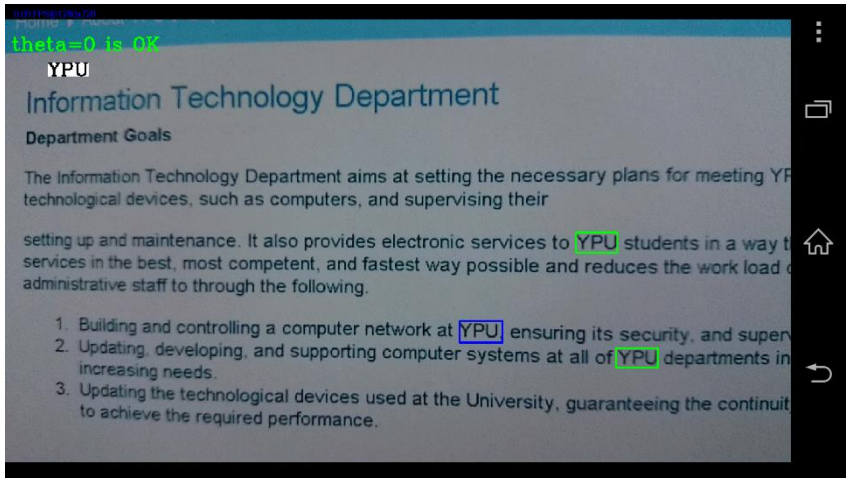


Figure 23 (Word Finder UI shows the tilt degree of 0)

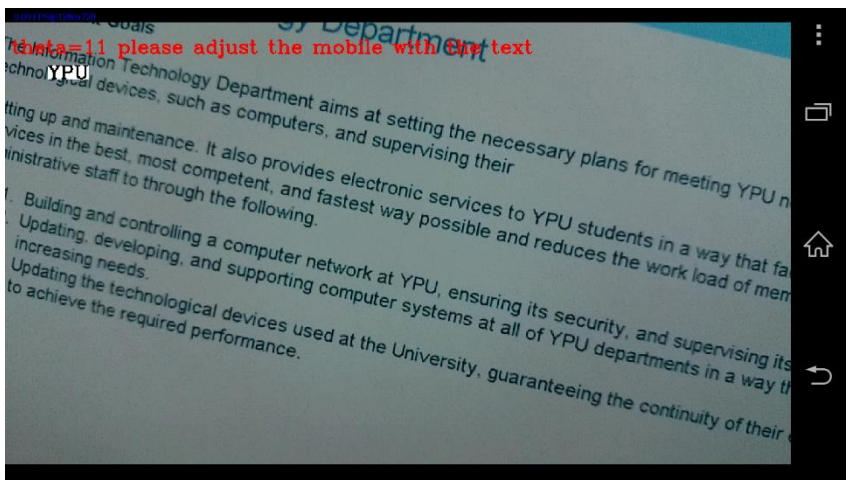


Figure 22 (Word Finder UI shows the tilt degree of 11)

3-2-1-3 De-noising

We also performed median filtering with a kernel size of 3 to eliminate any salt-and-pepper noise. This improves the accuracy of the characters detection because otherwise the size of noise spots will introduce error into character size calculation. That affects the extraction of connected components in a further steps of recognition.

3-2-2 Recognition

3-2-2-1 Smart Segmentation By Word

The motivation/assumption behind this method is that in a normal text document, linespacing, character size, word spacing, etc. scales linearly, i.e. large characters means large spacing, and vice versa. Therefore, we designed an algorithm to detect each word separately. The algorithm is: First we try to connect the characters of one word by applying the Closing Morphology on the filtered binary image with an elliptic structuring element, that is, a filled ellipse inscribed into the rectangle of size (9x2) the result shown in the figure below

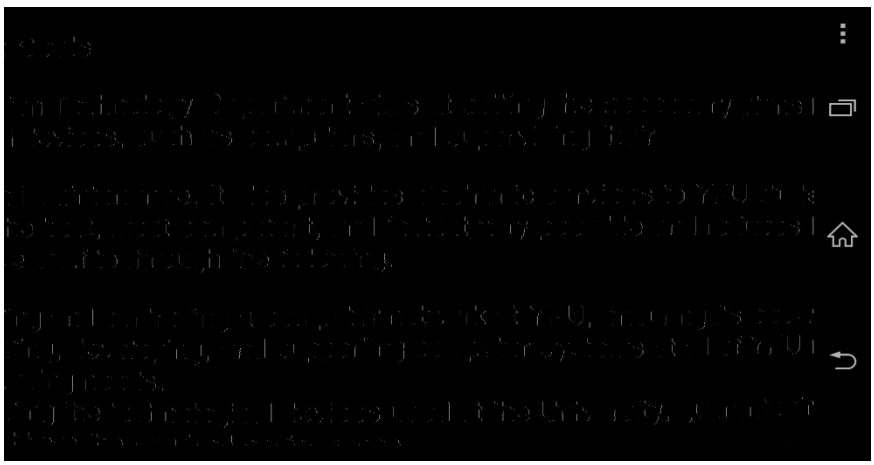


Figure 24 (the extracted bits after applying the closing morphology)

And by adding the Closed image to the binary one we result connecting the chars of the same word together and our image is now looking like the below figure

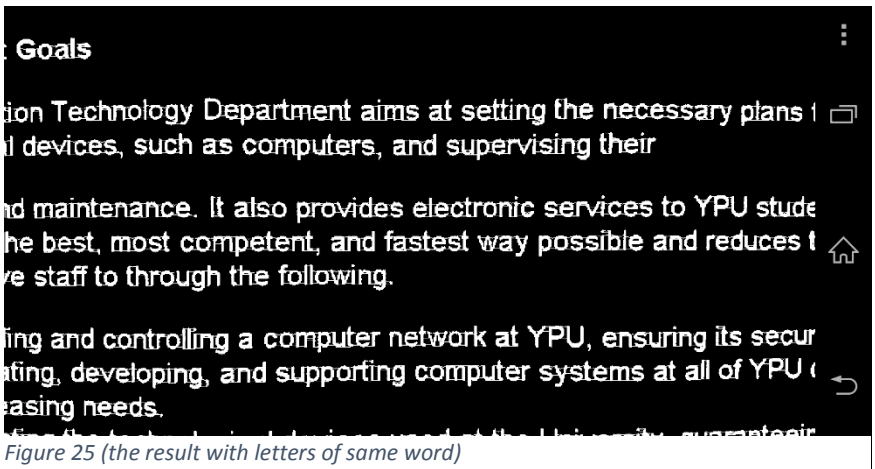


Figure 25 (the result with letters of same word)

Now by calculating the connected elements in the image and boundary them with a rectangle we finish the segmentation step with an image looks like the following one

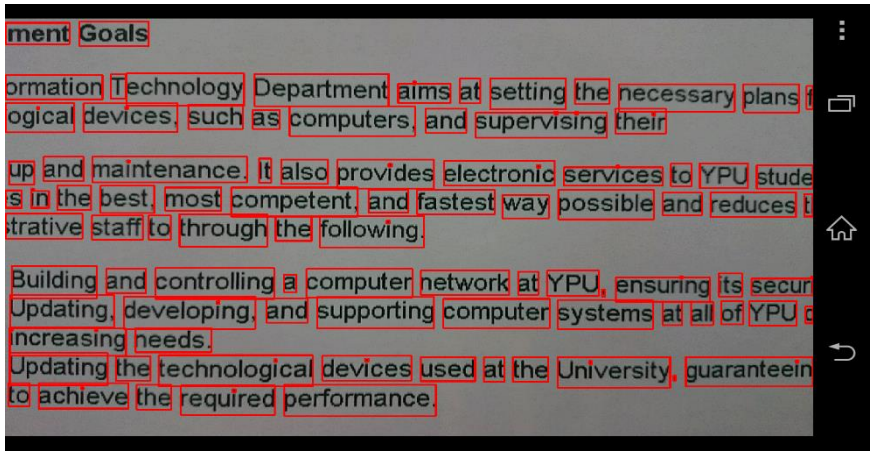


Figure 26 (bounding the contour of each word)

But as we see a for some letters (for instance, lowercase "i","j") which consists of two connected component, hence will have two bounding boxes, to fix this we decide to extend the height of each boundary box by 20%~ which doesn't affect the

3-2-2-2 Search and label matches:

At last bounding boxes (hopefully each containing exactly one word at this point) can be passed to the Tesseract OCR engine. In order to improve efficiency, we implement a function to filter out boxes too wide or too narrow given the keyword length. Sometimes image is out of focus or blurry due to vibration, therefore the result from Tesseract is not accurate. To cope the imperfection, we label exact matches with Green rectangles, non-exact matches with a blue rectangle. Exact or non-exact matches is determined by the ration between edit distance (or levenshtein distance) and the length of the keyword. A ratio of exact zero (or edit distance equal to zero) meant that is an exact match. A ration reasonably close to zero means that we have a non-exact match.

Levenshtein distance is defined to be minimum number of single-letter operation (insertion, deletion or substitution) needed to transform one string into another. For example, to change “abcd” into “bcda”, one can either change each letter (change the first letter "a" to "b", the second letter "b" to "c", the third letter "c" to "d", the fourth letter "d" to "a"), which has a total of four operations, or delete the "a" from the beginning of the string and add an "a" to the end of it, which has a total of two operations. Therefore, the Levenshtein distance between the two strings is two. Finally, we draw rectangular boxes of different colors (in order to differentiate between exact matches and close matches) at the coordinates where we find matches, and then overlay the rectangles onto the original image.

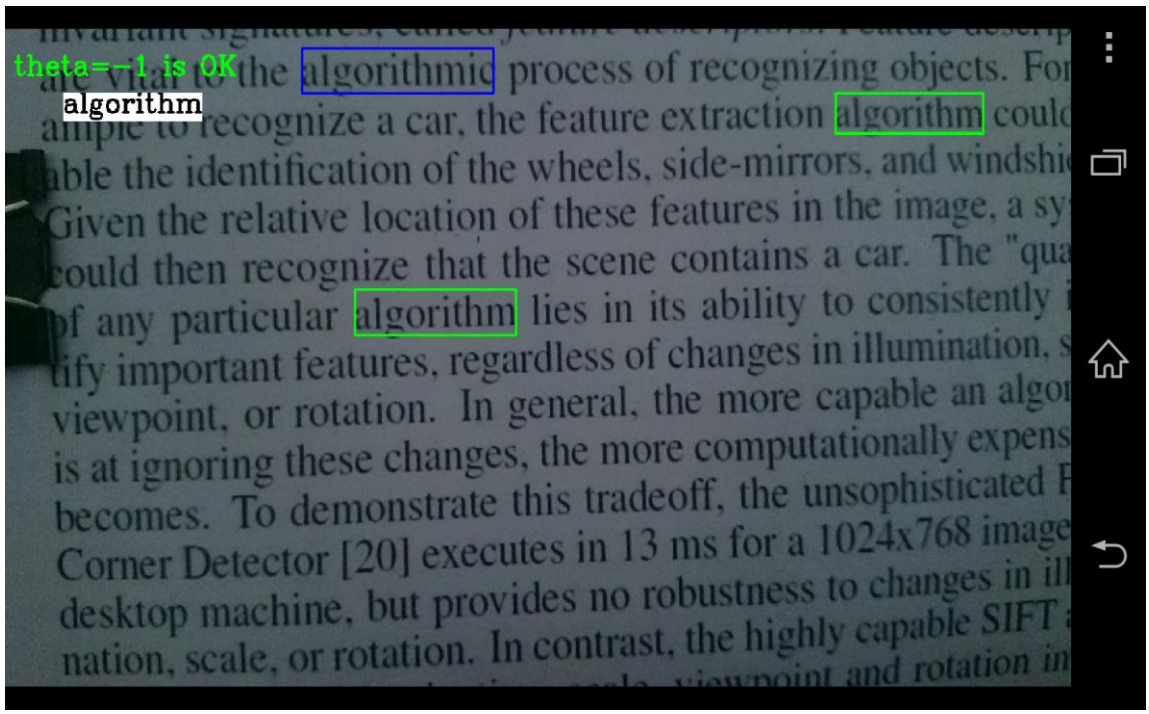


Figure 27 (Example of finding exact and non-exact matches)

Limitations

Computer vision projects are very time intensive ones; it require a lot of try and error and the ability to expect problems at any time.

The main limitation we faced within this project are:

- Slow processing

Due the fact of processing the whole steps with a mobile device which have relatively slow CPU and GPU comparing to a PC, which forced us to fix the mobile on a tripod to get the most accurate results.

- De-Skew

No more than 90° of rotation allowed in both (CW, CCW).

Conclusion and Future Work

We tried several methods and techniques to successfully identify a word from a corpus of words within an image. With local thresholding, de-skewing, and smart segmentation, we were able to successfully segment each word in a majority of test cases. Both preprocessing and post-processing were tailored to the powerful Tesseract OCR, Multi-language (Java, C++) were introduced in this project to bypass numerous configuration difficulties.

OCR is an extremely computationally intensive operation. To perform this operation using the processors on an outdated smartphone would be impractical. However, in a further work on the project, we can offload OCR and all other image-processing operations onto a server. The server side would be consists of two layers: The outer layer would be a PHP script that waits for an HTTP connection. Moreover, the processing side which should have all necessary libraries to run OCR and OpenCV in the other layer. Therefore, the two layers could communicate using python script to provide the smooth feeling to the end user on the mobile phone. In addition, to improve user experience and make our app interesting, we could further integrate the voice-aided search.

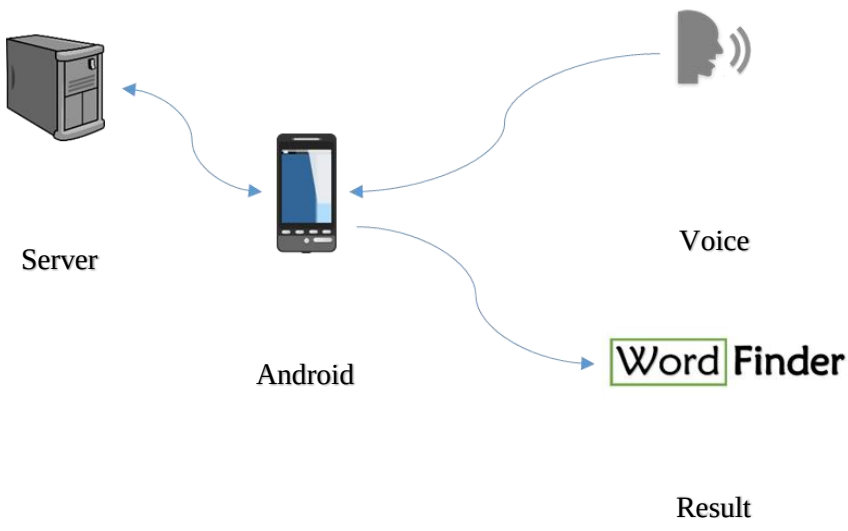


Figure 28 (Prototype of client-server connection with voice input)

List of Figures:

References

Brave, S. (n.d.). *Optical Character Recognition Using Artifical Neural Network.* International Journal of Advanced Technology & Engineering Research (IJATER).

Patel, P. A. (October 2012). *Optical Character Recognition by Open Source OCR , Tool Tesseract: A Case Study.* International Journal of Computer Applications.

Smith, R. (n.d.). *An Overview of the Tesseract OCR Engine.* Google Inc.

Szeliski, R. (2011). *Computer Vision, Algorithms and Applications.* Springer.

Yi-Tong Zhou, R. C. (n.d.). *Artificial neural networks for computer vision.* Springer.