國立臺灣大學電機資訊學院電機工程學研究所

碩士論文

Graduate Institute of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

利用影像分類技術識別網頁及手機程式中之圖形介面元件

Using Image Classification Techniques to Recognize GUI Elements in Web and Mobile Applications

林其政

Qi-Zheng Lin

指導教授：王凡 博士

Advisor: Farn Wang, Ph.D.

中華民國 108 年 7 月

July, 2019

doi:10.6342/NTU201903613

誌謝

能完成這篇論文，首先要感謝我的指導教授王凡老師。在做研究的路上迷惘的時候，是老師為我指引一條明路，讓我找到前進的方向。不單是學術上，老師更教會了我許多做人處事的道理，著實令我獲益良多。

也要感謝實驗室的每一位成員。在學業上，我們互相幫忙、彼此照應，順利地克服了各種困難。而當被學業的壓力壓得喘不過氣時，只要大家聚在一起吃喝玩樂，享受片刻的閒暇，就能重新振作起來，繼續面對挑戰。

最後要謝謝我的父母還有盈茜，你們的支持與鼓勵一直都我前進的最大動力。

doi:10.6342/NTU201903613

中文摘要

隨著網路與智慧型手機的普及，網頁及手機應用程式如雨後春筍般出現。若是一個應用程式在功能上或性能上有缺陷的情況下上架，它很快就會在市場中消逝，並造成開法者巨大的損失。因此，如何對網頁及手機應用程式進行有效的測試成為一個至關重要的議題。

當進行圖形介面程式的測試時，我們需要一個爬蟲來盡可能攫取圖形介面中的資訊，藉以設計相應的測試腳本。在圖形介面程式中，通常包含文字與圖像的部分。對於圖像的部分，我們難以單憑網路爬蟲解析其本質。然而，藉由圖像分類的技術，我們可以識別圖像所代表的意義。這麼一來，我們便能完整且準確的理解整個圖形介面的內容，以產生適合的測試腳本。

在這篇論文中，我們蒐集了 31490 張應用程式中常見的圖示，包含 57 種不同的種類，並利用卷積神經網路的技術在此資料集上訓練出圖像分類的模型。我們採用了適當的資料擴增方法，使得在預測現實世界程式中的的圖像時能達很高的準確度。此外，我們建立了一個可以持續更新模型的架構。

關鍵字：軟體測試、圖像分類、卷積神經網路、主從式架構

doi:10.6342/NTU201903613

ABSTRACT

As the internet and smart phones become more and more common, plentiful web and mobile applications show up. If an application is published with even some tiny flaws in functionality or performance, it will fade away in the market rapidly, and the developers will suffer tremendous losses. Consequently, how to test web and mobile applications effectively and efficiently become an important issue.

When conducting software testing for GUI applications, we need a crawler to grab the information of the GUI contents as much as possible in order to devise test scripts accordingly. It is general that there are both text contents and image contents in GUI applications. For the image contents, we may not always resolve the essence of them only by a crawler. However, we can recognize the meaning of image contents with the aid of image classification techniques. By doing so, we can understand the whole GUI contents thoroughly and accurately and generate suitable test scripts.

In this thesis, we collect 31490 images of 57 different classes commonly seen in real applications and use CNN to train a model to classify the images. We adopt some appropriate methods of data augmentation to reach high accuracy of predicting image contents in real-world applications. Besides, we build a framework update our model continuously.

Keywords: software testing, image classification, convolutional neural network, client-server model

doi:10.6342/NTU201903613

CONTENTS

doi:10.6342/NTU201903613

doi:10.6342/NTU201903613

LIST OF FIGURES

doi:10.6342/NTU201903613

LIST OF TABLES

doi:10.6342/NTU201903613

Chapter 1 Introduction

1.1 Background

As the internet and smart phones become more and more common, plentiful web and mobile applications show up. Developers faced with fierce competition are required to develop high-quality applications faster than their competitors do. If an application is published with even some tiny flaws in functionality or performance, it can still reduce the will of users to use it because there are so many substitutes for it. As a result, the application will fade away in the market rapidly, and the developers will suffer tremendous losses. Consequently, how to test web and mobile applications effectively and efficiently become an important issue.

Fig. 1.1 is the standard test flow chart. We can see that software testing usually contains the following steps:

1. Test plan: According to the specification about the requirements for functionality or performance of the products provided by the users, define corresponding test requirements. Meanwhile, arrange appropriate manpower, time, and resources.
2. Test design: Construct test procedures that can meet the test requirements defined in the test plan. Design appropriate test cases, including test input, test conditions, and expected results.
3. Test development: Implement reusable automated test procedures.
4. Test execution: Execute the test procedures and compare the expected results with the actual ones.
5. Test evaluation: Generate test reports to show the results in an organized manner. Evaluate the quality of the software under test (SUT).

doi:10.6342/NTU201903613

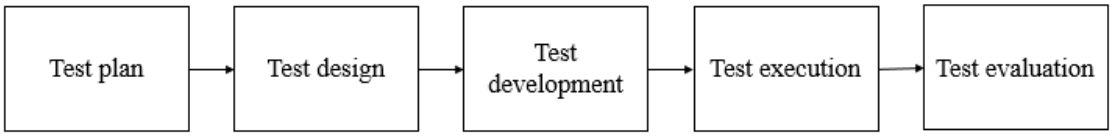


Fig. 1.1 Standard test flow chart

1.2 Motivation

While the concept depicted in Fig. 1.1 is simple, to test a GUI application is quite the opposite. When conducting software testing for GUI applications, we need a crawler to grab the information of the GUI contents as much as possible in order to devise test scripts accordingly. Although there are many helpful tools to do the task, obstruction still exists.

Fig. 1.2 is part of the screenshot of "Google News" website. As illustrated in it, there are icons that present corresponding meaning for every item, e.g., a magnifying glass icon usually symbolizes "search". It is general that there are both text contents and image contents in GUI applications.

For the image contents, we may not always resolve the essence of them only by a crawler, i.e., simply inferring the meaning of an image component by its id, name, and other attributes might not success.

doi:10.6342/NTU201903613

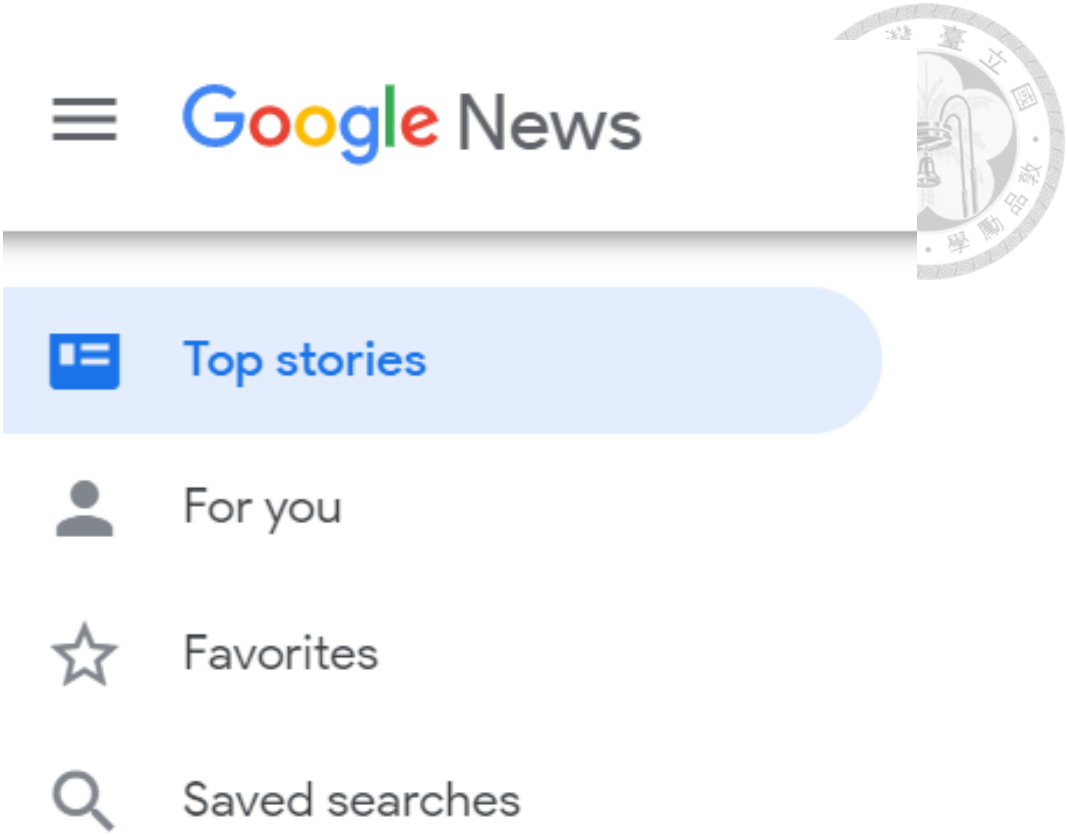


Fig. 1.2 Part of screenshot of the "Google News" website

However, for icons that aim at presenting similar concepts, they are often in similar shapes. Fig. 1.3 shows "user icons" from different applications, including "Google News", "MoPTT", "reddit", and "YouTube". As shown in Fig. 1.3, icons that represent the concept of user are usually in the shape of a human's outline. Thus, we can recognize the meaning of icons with the aid of image classification techniques. By doing so, we can understand the whole GUI contents not only thoroughly but also accurately and generate suitable test scripts.

doi:10.6342/NTU201903613

Fig. 1.3 "User icons" from different applications

1.3 Contribution

In this thesis, we have the following contribution:

- Figure out appropriate parameters to build and train a neural network to recognize GUI elements in web and mobile applications
- Apply suitable data augmentation methods so the model can recognize GUI elements in real-world applications.
- Establish a framework to update the model continually.

doi:10.6342/NTU201903613

Chapter 2 Preliminaries

2.1 Crawler

A crawler, which is also known as a "spider" or a "bot", is a computer program that automatically browses documents on the World Wide Web. Crawlers are primarily programmed to do repetitive actions so browsing is automated. Search engines use crawlers most frequently to browse the internet and build an index. Other crawlers search different types of information such as RSS feeds and email addresses. The term crawler comes from the first search engine on the Internet: the Web Crawler.

Crawlers consume resources on visited systems and often visit sites without approval. Issues of schedule, load, and "politeness" come into play when large collections of pages are accessed. There are mechanisms for public sites not wishing to be crawled to make this known to the crawling agent. For example, including a robots.txt file can request bots to index only parts of a website, or nothing at all.

The number of Internet pages is very large; it is hard to make a complete index even for the largest crawlers. For this reason, search engines strived to give relevant search results in the early years of the World Wide Web, before 2000. Today, relevant results can be given almost instantly.

2.2 Convolutional Neural Network

In 1998, Yann LeCun et al proposed the very first notable Convolutional Neural Network (CNN) model, LeNet-5 [1]. Although CNN was limited because of the insufficient computing power in the following years, it made another breakthrough in 2012 that AlexNet won the champion on ImageNet Large Scale Visual Recognition Competition with top-5 error of 15.3%, outperforming the second place. Nowadays, CNN

doi:10.6342/NTU201903613

becomes the most useful technique when dealing with tasks related to image processing and computer vision.

CNN contains the following stages:

1. Convolutional layers: Convolutional layers use several filters to do convolution operations on the images. This trick has two characteristics: locally connected and weight sharing. Locally connected means it can help extract the high-level features in the image locally. In this stage, filters with the size smaller than the images are used. Next, filters move with the size of stride and do convolutional operation with the corresponding area. Weight sharing means every part of the image shares the same parameter, which help reduce the computation complexity.

$$y_{m,n} = \sum_{i=-k}^{k} \sum_{j=k}^{k} h_{i,j} x_{m-i,n-j}$$

2. Pooling layers: Convolutional neural networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2×2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.
3. Fully connected layers: Finally, fully connected layers with output size equal to the classes are used. This stage use features extracted in previous hidden layers to do the final prediction.

doi:10.6342/NTU201903613

2.3 Online Learning

Online learning is a machine learning method in which the data is provided in a sequential manner and is used to update the best model for future data. Some techniques are designed to learn linear models, such as Perceptron [2], Online Gradient Descent [3], Passive Aggressive (PA) algorithms [4], Confidence-Weighted (CW) algorithms [5], etc. As for nonlinear models, online learning with kernels can do the trick. These methods received substantial interest from the community, and models of higher capacity such as online multiple kernel learning were developed. While these models learn nonlinearity, they are still shallow. Moreover, deciding the number and type of kernels is nontrivial; and these methods are not explicitly designed to learn a feature representation.

Online learning can be directly applied to DNNs ("online backpropagation") but they suffer from convergence issues, such as vanishing gradient and diminishing feature reuse. Moreover, the optimal depth to be used for the network is usually unknown, and cannot be validated easily in the online setting. Further, networks of different depth would be suitable for different number of instances to be processed, e.g., for a small number of instances, shallow networks that can end up a quick convergence would be preferred, whereas, for a large number of instances, the long run performance would be enhanced by using a deeper network. This makes model architecture selection very challenging. There have been attempts at making deep learning compatible with online learning. However, that they operate via a sliding window approach with a mini-batch training stage make them unsuitable for a streaming data setting.

doi:10.6342/NTU201903613

Chapter 3 Related Work

3.1 Sikuli

Sikuli [6] was started in 2009 as an open-source research project in the User Interface Design Group at MIT. It uses image recognition techniques powered by OpenCV to identify GUI elements and provides a visual scripting API to operate GUI elements automatically. It shows a brand-new way to automate software testing for GUI applications.

When generating a test script by Sikuli, one can take a screenshot of the GUI elements he wants to manipulates, and then the screenshot can be the target of a command in the test script. Usable commands include click, hover, type, etc. If he runs the script, Sikuli will search the target in the screen by image recognition and execute the command he has ordered.

The most important revolution Sikuli achieved is readability and usability. Sikuli makes writing programs more easily. It puts the screenshots in the code so people can see what they want to control directly. If one wants to execute some mass and regular actions with a computer, a programmer may use a command line tool or write a program to call the API functions, but an ordinary person would be unable to do this. With the help of Sikuli, people communicate with applications written by others no longer need to read the obscure documents, nor do they have to understand what the underlying architecture is. Combining usual ways to use a mouse and a keyboard with screenshots, one can easily write a program that can be executed automatically.

Sikuli supports different languages and platforms. It supports scripting languages including Python, Ruby, JavaScript, etc. One can programmatically control a web page,

doi:10.6342/NTU201903613

a desktop application running on Windows/Linux/Mac OS X, or even an iPhone or Android application running in an emulator.

3.2 Crawljax

Crawljax [7] is a well-known tool for crawling and testing modern web applications. It can explore any JavaScript-based Ajax web application through an event-driven dynamic crawling engine. It automatically produces a state-flow graph of the dynamic DOM states and the event-based transition between them. This inferred state-flow graph is powerful to facilitate the automation of many types of web analysis and testing techniques.

However, it is still weak when bumping into the complex page. The main purpose of Crawljax is to completely crawl through the pages. After that, nothing will be done further. It will be a disaster if the crawled website page was designed badly or had undergone code compression. Both of the cases are hard for human to read through, not to mention making pattern analysis.

3.3 Tesseract

Tesseract is an Optical Character Recognition (COR) engine. That is, it can recognize the text embedded in images. It was originally developed at HP Laboratories in 1985. It was one of the top 3 engines in the 1995 UNLV Accuracy test. Between 1995 and 2006 it had little work done on it, but since then it has been improved extensively by Google. The early versions of Tesseract could only accept TIFF images. Combined with the Leptonica library now it can read a wide variety of image formats. The count of languages it can recognize is over 100.

doi:10.6342/NTU201903613

3.4 Academic Work

C.-H. Yu proposed Smart-Eye, a visualized service for web and mobile applications [8]. By utilizing image classification, it helps software tester organize and label data for the later testing scripts. Yu used RGB images as training data and applied a 6-layer convolutional neural network. The resulting accuracy of validation was 82.4%.

C.-L. Li incorporated image classification with natural language processing (NLP) to identify the topic of a GUI element by its attribute, including text and image [9]. Li used residual network (resnet18 and resnet34) to perform image classification. As for NPL, Li employed BoW, tf-idf, and LSI to transform words to vectors, and computed the cosine similarity to the vectors of the corpus to determine the topic of an element.

doi:10.6342/NTU201903613

Chapter 4 IconNet

4.1 Overview

We build and train a CNN model to recognize GUI elements. We first present an overview of the whole procedures in this section and describe the implementation details in the following sections.

As shown in Fig. 4.1, we first split the dataset into training data and validation data. We use the training data to train for 100 epochs. Whenever an epoch of training is done, we use validation data to see whether the validation accuracy is improved. If it is, we save it as the best model.

doi:10.6342/NTU201903613

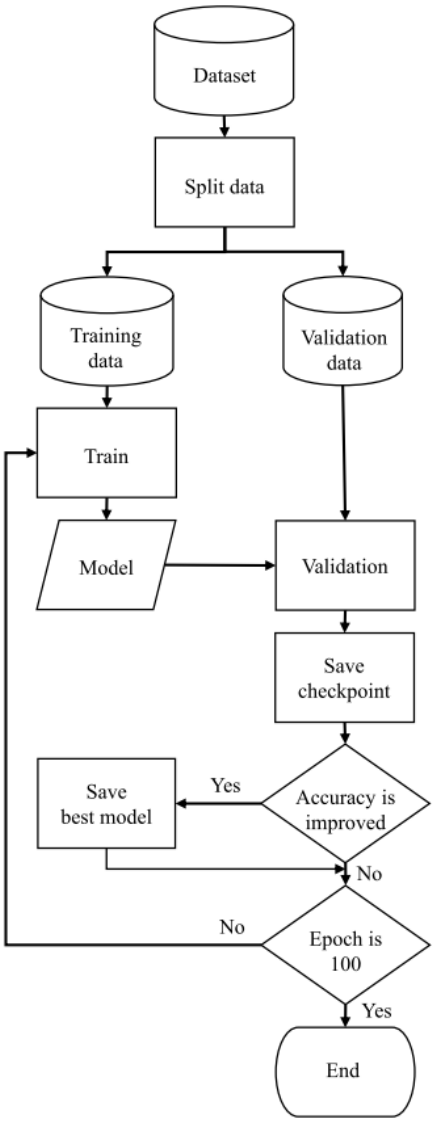


Fig. 4.1 Flow chart of our work

4.2 Data Preprocessing

Grayscale

We always read images in grayscale in our whole work, including training and predicting. Actually, the images in the data we use are all grayscale, even most of which are binary. Thus, the converting procedure will actually has effect only when predicting RGB images.

doi:10.6342/NTU201903613

4.3 Data Augmentation

When undertaking a task related to deep learning, researchers are often faced with lack of data, which would result in overfitting. Data augmentation is the easiest and most common method to solve this problem. By modifying the original data, more data different from the original ones can be generated.

4.3.1 Random Resized Cropping

This is a commonly used method for data augmentation. It randomly resizes the original image with aspect ratio between 0.75 to 1.33 and crop part of it to generate more data. Although the image is change, it would still be acceptable to belong to the original class.

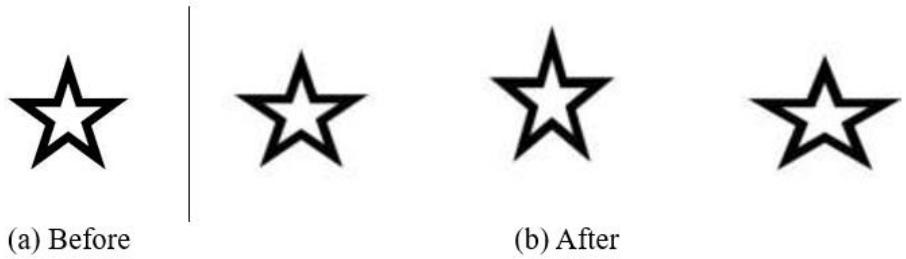


Fig. 4.2 Effect of random resized cropping on an image

4.3.2 Random Color Inverting

In the dataset we use, images are mostly with white background. Nonetheless, there are sometimes icons with darker color as background in real-world. Fig. 4.3 shows a screenshot of "MoPTT" application and grayscale version of it. We can see that the background is darker than the foreground. It would be impracticable if we want to predict this kind of images with a model trained by only images with white background. Hence, we randomly invert the color of training data with probability of 0.5, hoping that our model can predict images with either lighter color or darker color as background.

doi:10.6342/NTU201903613

(a) Original (b) Grayscale

Fig. 4.3 Screenshot of "MoPTT"

4.3.3 Color Jittering

Color jittering is to change the brightness, contrast, saturation, and hue of an image randomly. Because the images we use are binary, we only change contrast and brightness in this stage. Changing saturation and hue of binary images are useless. For similar reason we mention above, we want our data to be as diverse as possible to simulate the real-world data. Most images in the dataset we use are binary. Even with color invert, there will be only images with white and black. Thus, we need color jitter to generate images with different contrast. Fig. 4.4 shows some images after color jittering.

doi:10.6342/NTU201903613

Fig. 4.4 Effect of color jittering on an image

4.4 Network Architecture

Layer Name	Output Size	8-layer	14-layer	20-layer
conv1	30×30	3×3, 64	3×3, 64	3×3, 64
conv2_x	15×15	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times1$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$
conv3_x	8×8	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times1$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$
conv4_x	4×4	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times1$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$
	1×1	average pool, 57-d fc		

Table. 4.1Network architecture

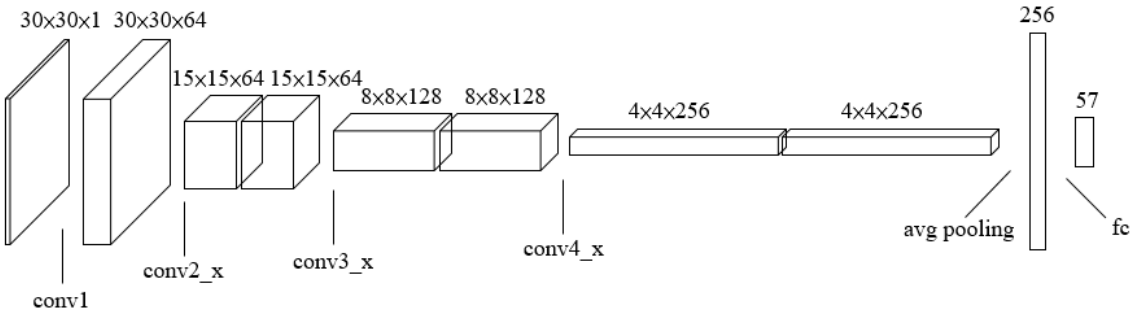


Fig. 4.5 Illustration of network architecture

We first use a convolutional layer with output channel size equal to 64. The following are three more blocks of convolutional layers and we set the stride to be 2 to reduce the output size by half and double the channels. The last layer is fully connected with output size equal to the number of classes we want to predict. All layers except the last use ReLU activation.

doi:10.6342/NTU201903613

4.5 Implementation

In the stage of random resized cropping, we mainly resize the images to 30×30. We also conduct the experiment that compares the performance of different image size, including 8×8, 15×15, and 60×60. We set batch size to 256. We use Stochastic Gradient Descent as the optimization algorithm and cross entropy as the loss function. The learning rate starts from 0.1 and is divided by 10 for every 30 epochs. The models are trained for up to 100 epochs. We use a weight decay of 0.0001 and a momentum of 0.9.

Algorithm 3: train(*training_data*, *model*)

for *input*, *label* **in** *training_data*:

input ← data_augmentation(*input*)

output ← model(*input*)

losses ← loss_function(*output*, *label*)

gradients ← backpropogation(*loss*)

model.weights ← stochastic_gradient_descent(*model.weights*, *gradients*)

end for

doi:10.6342/NTU201903613

Algorithm 4: main(*data_directory*)

training_data, validation_data ← split(*data_directory*)

model ← initialize_model()

learning_rate ← 0.1

best_accuracy ← 0

for *epoch* ← 1 **to** 100

  if *epoch* mod 30 == 0

    learning_rate ← *learning_rate* / 10

  end if

  train(*training_data*, *model*)

  accuracy ← validation(*validation_data*, *model*)

  is_best ← *accuracy* > *best_accuracy*

  best_accuracy ← max(*accuracy*, *best_accuracy*)

  save_checkpoint(*model*)

  if *is_best*:

    save_best_model(*model*)

  end if

end for

doi:10.6342/NTU201903613

Chapter 5 Online Learning based on Client-Server Model

We build an online learning framework based on client-server model for users to use our model and update it. Users can get our model from the server and use it to predict the images appear in their SUT. If they think the predictions are incorrect, they can send back the images along with the corresponding labels that they think are correct. After theses data are checked by the inspector, the server can retrain the model.

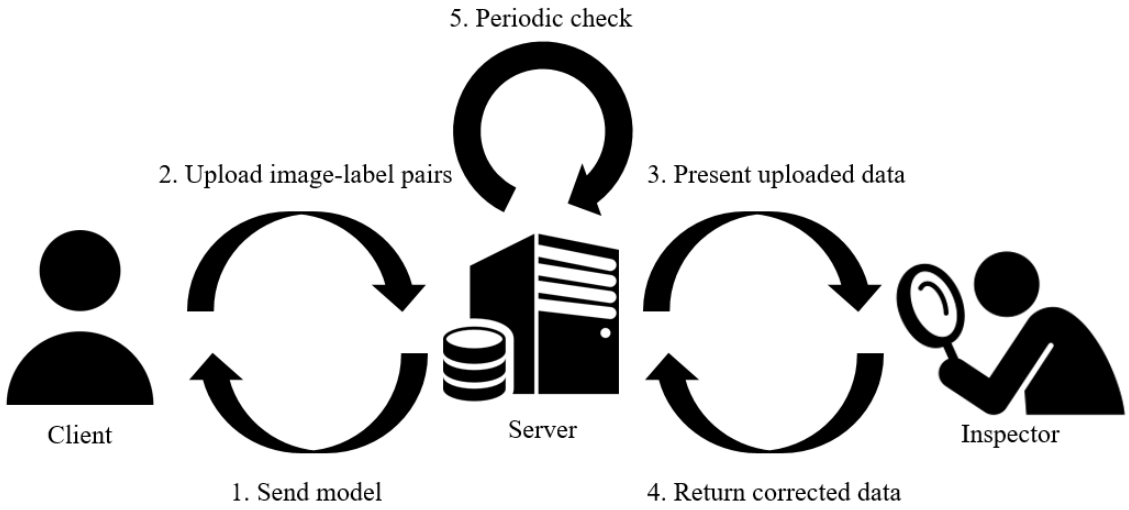


Fig. 5.1 Online learning framework

This framework contains the following use cases:

1. Send model:
2. Upload image-label pairs
3. Reviewing labeled examples via web pages.
4. Periodic training of new models.
    (1) Training new models when there are no new labels, including refining models and retraining completely new models.

doi:10.6342/NTU201903613

(2) Training new models when there are new labels.

5. Get online new models.

5.1.1 Server

We set up a web server with Flask, which is a micro web framework written in Python. The server can respond to two kinds to requests: "get model" and "upload data". For "get model", the server sends back the latest model to the user who makes the request. For "upload data", the server will receive an image-label pair from the client and then save the image to the corresponding directory under the "upload" directory according to the label. We run a thread that check whether there are images in the "upload" directory at 0 o’clock every day. If there are, the server will train the model with the images in the "upload" directory and then move them to the "trained" directory.

doi:10.6342/NTU201903613

Algorithm 5: Server

```
function get_model():
    return model
function upload_data(image, label):
    save image to upload/label/
function check():
    start a thread that calls check() after 86400 seconds
    if upload/ is not empty:
        train the model with the images in upload/
function main():
    compute the remaining time t to 0 o'clock
    start a thread that calls check() after t seconds
```

5.1.2 Client

The client can predict images in his SUT with the model received from the server. After he uses the model to predict the images, a report page will show up. This page contains three columns: "image", "prediction", and "check". As the names suggest, the columns of "image" and "prediction" show the images and the corresponding predictions given by the model. The column of "check" is for the user to check whether the predictions are correct. If he thinks the predictions are incorrect, he can click the "Wrong" radio button and select correct labels with the drop-down list.

After the user checks all the results, he can click the submit button to submit the image that are marked as "Wrong" along with the corresponding labels to the server. What will happen next has mentioned in the previous section.

doi:10.6342/NTU201903613

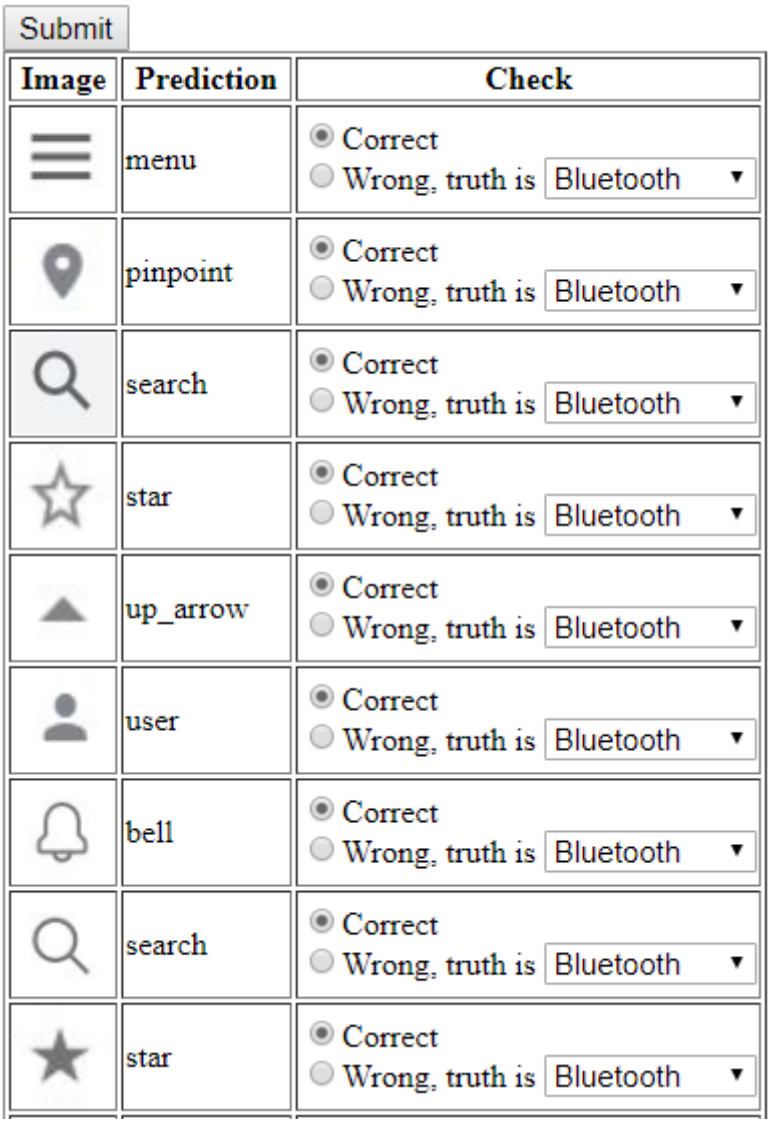


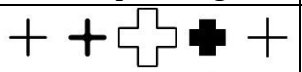
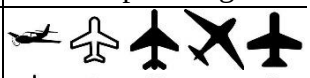
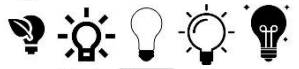
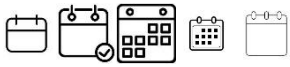
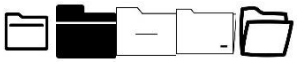
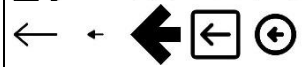
Fig. 5.2 Report page

doi:10.6342/NTU201903613

Chapter 6 Experiments

6.1 Dataset

We use the dataset "Common Mobile/Web App Icons" from kaggle and rearrange 31490 images of 57 different classes from it. Each of them has the size of 200×200. Some sample images of every class are shown in Table 4.1. We further split the data into training and validation data randomly. Training data contains 23639 images and validation data contains 7851 images.

Class	Example Images	Class	Example Images
add		airplane	
battery		bell	
Bluetooth		bulb	
calculator		calendar	
call		camera	
car		cart	
clock		closed_lock	
cloud		coin	
copy		crop	
cut		delete	
document		down_arrow	
email		eye	
Facebook		fire	
flag		folder	
gift		heart	
help		home	
info		left_arrow	

doi:10.6342/NTU201903613

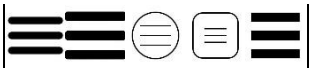
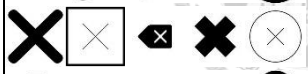
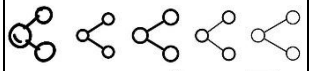
menu		microphone	
moon		music	
mute		no	
open_lock		pinpoint	
play		refresh	
right_arrow		search	
send		settings	
share		star	
tag		thumbs_up	
trophy		up_arrow	
user		warning	
yes			

Table. 6.1Some sample images of the dataset

We also collect 53 images in real-world applications including "Google News", "Medium", "MoPTT", "reddit", "露天拍賣", and "YouTube" as test data to see whether our model can recognize icons in real-world applications correctly.

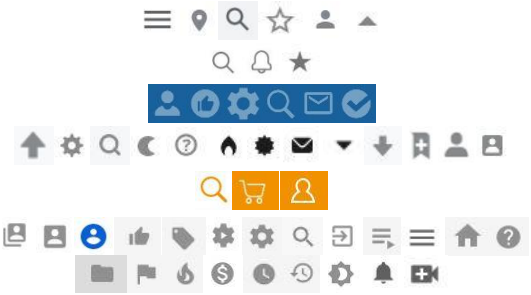
Application	Images
Google News Medium MoPTT reddit 露天拍賣 YouTube	

Table. 6.2Test data

6.2 **Results**

Fig. 6.1 shows the validation accuracy of different network architecture. We can see that all of them would converge around 40 epoch. The accuracy of 8-layer network is

doi:10.6342/NTU201903613

slightly lower than the others.

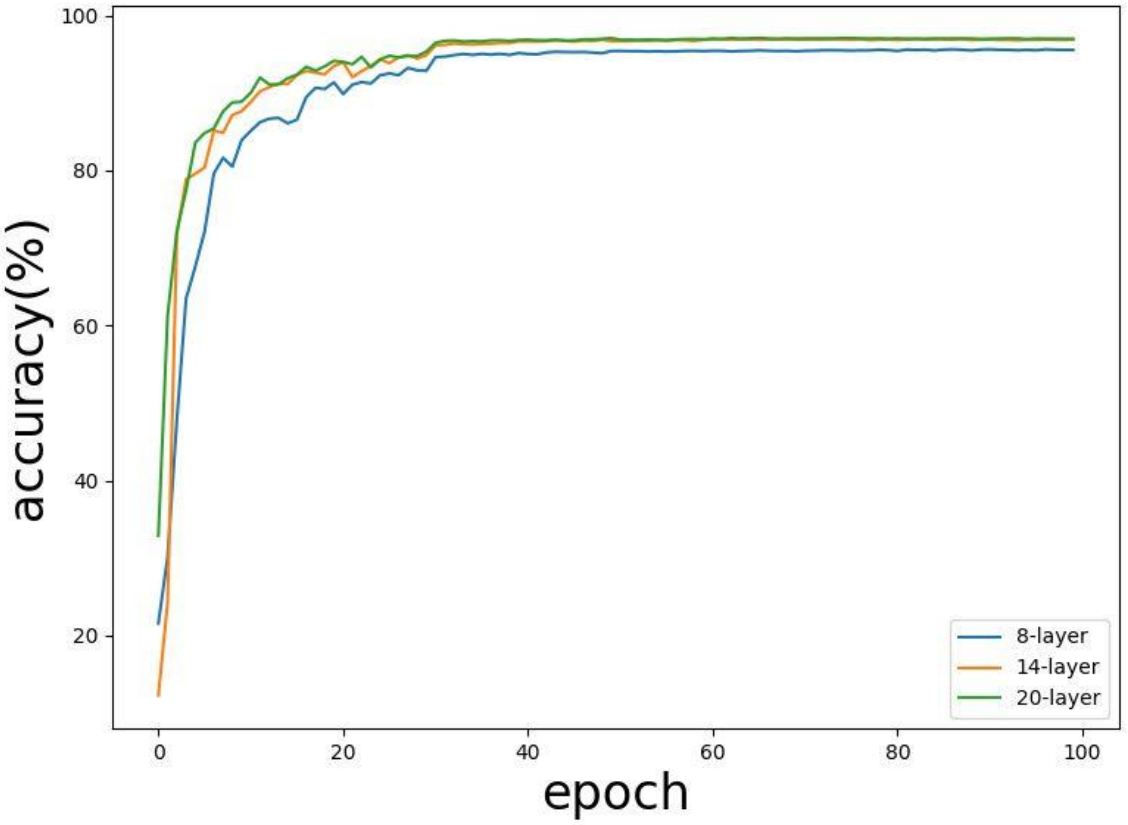


Fig. 6.1 Validation accuracy

Table 4.3 and table 4.4 show accuracy and elapsed time under different circumstances respectively. We can see that an 8-layer network architecture is enough to do the task well with accuracy up to 96.9%. Not surprisingly, a network with more layers spends more time on training, loading model, and predicting image. However, because a deeper network has more representational power, it can increase the accuracy a little bit. The accuracy of a 20-layer network is 97.1%.

When the size of the input images is equal to 30×30, the test accuracy is the highest. The reason might be that the size the images in real-world applications are close to 30×30. If the size is smaller than 30×30, there is no enough information for the model to recognize different images. If the size is bigger than 30×30, the images in real-world

doi:10.6342/NTU201903613

applications need to be resized to bigger size and would be distorted because of interpolation.

With data augmentation, we can effectively predicting icons in real-world applications well. Although we have all binary images in our dataset, by augmenting them to mimic real images as similar as possible, the network can learn good knowledge from augmented data to deal with the real-world problem.

		Accuracy		
		Training (%)	Validation (%)	Test (%)
Network Architecture	8-layer	89.8	95.6	**94.3**
	14-layer	92.2	96.9	90.6
	20-layer	92.1	97.1	90.6
Size of Input Image	8	80.7	87.7	64.2
	15	88.1	94.3	88.7
	30	89.8	95.6	**94.3**
	60	85.8	94.4	75.5
Without Data Augmentation		89.6	95.5	35.9
With Data Augmentation		89.8	95.6	**94.3**

Table. 6.3Accuracy

		Elapsed time		
		Training (min)	Loading (sec)	Predicting (ms)
Network Architecture	8-layer	26	3.44	4.26
	14-layer	26	3.49	5.26
	20-layer	32	3.52	6.32
Size of Input Image	8	24	3.44	4.24
	15	25	3.50	4.57
	30	26	3.44	4.26
	60	46	3.47	4.37
Without Data Augmentation		24	3.40	4.26
With Data Augmentation		26	3.44	4.26

Table 4.4 Elapsed time

doi:10.6342/NTU201903613

Chapter 7 Conclusion

In this thesis, we use CNN to train a model to recognize GUI elements in web and mobile applications. With a 20-layer neural network, the validation accuracy is 97.1% and the test accuracy is up to 90.6%. We also try to resize the input image to different size and compare the results. To recognize GUI elements in real-world applications, it would be appropriate if we set the size of images to be 30×30. With the size equal to 30×30, the test accuracy is up to 94.3. Because there are only binary images in our datasets, we use appropriate data augmentation methods to increase the test accuracy from 35.9% to 94.3%.

doi:10.6342/NTU201903613

REFERENCE

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. J. P. o. t. I. Haffner, "Gradient-based learning applied to document recognition," vol. 86, no. 11, pp. 2278-2324, 1998.

[2] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," vol. 65, no. 6, p. 386, 1958.

[3] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 928-936.

[4] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," vol. 7, no. Mar, pp. 551-585, 2006.

[5] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in Proceedings of the 25th international conference on Machine learning, 2008, pp. 264-271: ACM.

[6] T. Yeh, T.-H. Chang, and R. C. Miller, "Sikuli: using GUI screenshots for search and automation," in Proceedings of the 22nd annual ACM symposium on User interface software and technology, 2009, pp. 183-192: ACM.

[7] A. Mesbah, E. Bozdag, and A. Van Deursen, "Crawling Ajax by inferring user interface state changes," in 2008 Eighth International Conference on Web Engineering, 2008, pp. 122-134: IEEE.

[8] C.-H. Yu, "Using image classification for automatic page analysis on the testing of Web APPs", Master Thesis of Dept. Electrical Engineering, National Taiwan University, Jan. 2018.

doi:10.6342/NTU201903613

[9] C.-L. Li, "Predicting the topics of GUI elements with both NLP and image classification techniques", Master Thesis of Dept. Electrical Engineering, National Taiwan University, Jul. 2018.

doi:10.6342/NTU201903613