Automated Mobile UI Testing of Cross-Platform Based on Computer Vision

Weinan Zhang
U5687862
Supervised by Associate Pro. Zhenchang Xing

June 2020

A thesis submitted for the degree of Master of Computing of the Australian National University

COMP 8755 Individual Project (12 Units)

Research School of Computer Science
The Australian National University

© Weinan Zhang 2020
All Rights Reserved
This thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of the author’s knowledge, it contains no material previously published or written by another person, except where due reference is made in the text.

Weinan Zhang
31 May 2020
I received useful advice and valuable inspiration from many people around me, which motivated me towards the completion of this thesis. Without the support that those people gave me, I could not have completed this project. I would like to thank you all here.

Thank you to Associate Prof. Zhenchang Xing, my supervisor for supervising me in this project. He gave me the assistance and guidance for my thesis and all other related assessments. I appreciate your valuable feedback regarding my thesis.

Thanks to Dr. Dehai Zhao for providing helpful assistance during the entire project. He gave me so much support and advice for my idea, that I really appreciate. He helped me from the beginning of the project to the end and support me all the time. I have gained more skills about computer version and software testing from him. I have learned so much from the advice he gave me in this field.

Thanks to Prof. Weifa Liang, the course convenor of this project. He gave me the valuable advice and guidance for my thesis, presentation, and guideline of the project. I really appreciate your useful feedback and improvement for this project.

Last but not least, thank you to my friends and colleagues who have spent time and effort proofreading my written work and given me valuable suggestions. Thank you to my family for the motivation and encouragement you have given me over the course of the project.
ABSTRACT

Automated mobile testing is commonly slow and wasteful for a simple feature. It is unable to test user experience factors. The automated testing software typically specific to each mobile app. It is also time-consuming to build automated test scripts and it cannot automate every test scenario. This thesis investigates how computer vision method applied to automated mobile user interface testing. Object detection is a method by which the predictions were implemented using neural network. This will be scalable for larger mobile apps and cross different platforms. Testing dataset is collected from the user interface of mobile app. The detected widgets are discussed using the trained YOLOV3 model. This thesis analyses the evaluation metrics used to validate the model. Possible improvements which can be carried out in the future are suggested.
Table of Contents

Contents
Table of Contents .................................................................................................................... V
List of Figures ......................................................................................................................... VII
List of Tables .......................................................................................................................... IX
Glossary of Terms .................................................................................................................. X
Chapter 1 Introduction ........................................................................................................... 1
  1.1 BACKGROUND ......................................................................................................... 1
  1.2 PROJECT MOTIVATION ......................................................................................... 3
  1.3 OBJECTIVES AND DESCRIPTION OF THE PROJECT ........................................ 4
  1.4 THESIS STRUCTURE ............................................................................................. 4
Chapter 2 Literature Review ................................................................................................. 5
  2.1 LIRAT -UI TESTING ............................................................................................... 5
    2.1.1 System Implementation .................................................................................. 5
    2.1.2 Script Recording and Replaying ..................................................................... 6
    2.1.3 LIRAT Evaluation ......................................................................................... 8
  2.2 FASTER R-CNN ...................................................................................................... 10
    2.2.1 Object Detection System .............................................................................. 11
    2.2.2 Regional Proposal Network ......................................................................... 12
    2.2.3 Evaluation ................................................................................................... 13
  2.3 YOLO .................................................................................................................... 15
    2.3.1 YOLO Detection System and Architecture .................................................. 15
    2.3.2 YOLO Evaluation ....................................................................................... 17
    2.3.3 Comparison with R-CNN ............................................................................ 17
  2.4 HUMANOID ............................................................................................................ 19
    2.4.1 Approach Review ....................................................................................... 19
    2.4.2 Architecture ............................................................................................... 20
    2.4.3 Humanoid Evaluation ................................................................................ 21
Chapter 3 Methodology ........................................................................................................ 23
  3.0 METHODOLOGY OVERVIEW ............................................................................. 23
  3.1 DATASET PRE-PROCESSING .............................................................................. 24
  3.2 YOLOV3 ............................................................................................................... 25
    3.2.1 Bounding Box Prediction ............................................................................ 25
    3.2.2 Class Prediction ......................................................................................... 26
    3.2.3 Network Architecture ............................................................................... 26
List of Figures

Figure 1-1: UI Components on mobile device example based on appery.io ......................... 1
Figure 1-2: Human testers to operate the GUI and testing the UI behaviour visually traditionally [6] .................................................................................................................. 2
Figure 1-3: Computer Vision used in mobile user interface testing [11]. .............................. 3
Figure 2-1: LIRAT workflow [15]. ....................................................................................... 5
Figure 2-2: Script recording and replaying on the UI of LIRAT [15] .................................... 6
Figure 2-3: Widget attributes, page screenshot information [6] ........................................... 7
Figure 2-4: Layout matching positioning, to detect the layout of the UI [11] ....................... 8
Figure 2-5: The comparison of RCNN, Fast R-CNN and Faster R-CNN [19] ....................... 11
Figure 2-6: Different schemes for addressing multiple scales and sizes [20] ....................... 11
Figure 2-7: Faster R-CNN is a single, unified network for object detection [20] ................. 12
Figure 2-8: Regional Proposal Network (RPN) [20] ............................................................ 13
Figure 2-9: Recall and IoU overlap ratio [20] ..................................................................... 13
Figure 2-10: YOLO Detection System [28] ....................................................................... 15
Figure 2-11: YOLO Architecture [28] ............................................................................... 16
Figure 2-12: Error Analysis to compare Fast R-CNN vs. YOLO [28] ................................. 17
Figure 2-13: Comparison of different methods as to precision-recall curves [20] ............... 18
Figure 2-14: Humanoid chooses test inputs for UI state [34] ............................................. 19
Figure 2-15: System overview of Humanoid [34] ............................................................... 20
Figure 2-16: Humanoid Architecture [34] ......................................................................... 20
Figure 2-17: Progressive activity coverage for market apps [34] ........................................ 21
Figure 2-18: Compare with other methods [34] ................................................................. 22
Figure 3-1: Training Phase ................................................................................................. 23
Figure 3-2: Testing Phase .................................................................................................. 23
Figure 3-3: Dataset pre-processing ....................................................................................... 24
Figure 3-4: Bounding box with dimension priors and prediction [38] [39] It predicts the width and height of the box as offsets from cluster centroids ............................................. 25
Figure 3-5: Yolov3 Darknet 53 [38] ................................................................................... 26
Figure 3-6: YoloV3 Architecture and Convolutional Set [38] [41] ....................................... 27
Figure 3-7: SIFT method, the keypoints selection example ..................................................... 28
Figure 3-8: Feature matching examples.................................................................................. 29
Figure 4-1: Training Dataset Example The dataset is designed and used based the Rico Dataset [46]. ........................................................................................................................................ 31
Figure 4-2: Two groups of testing dataset (IOS app platform vs IOS web platform). ......... 31
Figure 4-3: The average loss during the training. The loss become stably at around 200...... 34
Figure 4-4: IoU description...................................................................................................... 35
Figure 4-5: Recall and Precision............................................................................................ 35
Figure 4-6: Random test on Android platform by using Rico Dataset ......................... 36
Figure 4-7: IOS App platform testing.................................................................................... 37
Figure 4-8: IOS web platform testing.................................................................................. 39
Figure 4-9: Similarity comparison....................................................................................... 40
Figure 9-1: The files of artefacts............................................................................................ 51
Figure 9-2: Testing dataset................................................................................................... 52
Figure 9-3: Hardware requirements...................................................................................... 52
Figure 9-4: The weights attained during the training.......................................................... 53
List of Tables

Table 2-1: The applications used in LIRAT system [15] ........................................................... 8
Table 2-2: The devices used in LIRAT testing [15] .................................................................... 9
Table 2-3: The number of successful script replaying [15] ....................................................... 9
Table 2-4: Detection mAP (Percent) of Faster R-CNN [20] .................................................... 14
Table 4-1: Widgets that the model can identify in UI. The icons of the buttons were collected from author’s phone ................................................................................................................ 32
Table 4-2: Hyperparameters for the network model .................................................................. 33
Table 4-3: The varying indicators of errors ............................................................................. 34
Table 4-4: Evaluation metrics results for Android platform ..................................................... 37
Table 4-5: The evaluation metrics for IOS App platform .......................................................... 38
Table 4-6: Evaluation metrics for IOS Web Platform ............................................................... 38
Table 9-1: Software requirements ............................................................................................ 53
### Glossary of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>MUI</td>
<td>Mobile User Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>R-CNN</td>
<td>Region Convolutional Neural Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-term Memory</td>
</tr>
<tr>
<td>YOLO</td>
<td>You Only Look Once (Object Detection Algorithm)</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction

1.1 BACKGROUND

Smartphones and tablets are the essential part of people's lives. User interface designers is trying to catch up by building more attractive interfaces. That is because a new design is only a few touches away. Developers made the application as user-friendly as possible. However, mobile device capabilities constantly evolve, mobile device usability is a moving target [1].

Mobile app user interface testing is mainly about ensuring that the UI functions on the application of mobile device in the right, that an app follows its written specifications and that defects are identified. It is a software testing type that will check the user interface of the application under test [2]. MUI testing basically involved in checking the screen with the controls, such as menus, buttons, icons, and dialog boxes, etc. The aim of MUI testing is to ensure the functionality of UI of the applications work well as per the specification [3].

Figure 1-1: UI Components on mobile device example based on appery.io
Tester typically uses different test cases to set the criteria to assess the device is functioning as it should. There are two methods to do that. The first is manually tested by a human software tester. Another method is the automatically testing which done by a software program [4].

Manual testing means having a human tester running a series of operations and effectively manually verifying whether the app is acting in the right way and graphical displays are consistent with the requirements [4]. While this approach has some benefits, it has some problems including the fact that it can be time-consuming, it needs a lot of work and the consistency really depends on the tester’s capabilities [5].

Automated UI testing is the automation of manual test tasks. Because manual test tasks can be time-consuming and prone to error, using automated UI testing as a more accurate, effective and reliable process. Automated UI testing is a cost-effective alternative for manual testing over an extended timeframe. However, it should be introduced after any functionality is already finished. Developing them alongside mainstream implementation could results of mistakes while retaining new functionality [7].

Computer vision is an artificial intelligence method that helps computers to identify and label images in various areas, such as driverless car testing and monitoring crops and livestock [8]. Machines can accurately recognize and classify objects using digital images from cameras and videos, and then respond to what they "see" [9].

Figure 1-2: Human testers to operate the GUI and testing the UI behaviour visually traditionally [6]
Computer vision methodology can also be extended to GUI testing for testers to simplify their tasks. Using pictures to write a visual test script to determine which GUI components to interact with and what visual feedback to be received. Testers can also generate visual test scripts by demonstration [10]. Through recording all input events and screen images, images of interacted components and visual feedback from the demonstrator can be extracted and a visual test script created automatically. Recent research shows that GUI behavior can be tested using this approach. This also indicates how this approach can promote excellent testing practices including unit testing, regression testing, and test-driven development [6].

![Figure 1-3: Computer Vision used in mobile user interface testing](image)

1.2 PROJECT MOTIVATION

Cross-platform app testing is difficult due to system settings and mobile OS on the market. Since testing the app in a single device does not guarantee proper operation in others, each device represents a configuration model, hardware, screen size, sensors which need to be verified [12]. Although automation is essential to many applications, existing test frameworks are not cross-platform. For example, a UI test script using a tool like Appium must be written twice as Android and iOS XML representations of the UI are not same [13]. Such representations may also differ between platform versions, as in Android 4 and Android 6. Cross platform apps are running in different operating systems, like Android, IOS, and Windows. These apps are developed by using different frameworks for cross-platform app development like Apache, Cordova, Xamarin, and React Native [14]. However, the automated
testing generally not support the cross platform. Different testing system will be built for each platform, and there is no guarantee they will work properly in different platform. This project is motivated by this and proposes the computer vision method based on previous work to implement the automated tests for cross platform mobile apps.

1.3 OBJECTIVES AND DESCRIPTION OF THE PROJECT

The objective of this project is to apply computer vision method to software engineering, which using object detection method to implement the automated testing on cross platforms. This project using the Rico dataset as the training dataset to train Yolov3 model, which is a fast and powerful neural network. After training, this project using the collected testing dataset to test the trained model. The testing dataset are collected from three different platforms to test the object detection method is working cross platform. In addition, SIFT similarity comparison method also be used to evaluate the results. For measuring the accuracy of the results, the evaluation metrics are provided to evaluate the results based on different platforms.

1.4 THESIS STRUCTURE

This thesis consists of six chapters. After the introduction of Chapter 1, literature related to automated user interface testing and computer version methods are reviewed in Chapter 2. A few different automated testing models, LIRAT models and deep learning-based model are introduces. The methods of object detection are also reviewed, like fast R-CNN in Chapter 2. Then the methodology used in this project are also introduced in Chapter 3. It involved the dataset chosen, neural network model implementation, and the SIFT similarity comparison. After that, Chapter 4 introduce the method to evaluate the neural network model. It does show the training dataset, testing dataset, hyperparameters and evaluation metrics. Chapter 5 will demonstrate the predicted results by the model. The discussion part shows the benefits and limitation of object detection method used in automated mobile app UI testing. Conclusion regarding the achievements and problems from this project are drawn in Chapter 5 along with the potential improvements suggested for the future study.
Chapter 2 Literature Review

2.1 LIRAT - UI TESTING

Regarding automated mobile cross-platform testing, LIRAT was accomplished by solving the problems of recording and replaying test scripts on different platforms. LIRAT (layout and Image Recognition Driving Tool) is an image-driven method to capture and replay cross-platform test scripts, first addressing the problem of cross-platform test script replay [15]. LIRAT records screenshots and widget layouts and uses imaging techniques to locate them in the process of replay. LIRAT allows replaying test scripts through devices and platforms, depending on the exact button location [15].

There are 25 scripts from 5 applications on 8 android devices, and 2 IOS devices are replayed to test LIRAT. The results of this approach could replay scripts on Android platforms with 88%, and IOS platforms with 60% [15].

2.1.1 System Implementation

![Figure 2-1: LIRAT workflow [15]](image-url)
LIRAT applies SpringBoot to server-end framework, Angular2 to client-end framework, and transfers information through Netty and WebSocket. In RESTful interfaces the image processing server provides external services. The system administration server manages linked mobile devices through USB. The management includes device access, status monitoring, distribution and execution of commands. For different Android and iOS platforms, LIRAT selects ADB and WDA to perform system control [15].

2.1.2 Script Recording and Replaying

Script recording is a single-stage operation recording process, it is extracting and recording widget screenshot and layout information for each operation, also including some widget attribute information [15].

The information extracted from the widget is recorded and the corresponding XML files are generated. The XML files are stored in the form of a nested directory together with the screenshot, the root directory represents the recorded script, each subdirectory is the proprietary widget file for each operation, and the operating sequence file is stored in the root directory [15].

Figure 2-2: Script recording and replaying on the UI of LIRAT [15]
LIRAT will retrieve the script file from the database and replay it in an orderly manner when users select the script and replay devices. LIRAT uses information about the widget and page screenshot, and the widget attributes to locate the widget on the replay devices [6].

LIRAT provides two techniques for positioning the widget: positioning matching the image and positioning matching the layout, which could highly support this project. The main supported part is to compare the detected UI components similarity.

Image matching positioning is the algorithm that is used between the screenshots of the app taken and the replaying device's current screenshot page to find the app positioning. The limitation of this method is that when the widget screenshot in the replaying device changes a lot, or when the page contains many dynamic or similar images, the widget can hardly be positioned correctly [15].

Layout matching positioning will be applied when meeting dynamic or similar widgets. The layout matching positioning is to separate the page screenshot into the script file, and use OCR technology to draw the layout, and obtain the widget 's location coordinate information under the replay device's current page configuration. This method is focused on the device layout details. Since most separate system and platform implementations have a common interface, cross-device and cross-platform replay can be best understood [11].
2.1.3 LIRAT Evaluation

To verify the reliability of the LIRAT system in the real situations. This method uses 5 applications and 10 different devices to validate this system. The table below shows the lists of the application the system used, and the devices used.

The applications are used in the testing crossed the different categories of the application. This is to ensure the LIRAT is working well in different UI design and different UI component. In addition, in order to compare, the system versions are also varied. The version is different for the same application to compare and validate the reliability of LIRAT.

To ensure the integrity of this method, the testing also choose the device from different platform. They are included different OS version and different screen size to validate this model.

<table>
<thead>
<tr>
<th>App ID</th>
<th>App Name</th>
<th>Category</th>
<th>Android Version</th>
<th>iOS Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Xiaomi Calculator</td>
<td>Tool</td>
<td>1.0.5</td>
<td>1.0.3</td>
</tr>
<tr>
<td>A2</td>
<td>AnTuTu Benchmark</td>
<td>Tool</td>
<td>7.2.8</td>
<td>7.2.0</td>
</tr>
<tr>
<td>A3</td>
<td>One App</td>
<td>Reading</td>
<td>4.5.9</td>
<td>4.6.2</td>
</tr>
<tr>
<td>A4</td>
<td>Shark Accounting</td>
<td>Finance</td>
<td>2.3.7</td>
<td>2.3</td>
</tr>
<tr>
<td>A5</td>
<td>QQ Movie</td>
<td>Amusement</td>
<td>3.2.0</td>
<td>1.3.2</td>
</tr>
</tbody>
</table>

Table 2-1: The applications used in LIRAT system [15]
Table 2-2: The devices used in LIRAT testing [15]

The table below shows the results of the number of successful script replaying. The findings show that the amount of successful cross-device replay exceeded 88%, and the number of successful cross-platform replay is over 60%, which means LIRAT has made significant strides in replaying mobile application test scripts on various apps and platforms [15].

Typically, smartphone applications run on different devices and platforms, and limited test scripts between devices and platforms can result in repetitive work. LIRAT addresses these issues by combining image recognition technology, SIFT and interface positioning technology, OCR and Canny, accurately putting widgets on various devices and platforms, dramatically simplifying scripting work and allowing one script capture, simultaneous scripting in different devices and platforms [15].

Table 2-3: The number of successful script replaying [15]
In summary, this system provides an excellent idea for developers to test the user interface based on different platform and different devices. This project also to test the user interface for different platforms, this provide an idea for the starting of this project. The SIFT, layout matching methods are valuable for this project. However, the limitation of this system is shown by the failing cases. They are mainly due to the difference between the UI of different platforms. It results of the failure of the matching od the widgets. Therefore, this project will decide to use computer version method to train the model, then the prediction will be illustrated based on the learning model. This will enhance the accuracy of the UI testing to avoid the differences of different UI in different platforms.

2.2 FASTER R-CNN

R-CNN (Regions with Convolutional Neural Networks) is the first step towards faster R-CNN. It uses search selective to pick interest regions and pass them to a ConvNet. It attempts to determine the areas that may an object by combining similar pixels and textures into several rectangular boxes. The R-CNN paper includes 2,000 proposed areas from search selective. Furthermore, passing these 2,000 areas to a pre-trained CNN model. Finally, the outputs (feature maps) are passed to classification SVM. The regression between predicted bounding boxes (bboxes) and ground-truth bboxes is calculated [16].

Fast R-CNN moves one step ahead. Instead of applying 2,000 times CNN to proposed areas, it only moves the initial image to a pre-trained CNN model once. Search selective algorithm is determined based on the output feature map of the previous step. ROI pooling layer guarantees the standard and pre-defined output size. These valid outputs are passed as inputs to a fully connected layer. Finally, with a softmax classifier, two output vectors are used to simulate the observed entity and conform bounding box localizations with linear regressor [17].

Faster R-CNN progress further than Fast R-CNN. Search selective process is replaced by Region Proposal Network (RPN). As the name indicated, RPN is a region-specific network. For instance, after obtaining the output feature map from a pre-trained model (VGG-16), the output feature map from a pre-trained model (VGG-16) will be 37x50x256 dimensions if the input picture has 600x800x3 dimensions [18].
2.2.1 Object Detection System

The training schemes are established to unify RPNs with Fast R-CNN Object Detection Networks, that alternates between fine tuning for the proposed region task and then fine-tuning for object detection, while maintaining proposals fixed. This scheme converges easily, creating a single network of convolutional features exchanged by all tasks [20].

Object detection technologies are driven by the success of region-based convolutional neural networks(R-CNNs). For the object detection system, Faster R-CNN does have two components. The first component is a deep fully convolutional network that proposes the regions, and the other component is the Fast R-CNN detector, which is using the proposed regions. This object detection system is a single, unified network for object detection [20].
2.2.2 Regional Proposal Network

A Region Proposal Network takes a picture as input and outputs a series of rectangular object proposals, each with an object score of 3. The model is the method of a fully convolutional network. The aim is to share computation with a Fast R-CNN object detection network, thereby ensuring all networks share a similar collection of convolution layers [21].

To establish region proposals, it designs a small network over the convolutional feature map output over the last shared convolutional layer. This small network takes a spatial window of the input convolutional feature map as data [20] [22].

Back-propagation and stochastic gradient descent (SGD) can train end-to-end RPN. It adopts the "image-centric" approach to train this network [23]. Each mini batch comes from a single picture comprising both positive and negative anchors. Optimize for the failure features of both hosts, although that can skew against pessimistic tests because they rule. Alternatively, it randomly select 256 anchors in an picture to determine a mini-batch loss function where
positive and negative anchors have a ratio of up to 1:1. When an image comprises fewer than 128 positive tests, this method patches the mini-batch with negatives [20].

Both RPN and Fast R-CNN, trained independently, will change their convolution layers differently. Therefore, this approach needs to develop a technique for sharing convolutional layers between the two networks, rather than learning two separate networks.

### 2.2.3 Evaluation

![Figure 2-9: Recall and IoU overlap ratio [20]](image)
The figure (Figure 2-9) show the outcomes of using 300, 1,000, and 2,000 proposals. Comparing with SS, EB and MCG, and the N proposals are the top-N ranked based on the confidence generated by these approaches. The graphs demonstrate that the RPN system reacts remarkable when the number of proposals decreases from 2,000 to 300. This illustrates that the RPN has a successful mAP by using only 300 proposals. This property is mainly related to the cls term of the RPN. The recall of SS, EB and MCG drops faster than RPN when proposals are fewer [20].

<table>
<thead>
<tr>
<th>training data</th>
<th>2007 test</th>
<th>2012 test</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOC07</td>
<td>69.9</td>
<td>67.0</td>
</tr>
<tr>
<td>VOC07+12</td>
<td>73.2</td>
<td>-</td>
</tr>
<tr>
<td>VOC07++12</td>
<td>-</td>
<td>70.4</td>
</tr>
<tr>
<td>COCO (no VOC)</td>
<td>76.1</td>
<td>73.0</td>
</tr>
<tr>
<td>COCO+VOC07+12</td>
<td>78.8</td>
<td>-</td>
</tr>
<tr>
<td>COCO+VOC07++12</td>
<td>-</td>
<td>75.9</td>
</tr>
</tbody>
</table>

Table 2-4: Detection mAP (Percent) of Faster R-CNN [20]

For both PASCAL datasets tests, Fast R-CNN was used as a detector. Using the RPN+ZF backbone as just a proposal network matched the performance of using "Selective Search" (SS) as a proposed region algorithm. This gives comparable results with a large detection time decrease. RPN+VGG backbone does marginally higher than the SS area proposal benchmark as a proposal network of unshared weights. Using mutual weights with the detector, both ZF and VGG backbones in RPN exceeded SS baseline results. This, along with several other studies, confirmed the usage of RPN as a regional proposal process [24] [20].

At the first stage of this project, Faster R-CNN was suggested to be used on the automated user interface testing. However, although faster-R-CNN had an excellent performance on the object detection, it still has high requirements about the CPU and GPU. Faster R-CNN already used on the automated user interface testing and achieved good results. Therefore, it is decided that to explore another similar but more efficiency and faster method to be used in user interface automated testing.
2.3 YOLO

YOLO (You Only Look Once) is a real-time object detection algorithm, one of the powerful object detection algorithms, combining with many of the most innovative concepts from the computer vision research community. Object detection is important to autonomous vehicle technology [25] [26].

Object detection is one of the classic computer vision applications working to recognize what and where — specifically what objects are inside a given image and where they are inside the image. Object detection is more complex than classification, which can recognize objects, but does not indicate where the object is in the image. Moreover, classification doesn't fit on more than one object on images [27].

2.3.1 YOLO Detection System and Architecture

![Figure 2-10: YOLO Detection System](image)

Present detection systems repurpose classifiers to conduct object detection. To detect an object, these systems take a classifier in a test image and evaluate it at various locations and scales. Systems like deformable part models (DPM) use a sliding window method here the classifier runs all over the image at entire spaced locations [29].

YOLO is quite straightforward (as seen the figure above). Simultaneously, a single convolutional network predicts multiple bounding boxes and class probabilities. YOLO trains on full images and improving detection performance. This unified model has many benefits over conventional methods of object detection [28].
The YOLO network architecture is inspired by the GooLeNet, which is a model designed for image classification [30] [28]. YOLO detection network does have 24 convolutional layers and with 2 fully connected layers. YOLO network uses alternate 1x1 convolutional layers to reduce the feature spaces from previous layers. It does pertain the convolutional layers on the ImageNet classification task at half the resolution and then will be doubles the resolution for detection [28].

YOLO uses a linear activation function between final layer and all other layers. The leaky rectified linear activation function is used [31].

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

(1) [28]

This approach optimizes YOLO model performance by using sum-squared error. It uses sum-squared error since it is easy to optimize, but not exactly matched with the objective of optimizing average accuracy. It weights localization error equally with classification error that is not ideal [28]. To boot this, this architecture improves the loss from bounding box coordinate predictions and reduces the loss from confidence predictions for boxes that do not contain objects [32].
2.3.2 YOLO Evaluation

YOLO places strong spatial restrictions on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits how many adjacent objects can be predicted. YOLO model failed with small objects that appear in groups, such as flocks of birds [28]. As YOLO model attempt to predict bounding boxes from data, it fails to generalize to objects in new or different aspect ratios or configurations. YOLO model also uses relatively coarse features to predict bounding boxes, as the architecture has multiple down sampling layers from the input image. Finally, while training on a loss function approximating detection performance, the loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign, but a small error in a small box affects IOU much more. The key cause of error is incorrect localizations [33].

2.3.3 Comparison with R-CNN

![Error Analysis to compare Fast R-CNN vs. YOLO](image)

**Figure 2-12: Error Analysis to compare Fast R-CNN vs. YOLO [28]**

YOLO is hard to localize objects correctly. Localization inaccuracies contribute for more of YOLO ’s errors than all others combined. Fast R-CNN makes less localization errors but far
more background errors. 13.6% of its top detections are false positives that contains no objects. Fast R-CNN predicts background detections nearly 3 times higher than YOLO [28].

![Figure 2-13: Comparison of different methods as to precision-recall curves](image)

YOLO provides a unified model for object detection. The model is easy to construct and can be trained directly on full images. Unlike classifier-based models, YOLO is trained on a loss function that explicitly corresponds to detection performance, and the entire model is trained jointly [28]. However, YOLO still gives an interesting idea for this project. Most of the YOLO algorithm is to do the object detection in terms of the real world. For the automated testing of mobile user interface, most approaches are typically about the R-CNN, fast-RCNN, faster-RCNN. Such methods will require higher GPU and CPU, which is expensive, and time costed. YOLO will be suitable for automated user interface testing. But YOLO also needs to be improved and implement on software testing platform. The user interface automated testing based on YOLO will be more efficiently and fast then others. In addition, the YOLO algorithm also needs to be strengthened to prevent the issue of wrong locating.
2.4 HUMANOID

Humanoid is a deep learning-based approach to automated Android app testing. The aim is to learn from human-generated interaction traces, to create human-like test inputs based on visual details in the current UI state and the latest state transformations. It develops and implement a deep neural network model to learn how end-users will interact with an app and demonstrate that they can effectively generate human-like inputs for any new UI based on the trained model. First extending the concept to automatic testing of Android devices and showing it can achieve higher coverage and quicker than state-of-the-art check input generators [34].

![Image of Humanoid testing](image)

**Figure 2-14: Humanoid chooses test inputs for UI state [34]**

2.4.1 Approach Review

Humanoid ’s core is a computer learning platform that studies how humans interacts with devices. Based on the interaction model, the whole method may be split into two phases, namely an offline step to train the model with traces of human-generated interaction and an online phase to direct the generation of test inputs. Using a neural-deep network architecture, it discovers the interaction between GUI environments and user-performed interactions during the offline learning process. A GUI context is represented as visual information in the current
UI state and the latest UI transitions, while an interaction is represented as action type and action location coordinates. After learning from signs of large-scale human activity, Humanoid may predict a distribution of the action form and place of operation for a new UI state. The expected distribution will then be used to quantify the likelihood and interaction with each UI item being encountered by humans [35]. During the online testing process, Humanoid generates an Interface model called UI Transformation Graph (UTG) for the test device. Humanoid utilizes both GUI model and interface model to determine what test data to give. UTG is responsible for leading Humanoid to navigate between existing UI states, while the concept of interaction directs the discovery of new UI states [34].

Figure 2-15: System overview of Humanoid [34]

2.4.2 Architecture

Figure 2-16: Humanoid Architecture [34]
This design uses 5 convolution layers with RELU activations to extract features from UI skeleton photos and activity heatmaps. After each convolutional layer, there is a stride-2 max-pooling layer that reduces the width and height of its input to half. Basically, pooling layers help the model identify UI components of the same shape but different surroundings. Extracting features from historical transitions is also a sequence modelling issue. This incorporates residual LSTM modules after the last 3 convolution layers to get UI transformation series features at various resolution speeds. In a residual LSTM node, a residual path provides the last input element and output of the regular LSTM [36]. De-convolutional layers are used to produce high-resolution probability distributions from the low-resolution output of residual LSTM modules. There are many methods for this, including bilinear interpolation and deconvolution. It approaches utilizes de-convolutional layers, since it is easier to integrate with deep neural networks than interpolation approaches. Fully connected layer is a single fully connected layer with softmax is used to generate the probability distribution of action types [34].

2.4.3 Humanoid Evaluation

![Progressive activity coverage for market apps][34].
Compared to open-source apps, market apps typically have different and more complex functionality and UI structures. Further experiments should be conducted on the market apps to see if Humanoid is still more effective. The final activity coverage achieved by the testing tools and the progressive coverage are shown in Figure 2-17 and Figure 2-18, respectively. Like open-source software, Humanoid has attained the highest visibility (24.1%) relative to other devices. Due to the complexity of market apps, at the end of testing, coverage for some apps was not converged [34].

This method is reviewed before the start of implementation of this project. It provides a clear idea about how to use machine learning method to test the user interface. But it only focuses on the Android App testing, which means it will be only used on Android platform. This project would like to design a model which could test the user interface across the platform, like testing on Android, IOS, and web browser on mobile device. This method is based on deep learning which is a valuable method and provide precious idea for the design of this project. It gives a clear idea about how to generate human-like inputs based on the visual information in the current UI state and the latest state transitions. The limitations are the inputs are difficult to collect from human interactions. This project will improve this and using object method to implement this.
Chapter 3 Methodology

3.0 METHODOLOGY OVERVIEW

Automated mobile apps user interface testing based on computer vision is basically implemented by two phases. The first phase is the training the model by the dataset, the second phase is tested the results on different platforms. The figure below shows the methodology of the training process. The training dataset are chosen from Rico.

Figure 3-1: Training Phase

The figure below shows the testing steps of this project. Using the collected dataset to test the trained model on the same and different platforms to validate the results. Also, the SIFT similarity comparison method is used to evaluate the results.

Figure 3-2: Testing Phase
3.1 DATASET PRE-PROCESSING

Before the training process, the trained dataset should be pre-processed. The training dataset in this project is chosen as Rico dataset. During the pre-processing period, the UI components are labeled, which is used for trained the model. For example, there are 5 classes in a UI. Then the 5 classes of components will be labeled to their corresponding classes. After that, the animation will be generated. The next step is to transfer the format of this from VOC to YOLO, which are compatible for the format of the input of YOLOV3 model. Then the labels will be generated and then they will be normalized. Thus, the training dataset is ready for the training process. It is notable that the most important part in dataset pre-processing is to label the classes which used in model. The model needs to know the classes that the UI components belong to. Furthermore, the model only supports the format of YOLO, therefore, the format transition also should be considered.

![Figure 3-3: Dataset pre-processing](image)

In the pre-process of the dataset, assuming the upper left corner and lower right corner label of the bounding box are \((x_1, y_1), (x_2, y_2)\). Therefore, the coordinate of x after normalization will be:

- The center point of x coordinate after normalization: \(((x_2 + x_1)/2.0)/w\)
- The center point of y coordinate after normalization: \(((y_2 + y_1)/2.0)/h\)
- The target bounding width after normalization: \((x_2 - x_1)/w\)
- The target bounding height after normalization: \((y_2 - y_1)/h\)
3.2 YOLOV3

Yolov3 is an algorithm using convolutional neural networks to detect objects. Yolov3 is one of the faster object detection algorithms out there. In comparison to recognition algorithms, a detection algorithm not only predicts class labels, but also detects positions of objects [37]. Not only can it classify the image into a category, it can detect multiple objects within an image. This algorithm applies a single neural network to the full image. This network divides the image into regions, predicting boundary boxes and probabilities for each region. The expected probabilities weigh these bounding boxes [38].

3.2.1 Bounding Box Prediction

Yolov3 method predicts the bounding boxes by using dimension clusters as anchor boxes. It predicts 4 coordinates for each bounding box, \( t_x, t_y, t_w, t_h \). The predictions will be done by the top left corner of the image \((C_x, C_y)\) and the width and height of the bounding box \( p_w, p_h \). Yolov3 using logistic regression to predict the objectness score for every bounding box. When the bounding box overlaps the object more than other bounding box prior, the objectness score will be 1 [38].

![Bounding Box with Dimension Priors and Prediction](image)

**Figure 3-4: Bounding box with dimension priors and prediction** [38] [39] It predicts the width and height of the box as offsets from cluster centroids.
3.2.2 Class Prediction

Each bounding box predicts the classes that probably contains multiple labeled classification. Therefore, Yolov3 does not use Softmax as it cannot give a good performance [40]. This model will use the logistic classifiers and the cross-entropy will be decided to use as loss function. Because the real dataset will have many overlapped labels. If using Softmax will lead to exactly one class [38].

3.2.3 Network Architecture

Yolov3 uses a new network for extracting features. The following figure shows the architecture of Yolov3. Compared with 19 convolutional layers, Yolov3 does have 53 convolutional layers, which is more powerful and efficiently [38].

![Figure 3-5: Yolov3 Darknet 53](url)
Yolov3 does have 53 convolutional layers. In addition, Yolov3 uses the strategy which is similar with the multiple scales of SSD. It uses feature map which does have 3 scales (13*13, 26*26, 52*52). The batch normalization also being used. In deep learning neural network, the distribution of each layer should be same. This project uses Yolov3 as the object detection method used in automated apps testing, this is because the UI of the apps does have different size and different components. The UI components does have many classifications, such as buttons, icons, toggles, and tick box. By compared with R-CNN, fast R-CNN, faster R-CNN, and Yolo family. Yolov3 will be more powerful and efficiently to train the Rico dataset and will also be more accurately for the predictions. Furthermore, Yolov3 better utilizes the GPU, which make it more efficient to evaluate and faster.

Figure 3-6: YoloV3 Architecture and Convolutional Set [38] [41]
3.3 SIFT SIMILARITY COMPARISON

The step after the object detection is to compare the similarity of the results in different platform to evaluate the accuracy of this method. SIFT then will be used. SIFT, or Scale Invariant Feature Transformation is a feature detection algorithm in computer vision [42]. SIFT is able to locate the features of an image, which are known as the keypoints of the images. These points are scaled, and rotated invariant thus could be used in computer vision applications. SIFT in this project is used in UI component detection and evaluation. The main benefit of SIFT is that would not be affected by the size or orientation of the image [43].

3.3.1 Keypoints Selection

Before keypoints selection, there are several pre-processing need to implement. First step is to construct the scale space, which need to identify the most distinct feature in a given image and ignoring any noise. SIFT uses Gaussian Blurring technique to reduce the noise in an image. Secondly, SIFT will enhance the feature using a technique called Difference of Gaussian [44]. After that, the next step is to find the important key points from the image that can be used for feature matching. The basic solution is to find the local maxima and minima for the images. Then for the selection of the points, some of them are not robust enough to the noise, therefore, SIFT will eliminate the keypoints that have low contrast, or lie every close to the edge. The figure below will use the example to show the keypoints of an image. The original pictures below are from ANU official website [45].

![SIFT method, the keypoints selection example](image)

Figure 3-7: SIFT method, the keypoints selection example
3.3.2 Feature Matching

SIFT will be used for feature matching in this project. In the automated software app testing, after the object detection. SIFT is expected to be used to evaluate the results. For example, the buttons are tested on different platforms. To ensure the button is detected correctly at the same app for different platforms, SIFT will be used for the feature matching to examine the results. The figures below show the feature matching example by using SIFT. The pictures are chosen the CECS building of ANU. The pictures are from different angles and different size to implement the feature matching.

Figure 3-8: Feature matching examples.
Chapter 4 Evaluation and Results

4.1 EVALUATION

This chapter will introduce the evaluation and results of this model, including the dataset used in the evaluation and the corresponding results. The dataset will be divided into training dataset and testing dataset. The testing UI dataset are collected from different platforms, which are including IOS App, IOS Web, and Android Platform. Furthermore, the widgets that will be used for detecting will be introduced at section 4.1.3. In addition, the hyperparameters set in this project will be provided and explained at section 4.1.4. The last section will introduce the stopping criteria, which used in the process of training the model. The stopping criteria is to be used avoid the overtraining.

4.1.1 Training Dataset

The training dataset is a dataset used for learning. The dataset used for training in this project is chosen from the Rico dataset. The Rico dataset contains more than 66,000 unique UI screen [46]. They are all UI screenshots and view hierarchies. The training dataset will be chosen from them which does have essential UI components, like buttons, toggles and icons. The UI which contains Images will be not considered. For example, the shopping website which shows the demo of the products by using images. This project will work on the functional UI components. The figure below shows the training dataset, which are covered the essential UI components in market apps.

The figure below shows the sample of the training dataset. During the process of choosing mobile UI dataset, the priority is to choose the mobile UI interface which does have many essential UI components. The mobile UI above does have buttons, icons, and tick box. It is expected to choose the training dataset with more UI components within less pictures. This will certainly reduce the pressure of the learning and improves the efficiency. Therefore, the training dataset of this project will be chosen from the variety of mobile UI with different UI components.
Figure 4-1: Training Dataset Example  The dataset is designed and used based the Rico Dataset [46].

4.1.2 Testing Dataset

The aim of this project is to implement this method to use in different platforms. To test the feasibility of this method. The testing UI dataset are chosen from two different platform. The testing dataset are collected from Amazon UI, the first group testing UI dataset are from the application of Amazon based on IOS system (IOS 13.5). The second group of testing UI dataset are from the Web based on the browser of IOS device. All the testing dataset all collected from the same application (Amazon).

Figure 4-2: Two groups of testing dataset (IOS app platform vs IOS web platform).
### 4.1.3 Classes Defined

The table below shows the widgets that will be used in this project. There are total 10 classes or widgets used in this project. The widgets are very common in most of the mobile UI dataset. Therefore, the widgets are selected to be used for the automated testing in this project.

<table>
<thead>
<tr>
<th>Widget/Class</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Home Button</strong></td>
<td>Allow the user to return to the main UI.</td>
<td><img src="image" alt="Home Button" /></td>
</tr>
<tr>
<td><strong>Click Button</strong></td>
<td>Allows the users to forward the next page.</td>
<td><img src="image" alt="Click Button" /></td>
</tr>
<tr>
<td><strong>Shopping Button</strong></td>
<td>Allow the users to shopping by using this button.</td>
<td><img src="image" alt="Shopping Button" /></td>
</tr>
<tr>
<td><strong>Toggle Button</strong></td>
<td>Toggles that does have two states by clicking them.</td>
<td><img src="image" alt="Toggle Button" /></td>
</tr>
<tr>
<td><strong>Menu</strong></td>
<td>The textual button across the bottom of top of UI</td>
<td><img src="image" alt="Menu" /></td>
</tr>
<tr>
<td><strong>Tick Box</strong></td>
<td>The button which the users could select the states by clicking that.</td>
<td><img src="image" alt="Tick Box" /></td>
</tr>
<tr>
<td><strong>Profile</strong></td>
<td>The button which could lead the users to profile.</td>
<td><img src="image" alt="Profile" /></td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>The button that could help users to return to previous page.</td>
<td><img src="image" alt="Return" /></td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>The button that could search the key information on UI.</td>
<td><img src="image" alt="Search" /></td>
</tr>
<tr>
<td><strong>Slider</strong></td>
<td>The button that allows users to adjust the level.</td>
<td><img src="image" alt="Slider" /></td>
</tr>
</tbody>
</table>

*Table 4-1: Widgets that the model can identify in UI. The icons of the buttons were collected form author’s phone.*
4.1.4 Hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>64</td>
</tr>
<tr>
<td>Subdivisions</td>
<td>16</td>
</tr>
<tr>
<td>Max batches</td>
<td>20,000</td>
</tr>
<tr>
<td>Steps</td>
<td>16,000,18,000</td>
</tr>
<tr>
<td>Width/height</td>
<td>416,416</td>
</tr>
<tr>
<td>Classes</td>
<td>10</td>
</tr>
<tr>
<td>Yolo Filters (convolutional)</td>
<td>45</td>
</tr>
<tr>
<td>Gaussian yolo filters</td>
<td>57</td>
</tr>
</tbody>
</table>

Table 4-2: Hyperparameters for the network model

As to the hyperparameters in this model, there are several key hyperparameter need to be set before start training the network. The batch size is a hyperparameter that defines the number of samples to work over before updating the internal parameters. The batch size commonly will be 32, 64, 128. The batch size in this model will be set as 64 and the subdivisions is 16.

The max batch size depends on the number of the classes. \( \text{max batch size} = \text{classes} \times 2000 \). Therefore, the max batch size is set as 20,000. The steps of this model basically will be set 80% and 90% of the max batches, so the steps will be 16,000 and 18,000 respectively. In addition, according to the YOLOV3 rules, \( \text{Filters} = (\text{classes} + 5) \times 3[\text{convolutional}] \), which the number of filters is 45 [47]. When using the Gaussian yolo layers, the \( \text{filters} = (\text{classes} + 9) \times 3 \).

In addition, during the training process, the learning rate will be adjusted with the monitoring. If the results get a Nan, then for some dataset better to decrease the learning rate, the learning rate will be set as \( (0.00261/GPUs) \) [47].
4.1.5 Stopping Criteria

When training this large network, there will be a point during training when the model will stop generalizing and begin to learn [48]. To avoid overtraining, stop training early provide a clear solution to this issue. A large number of training iterations or epochs is used then usual when training the network [49]. This project using the loss as one of the indicators to monitor the performance of this model. The training process for this project basically will spend more than 12 hours, therefore find the early stopping point will be more useful. The figure below shows the loss curve of the training process for this project.

Figure 4-3: The average loss during the training. The loss become stably at around 200.

<table>
<thead>
<tr>
<th>Region Avg:</th>
<th>Class:</th>
<th>Obj:</th>
<th>No Obj:</th>
<th>Avg Recall:</th>
<th>count:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.363987</td>
<td>0.898221</td>
<td>0.890010</td>
<td>0.006567</td>
<td>1.000000</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4-3: The varying indicators of errors

The table above shows the indicators that shows in the training process, when the average loss is no longer decreases at many iterations then the training should be stopped. The final average loss can be from 0.05 to 3.0. Another way is to observe the indicator of mAP(mean average precision), which is better than average loss, so training when mAP increases. To avoid overfitting that could only detect the objects on images from training dataset, the best weights should be got from early stopping point.
4.1.6 Evaluation Metrics

mAP (mean average precision) is commonly used in measuring the accuracy of object detectors. mAP calculates the mean of the average precision value. For this project, mAP is the mean value of average precisions for each class (widget defined before). The mAP is defined as below [50], where $Q$ is the number of queries in the set and $AveP(q)$ is the average precision (AP) for a given query $q$.

$$mAp = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$ (2)

IoU (Intersection over union). IoU measure the overlap between boundaries. IoU is used to measure how much predicted boundary overlaps with the real object boundary. The way to predefine an IoU threshold is to classify the prediction is a true positive or a false positive.

![Figure 4-4: IoU description](image)

Figure 4-4: IoU description

Precision measures how accurate of the predictions (the percentage of the predictions are correct). Recall measures how well to find all the positives [51].

![Figure 4-5: Recall and Precision](image)

Figure 4-5: Recall and Precision
4.2 RESULTS

This section will demonstrate the results of different platform, which are based on Android platform, IOS App platform, and IOS Web platform. In addition, the evaluation metrics will also be given for each result.

4.2.1 Android Platform

The testing UI dataset are chosen from Rico dataset which are not used during the training process. This is only for the initial validation for this model. The training dataset are all chosen from Rico dataset which are based on Android platform. Therefore, the testing dataset for Android platform are randomly chosen from the testing dataset.

![Random test on Android platform by using Rico Dataset](image)

**Figure 4-6: Random test on Android platform by using Rico Dataset**

The testing results are shown above. In general, the essential predefined classes are all detected during the testing process. This project used 20 Android UI dataset to test the model. The evaluation metrics are shown below for the testing. The aim is to test the model initially, which will prove the model is learning well and the weights are optimized. Therefore, the dataset is chosen from the same platform with training dataset. The mAP and precision are relatively high, which proved the model already being trained. Then the next step is to test and validate the model in cross platform, which are IOS app platform and IOS Web platform respectively.
<table>
<thead>
<tr>
<th>mAP threshold</th>
<th>mAP</th>
<th>Average IoU (%)</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.989899</td>
<td>75.60</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>0.75</td>
<td>0.978523</td>
<td>75.67</td>
<td>0.97</td>
<td>0.95</td>
</tr>
<tr>
<td>0.95</td>
<td>0.987597</td>
<td>76.85</td>
<td>1.00</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Table 4-4: Evaluation metrics results for Android platform

4.2.2 IOS App Platform

Figure 4-7: IOS App platform testing
The IOS app UI dataset are collected from the screenshots of Amazon Apps. The testing dataset is to simulate the users’ actions trace. During the trace of each action, the UI dataset has been recorded. This will more valuable for the automated testing of UI. The results show above are the actual predictions made by this model. There are almost 80% of classes are detected by the model. Although the UI widgets of training dataset are different with the testing dataset, the widgets are still detected well. However, there are still some classes not detected by this model.

The table below is the evaluation metrics for this group of datasets.

<table>
<thead>
<tr>
<th>mAP Threshold</th>
<th>mAP</th>
<th>Average IoU (%)</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.789899</td>
<td>66.15</td>
<td>0.85</td>
<td>0.76</td>
</tr>
<tr>
<td>0.75</td>
<td>0.786898</td>
<td>68.94</td>
<td>0.89</td>
<td>0.74</td>
</tr>
<tr>
<td>0.95</td>
<td>0.773564</td>
<td>72.70</td>
<td>0.95</td>
<td>0.65</td>
</tr>
</tbody>
</table>

Table 4-5: The evaluation metrics for IOS App platform

4.2.3 IOS Web Platform

The testing dataset for IOS web platform are collected from the same app with IOS app platform. This is aimed to evaluate the detected results for same app in different platform. Furthermore, the testing dataset for IOS web platform also simulate the trace of the users. For each of the trace, the UI widgets will be detected. The table below shows the evaluation metrics for IOS web platform.

<table>
<thead>
<tr>
<th>mAP Threshold</th>
<th>mAP</th>
<th>Average IoU (%)</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.657256</td>
<td>60.56</td>
<td>0.77</td>
<td>0.67</td>
</tr>
<tr>
<td>0.75</td>
<td>0.662314</td>
<td>61.25</td>
<td>0.81</td>
<td>0.64</td>
</tr>
<tr>
<td>0.95</td>
<td>0.642531</td>
<td>65.88</td>
<td>0.84</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Table 4-6: Evaluation metrics for IOS Web Platform
The figure below shows the prediction results for the IOS web platform. Most of the classes or widgets are detected successfully. The click button and tick box are almost all detected successfully. However, the profile button is detected wrongly, as well as for some of the buttons. The reason that caused the error is the appearances of same class in different platform are not same at all. Another widget that cannot be detected in each dataset is search button. The defined class of search is transparent, the model could not detect the search button when it does have colourful background.

Figure 4-8: IOS web platform testing
4.2.4 Similarity Comparison (SIFT) Results

After the detection of the widgets, this part will demonstrate the results of the comparison of the detected button, tick boxes, and other components. This is aimed to evaluate the detected results further to validate the accuracy of this method used in automated testing of cross-platform. The method of similarity comparison already been introduced in 3.3. The results of comparison between Android, IOS app platform and IOS web platform are shown below.

![Figure 4-9: Similarity comparison](image)

Using SIFT method to compare the similarity of the same class of widgets on different platforms. It is found that the similarity is relatively high although their appearance have some difference. This model still could detect the widget successfully. For example, the upper left figure shows the similarity of two same widgets in different platforms. The length of key points for them are 12 and 18 respectively. The model could detect them as the same class of widgets. The upper right figure shows the different shapes of buttons in different platforms. The length of key points for them are 12 and 27. But the trained model also could detect the class of this widget. The other subplot in figure shows the same class of button in IOS app platform and IOS web platform respectively. The appearances of the widgets are different in various platforms even though for a same app. But this model still could detect that successfully by training.
4.3 DISCUSSION

This project is aimed to use the object detection method in automated app testing. It does show the results of three platforms, one is Android platform which using Rico dataset. The other platforms are iOS app and iOS web. The results show the highest accuracy is the Android platform. This is because the model is trained by Android dataset. The mAP for Android UI dataset testing achieved 0.98, which means it could almost detect all of the widgets in a user interface.

The results of IOS App show the better performance compared with the IOS web. The mAP of IOS App testing dataset achieved 0.79. It means that this model could detect almost 80% classes in this project. However, there are still some classes could not be detected. Besides, the classes detected successfully in some of UI dataset but cannot be detected in another UI dataset. The reason caused this is probably the resolution of the images.

The results of IOS web shows the average performance compared with the other platforms. The highest mAP is 0.65. However, the results still show the model could detect beyond 50% of classes in this project. It is notable that the icon and buttons are not same in different platforms. This is also the reason that caused the errors occurred. The results of SIFT shows the similarity of widgets in these platforms. The similarity of most detected classes is highly matched. It means the detected results have a better performance.

In short, the results prove the feasibility that object detection method used in automated app user interface testing. Using object detection method in automated UI testing will be more effectively.
5.1 CONCLUSIONS

Automated UI testing can handle repetitive, time-consuming tests and find the errors compared with manual testing. It gives the software developer and software tester a unique advantage of quick, hassle-free testing and an excellent way to save precious resources. Object detection is a computer vision method used in the actual environment. This project applies this method in the automated mobile user interface testing. In addition, to validates the feasibility of this method, this project testing object detection method in different platforms, which are Android, IOS App, and IOS Web. These operating systems are the most common systems used by users.

This project simulates the user environment to test the UI dataset by using this method. First, using the UI dataset from Android platform to test this model. Because the model is trained by using the dataset from Android platform, the performance is the best of all the dataset, achieving the highest accuracy. Second, the model is tested by the dataset from IOS App platform. Although the performance is worse than Android, almost beyond 80% of widgets are detected successfully. This proves this method also could be used on different platform, although some widgets do have different appearances. Third, the model is being tested by the dataset from IOS web platform, detecting around 70% widgets. The performance is worst but most of the widgets could be detected. Finally, the Similarity comparison also be used to evaluate the feature matching of these widgets. It shows that the widgets that detected in different platform are relatively matched.

However, there are still has some limitations in this project. Some of the widgets could be detected in some UI dataset but cannot be detected in a different UI dataset. There are some errors happened when detected widgets, such as detected wrongly. In future, these limitations will be optimized.
5.2 FURTHER WORK

After considering the range of this project and the results obtained accordingly, a few aspects are identified that can be revisited in future work.

1) More widgets will be added in the pre-defined classes, there are many widgets in the actual mobile user interface.
2) Test this approach on platforms like Windows Phone, BlackBerry and Symbian.
3) Using the UI dataset from different platforms to train model to improve prediction performance.
4) In future, the YOLOV3 structure needs to be optimised to improve predictive efficiency.
5) The configuration environment is complex, it needs a particular setting. Future world mode should be simpler.
6) In the future, different object detection methods will be compared. For example, R-CNN, fast R-CNN, and faster R-CNN. Future analyses will be performed to compare the difference and analyses efficiency.
7) To compare the difference among YOLO family, and to prove which method could achieve a better result. To implement the automated testing using YOLO lite which does not need GPU. Then comparing the difference of using GPU.
8) For comparison of similarities, a further optimised algorithm will be implemented which could be easier to compare.
Chapter 6 Bibliography


[42] "Introduction to SIFT (Scale-Invariant Feature Transform)," OpenCv, [Online]. Available: https://opencv-python-


Chapter 7 Appendix A Detailing Tasks and Expected Outcome

The project is approved by the supervisor Associate Pro. Zhenchang Xing. The project is to explore a new method for the mobile user interface automated testing based on computer vision method. The expected outcome is to complete the widgets detection in different mobile platforms. The final results show that this project achieved this outcome.

The first stage of this project is to analyse the technology or method used on the automated testing of mobile user interface. After a lot of literature review, it is decided to use the YOLOV3 which is a new method for the object detection.

The second stage of the project is the dataset collected and dataset pre-processing. This project collected the dataset from different platforms. For the dataset, the training dataset are collected from Rico dataset, which is used for machine learning and data mining. The training dataset are the user interface from Android platform.

The third stage of this project is to implement the object detection model. It is essential to configure the environment for this model before implement the model. After that, the task is to analyse the architecture of this model and how it runs to do the object detection.

The fourth stage of this project is the evaluation the results. Monitoring the training process is an important task for this learning activity. Stopping criteria provided a clear solution for this training process to decide which iteration or batches should be stopped to avoid overfitting.

The last stage of this project is to validate this model in different platforms and evaluate the results. During this period, SIFT method and evaluation metrics are used to evaluate the accuracy of the results. The results show that this model used in this project achieved the good performance on different platforms. In addition, the limitation and future works are also discussed in this thesis.
INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the Honours/Projects co-ordinator

SECTION A (Students and Supervisors)

<table>
<thead>
<tr>
<th>UnitID:</th>
<th>V5687862</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY NAME:</td>
<td>Zhang</td>
</tr>
<tr>
<td>PERSONAL NAME(S):</td>
<td>Weinan</td>
</tr>
<tr>
<td>PROJECT SUPERVISOR (may be external):</td>
<td>Dr. Zhenchang Xing</td>
</tr>
<tr>
<td>COURSE SUPERVISOR (an RSCS academic):</td>
<td>Dr. Zhenchang Xing</td>
</tr>
<tr>
<td>COURSE CODE, TITLE AND UNIT:</td>
<td>COMP 8755 Individual Computing Project (12 crs)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEMESTER</th>
<th>☑ S1</th>
<th>☑ S2</th>
<th>YEAR: 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECT TITLE:</td>
<td>Automated Mobile UI Testing of Cross-Platform Based on Computer Vision</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LEARNING OBJECTIVES:

- Using Computer-Vision Method to Implement Software Testing (UT testing)
- Cross platform testing, implement action.

PROJECT DESCRIPTION:

* Write a literature survey

The project is based on Computer-Vision method to implement UI mobile testing for different platforms, such as Android platform, iOS App platform, and iOS Web platform. By using the object detection method to reorganize the widget (the most common used) in the UI (mobile), it will provide the current tool for developers to test during the testing period.
### ASSESSMENT (as per the project course's rules web page, with any differences noted below)

<table>
<thead>
<tr>
<th>Assessed project components:</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report style: Report</td>
<td>(min 45, def 60)</td>
<td>45%</td>
<td>(examiner)</td>
</tr>
<tr>
<td>(e.g. research report, software description...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstract: kind: Software/Dataset</td>
<td>(max 45, def 30)</td>
<td>45%</td>
<td>(supervisor)</td>
</tr>
<tr>
<td>(e.g. software, user interface, robot...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td>(10)</td>
<td></td>
<td>(course convenor)</td>
</tr>
</tbody>
</table>

### MEETING DATES (IF KNOWN):


### STUDENT DECLARATION: I agree to fulfil the above defined contract:

Signature: ___________________________ Date: ____________

### SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following examiner, and have obtained their consent to review the report (via signature below or attached email):

Signature: ___________________________ Date: ____________

Examiner: ___________________________ Signature: ___________________________

(Nominated examiners may be subject to change on request by the supervisor or course convenor)

### REQUIRED DEPARTMENT RESOURCES:


### SECTION C (Course convenor approval)

Signature: ___________________________ Date: ____________

Research School of Computer Science

Form updated: Jun 2018
Chapter 9 Appendix C Description of Artefacts

List of all program code files

- Dataset_preprocessing
- SIFT(similarity_companion)
- Training_dataset
- Validation_dataset
- YoloV3_model
- Coding_References.txt
- Command.txt
- ReadMe.txt

Figure 9-1: The files of artefacts

- Dataset pre-processing is implemented by the author of this thesis (Weinan Zhang).
- SIFT (similarity comparison method) is modified based on the references.
- Training dataset is collected from Rico dataset.
- Validation dataset is collected by the author (Weinan Zhang).
- YOLOV3 model is modified and implemented based on the original model of YOLOV3, the references are providing in the file.
- The configuration hyperparameters, the weights, and all the network are implemented by the author of this thesis (Weinan Zhang).

Details of the code was tested for correctness

This project is aimed to implement the mobile user interface automated testing. Therefore, the testing dataset are used instead of the code. The model will be improved with the results of the testing dataset. The testing dataset is collected from Android platform, IOS APP platform, and IOS WEB platform. The figures below show the testing dataset which collected by author. The dataset is the mobile user interface collected from Amazon app on different platforms.
Figure 9-2: Testing dataset

Experiment: Environment configuration

GPU

<table>
<thead>
<tr>
<th>Activity</th>
<th>Dedicated GPU Memory Usage</th>
<th>Shared GPU Memory Usage</th>
<th>Driver Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>4.0 GB</td>
<td>7.9 GB</td>
<td>26.21.14.4/45</td>
</tr>
<tr>
<td>Video Encode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Decode</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This experiment is based on NVIDIA GeForce GTX 1650. The training needs high GPU. The GPU for this experiment is 12GB. But it still needs beyond 24 hours to be trained. The environment components are listed as:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Visual Studio 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Application</td>
<td>CUDA Toolkit 11.0 NVIDIA</td>
</tr>
<tr>
<td>Accelerator</td>
<td>NVIDIA cuDNN</td>
</tr>
<tr>
<td>Driver</td>
<td>NVIDIA® GPU drivers</td>
</tr>
<tr>
<td>OpenCV</td>
<td>OpenCV – 4.2.0</td>
</tr>
<tr>
<td>TensorRT</td>
<td>7.0.0</td>
</tr>
<tr>
<td>Environment</td>
<td>Windows</td>
</tr>
</tbody>
</table>

**Table 9-1: Software requirements**

The experiment is conducted by using the training dataset to train the model. The training dataset is pre-processed to satisfy the requirements of the YOLOV3 input. The batch size set as 64 and the subdivisions set as 16. During the training process, the weights will be generated after each 100 iterations. At the first several hundred iterations, the weights are not optimized enough. After training 500 iterations, the loss decreased and mAP increased, then the best weight is generated. The figure below shows the weights generated for this project. Using the generated weights to test the testing dataset to validate the performance of this model.

**Figure 9-4: The weights attained during the training**
Chapter 10 Appendix D ReadMe in Artefacts

**ReadMe.txt:**

1. *Data_pre-processing* is used for the pre-processing of the dataset, also including for the validation dataset.

2. *Training_dataset* is including the dataset used for training, this file includes the raw dataset and the dataset after pre-processing.

3. *Validation dataset* is the dataset used for validating the model, it includes the raw dataset and the dataset after pre-processing.

4. *YoloV3 model* is the implementation of the neural network model. The main configuration parameters are in *YoloV3_model\build\darknet\x64*

4.1 *The training log* is saved in *YoloV3_model\build\darknet\x64\log*, which used to record the whole process of training.

4.2 *The weights* are all stored in *YoloV3_model\build\darknet\x64\weights*, which have the weights of each iteration.

4.3 Predictions are shown in *YoloV3_model\build\darknet\x64*. The results are saved.

5. *SIFT* is the method used for similarity comparison, which does have 2 files, one is comparing the keypoints, another one is for feature matching.

6. The *Command.txt* is the command on Windows system used for training, testing, validating, and calculating.

7. All the references are saved in *Coding_References.txt*