The University of Hong Kong

Faculty of Engineering

Department of Computer Science

COMP7705

Project Report
Cloud Service for Captcha Recognition Based on Deep Learning

Submitted in partial fulfilment of the requirements for the admission to the degree of Master of Science in Computer Science

By

Gao Wenzhao (3035562112), Liang Manping (3035561443), Wang Yaodong (3035562241) & Wu Yi (3035561675)

Supervisor: Dr. T.W. Chim
Date of submission: 30/07/2019

Abstract

In our daily web surfing, it is very often to encounter with many captcha verifications when we login to accounts on website, which is designed to verify that we are not a robot. Sometimes these captcha verifications can be annoying when we cannot pass the captcha even after many times of trying and inputting the captcha code. Thus, we came up with an idea to pass captcha using deep learning technology.

In this project, we introduce and implement a cloud service of captcha recognition system based on deep learning. Our captcha recognition model is based on CNN (Convolutional Neural Network), LSTM (Long Short-Term Memory) and trained on GPU farms using more than 10 thousand training samples. In addition, we encapsulate our backend server in a docker image so that other advanced users can utilize our image for further development, which improve the portability of our service. In order to improve our backend server's throughput and scalability, we deploy our application server on the Kubernetes cluster, which provides automating application development, scaling and management. For better user experience, we present our captcha recognition service with developing a user-friendly chrome extension, by which users can easily enjoy our web service after installing a lightweight chrome extension.

Keywords: CAPTCHA, Deep Learning, Non-segmentation Image Recognition, Kubernetes.

Declaration

We hereby declare that this project report represents our own work which has been done after registration for the degree of MSc at the University of Hong Kong and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma or other qualifications.

Signature:

Date:

Signature:

Date:

Signature:

Date:

Signature:

Date:

Acknowledgments

We are extremely grateful to Dr. T.W.Chim for keeping paying a close attention to our project. Each time we meet with Dr. Chim, he gave us advice about our project process. His rich experience in application developing help us to think more comprehensively.

In addition, we also acknowledge the Msc Programme Office, Department of Computer Science, Faculty of Engineering, and the University of Hong Kong for providing us computing resources, such as GPU Farm, which helped us train our deep learning model.

We would also like to extend our gratitude to Google Cloud Platform for providing us certain amount of credits to build a cluster without purchasing any physical devices, as well as other open source libraries and repositories that help us to develop our application.

The Deep Learning course we took in the second semester helps us contribute fundamental knowledge in building our convolutional neural network. We would love to thank Dr. Bethany Chan and Dr. Mauro Sozio for giving this amazing course.

Cluster and Cloud Computing course from Prof. C.L. Wang also plays an extremely important role on our project development and implementation. Without the knowledge from cloud computing, we cannot finish our project in time. What’s more, Prof. C.L. Wang is a good lecturer who not only provides us with theoretical cloud computing knowledge but also encourages us to have more hands-on practice.

Last but not all, we would love to express our special thanks to all of our teammates, Yaodong Wang, Yi Wu, Wenzhao Gao and Manping Liang. It is a great teamwork that we work efficiently and productively together. The most important is that we have learnt a lot from each other during the whole project.

Contents

List of Figures

List of Tables

1. Introduction

A CAPTCHA [1] (Completely Automated Public Turing test to tell Computers and Humans Apart) is an automated test to identify whether the user is a human or not. CAPTCHAs are often used on the internet to prevent automated programs from abusing online services. Nowadays, most service providers such as email or online shopping sites require users to solve CAPTCHAs, which most often consist of some distorted words that must be read and typed in correctly. For humans this is a comparably simple task, but computers still have difficulties in recognizing the text. Useful CAPTCHAs should be solvable by humans at least 80% of the times while programs using reasonable resources should succeed in less than 0.01% of the cases [2]. An example of a CAPTCHA is shown below.

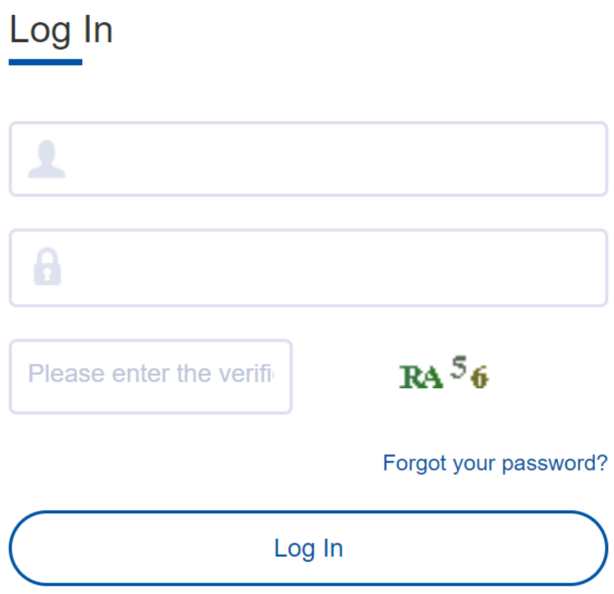


Figure 1. Common CAPTCHA. Consist of dislocation and overlapping character. Some characters even have noisy factors such as horizontal line. The user must type the words correctly to log in the system.

1.1. Background and Objectives

From daily web surfing, users will always encounter different kinds of CAPTCHAs, especially when users want to log in to a website. So far, there are the following four major types of CAPTCHAs:

- **Text-based mode** – they typically relay on sophisticated distorted words with very noisy background rendering them unrecognizable to the machine but recognizable to human eyes. Common texts include numbers, English characters, and simple mathematical formulas composed of numbers.

- **Image-based mode** – they typically require users to complete an image recognition task. The most common image-based CAPTCHA is Google's reCAPTCHA.

- **Motion-based mode** – they typically require users to perform certain movement like dragging the slider from one location to another by using the mouse.

- **Sound-based mode** – they typically require users to solve an audio recognition task. The common process is that the system will ask users to type in the words which they have just heard from the audio.

However, sometimes these CAPTCHA verifications can be very annoying. Even after many times of recognizing and inputting the CAPTCHA code, you still can't pass the verification process. In this project, we will introduce and implement an online CAPTCHA recognition application based on deep learning. We focus on solving standard distorted word CAPTCHA. Our model doesn't need any pre-segmentation and can handle variable length CAPTCHA.

In addition, in order to make our system more convenient to use and closer to users' habits. When facing common picture CAPTCHA like standard distorted word and picture identification CAPTCHA, our system uses drag to upload images. After the system recognizing the CAPTCHA results, it returns the most likely results to the user. Further, we provide a browser plugin that triggers the screenshot function by clicking on the plugin and automatically uploads the screenshot content to our system. The system further analyses the CAPTCHA and give the result.

In order to improve our backend server's throughput and scalability, we deploy our application server on the Kubernetes cluster, which provides automating application development, scaling and management. Besides that, for high-end users we encapsulate our backend server in a docker image and upload it onto docker-hub. Anyone can download it by using the following command:

```
docker pull taylorliang/mychaptcha
```

So that they can utilize our image for developing.

1.2. State-of-The-Art

Recently, researchers have started investigating automated methods to solve distorted words CAPTCHAs. Many of these existing solutions first perform character segmentation and then recognition. However, they cannot solve newer with variable length, more challenging CAPTCHAs, where the letters are slanted and overlap each other so that they cannot be separated by vertical lines. Thus, rectangular windows cannot be used for segmentation, and more powerful classification methods are needed.

One successful approach proposed recently by Kopp, Martin, Matej Nikl, and Martin Holena [3] presents an approach using neural networks and a simple clustering algorithm that consists of only two steps, character localization and recognition. The authors tested their approach on 11 different schemes selected to present very diverse security features. They focused on automatic character recognition from multiple text-based CAPTCHA schemes using artificial neural networks (ANNs) and clustering. The ultimate goal is to take a captcha challenge as an input while outputting transcription of the text presented in the challenge. Contrary to the most prior art, their approach is general and can solve multiple schemes without modification of any part of the algorithm. The experimental part compares the performance of the shallow (only one hidden layer) and deep (multiple hidden layers) ANNs and shows the benefits of using a convolutional neural networks (CNNs) multi-layered perceptron (MLP).

Another successful approach proposed by Stark, Fabian, et al. [4] uses an Active Deep Learning strategy that makes use of the ability to gain new training data for free without any human intervention which is possible in the special case of CAPTCHAs. The article discusses how to choose the new samples to re-train the network and present results on an auto-generated CAPTCHA dataset. The approach dramatically improves the performance of the network if there are only few labelled training data.

However, recognizing arbitrary multi-character text in unconstrained natural photographs is a difficult problem, which can be also used in CAPTCHA. Goodfellow, Ian J., et al. [5] indicate a method based on a deep convolutional neural network which can process on image pixels directly. Through implementing and testing this method, they find that this method’s predicting accuracy increases with the number of network layers. In evaluation stage, the authors evaluate their approach on different tasks and datasets. In the recognizing street numbers task, this approach gets over 96% accuracy on the public SVHN dataset. On a per-digit recognition task, the improved approach gets 97.84% accuracy. They also evaluate this approach on an even more challenging dataset generated from Street View imagery containing several tens of millions of street number annotations and achieve over 90% accuracy. They also apply this approach to transcribing synthetic distorted text from reCAPTCHA and achieves 99.8% accuracy in reCAPTCHA puzzle task.

1.3. Organization of the report

This report is divided into six chapter. Chapter 1 gives a literature review of CAPTCHAs, including what are CAPTCHAs and the research that have done on recognizing them. Chapter 2 briefly introduce the methodologies that are used in our project covers an introduction to neural network and various method we used in our project. Chapter 3 introduce our design and construction of system on neural network model design, online CAPTCHA recognition server, Chrome Extension – ByeCaptcha and distributed server deployment. Chapter 4 records all the experiments environment and results we got after training the dataset. Chapter 5 records the troubles we met throughout the whole project and how we deal with that. Chapter 6 concludes the whole report.

2. Methodology

2.1. Convolutional Neural Network

Early in 1995, the concept of the Convolutional Neural Network was first introduced by Yann LeCun[6]. When the motivation of developing the CNN is to help the post office to recognize handwritten characters. Convolutional Neural Network evolved from Neural Network which is a variant of multilayers perceptron. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers which is used to extract features from images.

In our project, we chose CNN to be our deep learning model. The most important advantage of using CNN is that it can be used without prior knowledge. As the input is an image, all the pixel values of this image will be directly received by the network, as contrary to hand-crafted feature values by performing segmentation and filtering in advance as mentioned earlier. CNNs take the advantage of local dependencies in the image. A pixel value is more similar to its neighbours than pixels that are farther away. A dense layer is designed to learn global information. However, CNN has local connections that works pretty well for images. Moreover, they are easier to train since they have fewer parametersAlsoMoreoverto learn They share same weights in convolutional layers, which means that the same filter is used for every pixel in the layer. This saves computational power and required memory, which will improve training performance. Another advantage is the availability of special purpose hardware like GPUs that can perform convolution very cheaply.

There are several convolutional layers stacked on a CNN model. Normally, there will be a fully connected layers on the top. We use different parts of the convolutional neural network when creating the model, parts are explained below:

Convolutional layer

The convolutional layer is the most essential building block of the convolutional network. It is the layer that applies the convolution filter on the input image (or multidimensional feature vector). Different filters are applied in each convolutional layer in the model. All pixels in the image are applied with same filters, which is also known as weight. Filter application acts as a bridge between input layers and filter vectors.

If input to a convolutional layer is an image of dimensions $W \times H \times D$ where W is the width, H is the height, D is the number of channels in the image, and the layer has K filters. Then the convolution layer will output a vector of dimensions $W \times H \times K$.

There some key advantages of convolution layers. The first is translation invariance. After learning a pattern in some region of a picture (e.g. bottom right), a convnet can recognize it anywhere. However, a densely connected network would have to start from scratch if that pattern occurs in some other region (e.g. top left). Fewer training samples to learn representations that have generalization power. The second is that they can learn spatial hierarchies of patterns. A first convolution layer will learn some small local patterns such as a vertical line, a circle, a second convolution layer will learn more complex patterns made of the features of the first layer (e.g. a circle plus a vertical line) and so on. Our visual world is fundamentally spatially hierarchical.

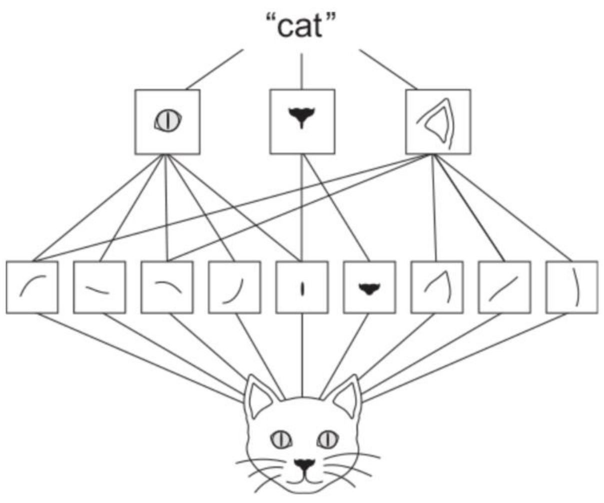


Figure 2. Spatial Hierarchies in Object Recognition. The CNN first recognizes simple features (vertical lines, edges) then more and more complex (eyes) until it finds a cat. [7]

The third advantage is that CNN uses the same matrix weight and bias for all the pixels which is called parameter sharing. In this way, the number of parameters can be significantly reduced. For example, if image has the size of 128×128, filter size is 3×3 and if there are 4 filters, we have only $4 \times (3 \times 3 + 1) = 40$ (1 for bias) weights to learn. Otherwise, it would be $\#\text{pixels} \times \#\text{pixels} \times \#\text{filters}$, which would be $128 \times 128 \times 128 \times 128 \times 4 = 1072$ million. This is way much larger number of weights and therefore difficult to learn.

Pooling layer

The pooling layer averages or maxes out feature values which are commonly stacked immediately after convolution layers. The main function of this layer is to reduce the number of parameters at the cost of losing the exact position of the feature. With a 2×2 pooling and slide = 2, still maintains a good approx. of the location while decreasing the number of parameters by a factor of 4. Therefore, having less parameters implies less overfitting, less memory consuming and more efficient. It allows next convolution layers to look at increasingly large windows over the original input, thereby allowing the detection of more complex features higher in the hierarchy.

It also ensures that same result can be detected even though there are some small translation or rotation in image features. This feature is known as Translation Invariance. In case of CAPTCHAs, the character “A” can be located anywhere in the image. The network tries to learn how the character “A” looks like, not the location of “A” in the image. That is to say, it doesn’t matter where “A” is located.

Dense layer

At the end of a CNN, we add one or more dense layers which is simply a fully connected layer with the neurons in the previous layer. Before adding the first dense layer we flatten the last tensor with size $W \times H \times D$ into $W \cdot H \cdot D$ neurons - a one dimensional vector. So that we can aggregate all together before preserving any spatial structure. Dense layers are commonly used in neural networks.

Activation function

All the neurons in a neural network add their inputs and put them in a nonlinear function, which is known as the activation function. There are various kinds of activation functions used in academic research and industry. Among these activation functions, some of the most popular are sigmoid, Tanh, and ReLU.

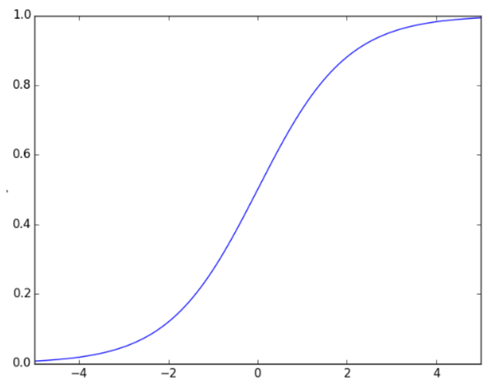


Figure 3. Sigmoid Activation Function. Sigmoid computes following function: $f(x) = 1/(1 + e^{-x})$.

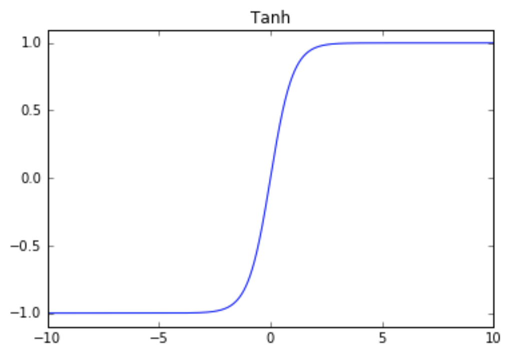


Figure 4. Tanh Activation Function. Tanh computes following function: $f(x) = (e^x - e^{-x})/(e^x + e^{-x})$.

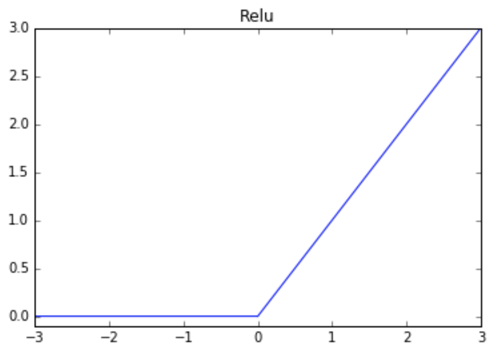


Figure 5. ReLU Activate Function. ReLU computes following function: $f(x) = \max(0, x)$. However, gradient is not defined in 0, as usual case, we assume that to be 0.

In our models, we used ReLU activation function. The reason why we use ReLU is because it has some advantages as follow:

- No vanishing gradient problem in the positive region.
- It is computationally efficient and it converges quicker than sigmoid/tanh since it is very simple and normalization is not required.
- It is widely used in CNN since Alexnet was published to the public.
- It can model better neurons.

Softmax Layer

A softmax layer takes the activations (probabilities) calculated so far and divides each of them by the sum of all the activations. That is to say, activations are normalized by softmax layer. Therefore, it forces the layer layer to take the form of probability distribution, for example, between [0, 1]. From these, the value with the maximum probability is chosen to be the output of the network. This kind of activation function can only be applied on the last layer.

Loss Function

Training a convolutional neural network is technically an objective function minimization problem. The objective function is also known as loss function which measures how much the prediction label of the model are different from the actual labels. During training, the gradient of the loss function is computed with respect to each weight $w_{i,j}$. We are able to calculate how a small change in weight can help with decreasing the loss. There are various types of loss functions that can be used in neural network training, such as mean squared error, cross entropy, logistic loss, hinge loss, and hamming distance loss.

Learning rate

Learning rate is one of the most important concepts in neural network training phase, this parameter decides how much the value of weight will be changed in order to converge. A low value of learning rate will take many iterations till convergence and may be trapped in local minimum. However, a large value of learning rate may cause the network to bounce around the optimal parameters and no convergence.

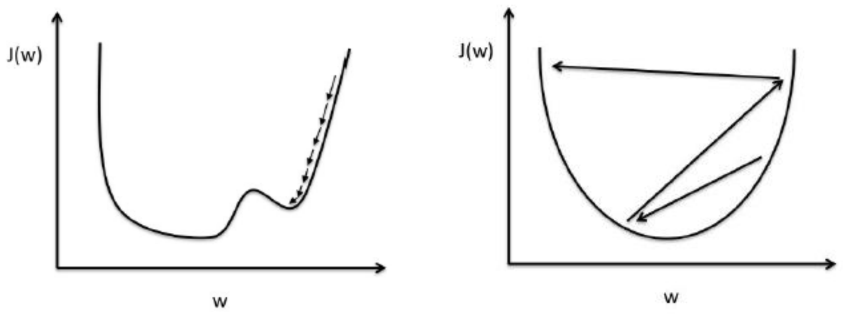


Figure 6. Impact of different learning rate. Left: with a lower learning rate, the loss tends to decrease slowly. **Right:** with a large learning rate, the network could be overshooting.

Recurrent Neural Networks

There is an disadvantage in feedforward networks, which is, they accept a fixed-length of input vector and only generate a fixed length of output vector. In addition, they perform mapping using the same set of layers, that is, it does not change at all.

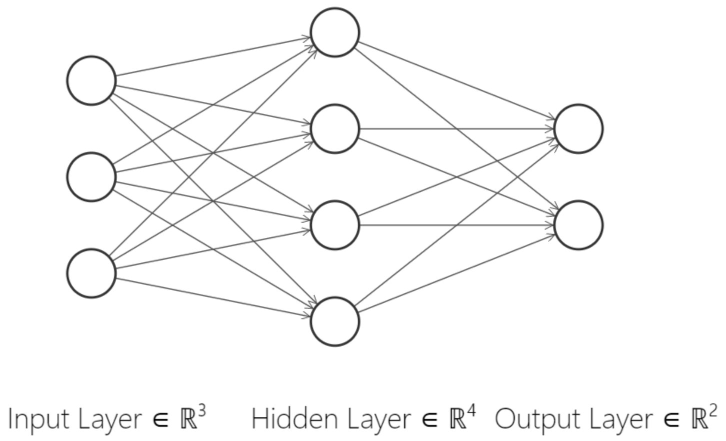

Input Layer ∈ $\mathbb{R}^3$
Hidden Layer ∈ $\mathbb{R}^4$
Output Layer ∈ $\mathbb{R}^2$

Figure 7. A Sample feed-forward network

Recurrent Neural Networks, also known as RNNs, can be used to overcome the disadvantages mentioned above. There is a directed cycle between connections units in RNNS. One of the reason why RNNs are called recurrent is that, they perform the same task for each element sequently with the output being dependent on the previous computations, which is not the case in feedforward networks, they just go in a forward directly using values that calculated before. Different from the previous one, internal memory is used in RNNS to process arbitrary input sequences. There are a lot of applications using RNNs including handwriting recognition and speech recognition. An example of a RNN network is shown below:

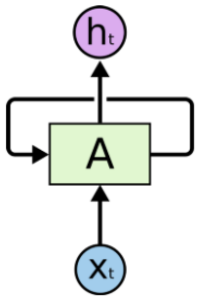


Figure 8. An example of RNN

Figure 8 can be interpreted as **Figure 9** after being unfold.

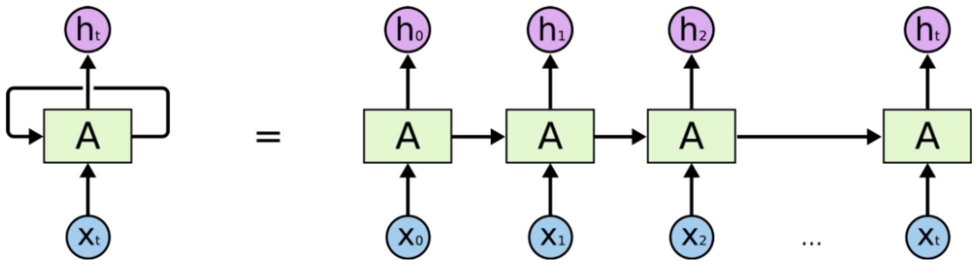


Figure 9. An example of RNN with unfolds.

RNNs return a good result in word-sequence prediction where the output from one field is correlated to another field[8]. One of the most frequently used RNNs is LSTM (Long Short-Term Memory), which has been used in this project. We will discuss more on LSTM in the next section.

LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTMs) networks are a special kind of RNNs. Training RNNs is difficult because of vanishing and exploding gradient issues. If the value of gradients and weightsare too small, gradients will decrease quicklyin backpropagation phase of training and vanish finally. Conversely, if the value of gradients andweights are too large, gradients will increase quickly during backpropagation then become “exploding”. The former iscalledvanishing gradient problem, and the latter is known as exploding gradient problem. The issue is inparticularly acute when the input or output sequences are long, as the gradients either vanish orexplode while back propagating through long sequences. LSTM improves this problem as described below.

In LSTM blocks, when the error values are back propagated from the output, the error becomes trapped in the memory portion of the block. Figure 10 shows a structural diagram of the LSTM memory cell.

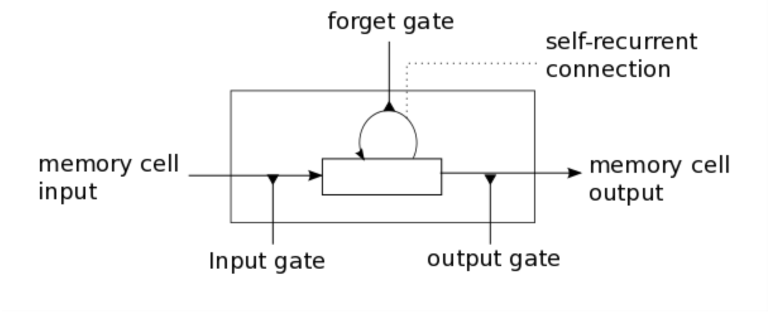


Figure 10. Memory cell in LSTMs [9]

In addition, tanh function is used in LSTMs as an activation function. A tanh gradient is always between 0 and 1, so the gradient does not explode when multiplied with gradients during backpropagation.

LSTMs are technically created to prevent the long-term dependency problem. They have shown great performance in unsegmented handwriting recognition [10], machine translation, and speech recognition.

There are two types of LSTMs: forward LSTMs and backward LSTMs. In the case of forward LSTMs, we consider the output that we have so far. For example, in the case of speech recognition, consider a sentence like “I __ a student” that a network wants to complete. In this case, we can make use of “I” to predict that it should be “am”. So, we took help of the predicted character “I”. Forward LSTMs first predict “I” and then “am” and so on.

In backward LSTMs, we try to predict backwards. For example, in the above example, it will predict “student” first, then “a”, then “am” and lastly “I”. In bidirectional LSTMs, both forward and backward LSTMs are combined. They try to predict all the characters together. For example, “He likes to eat __. He goes to the McDonald”. In this sentence, we know that because of “eat”, we need to fill the name of some food. But by only looking at McDonald, we could make a decision that it would be burger. In these scenarios, bidirectional LSTMs are useful.

We also hope that bidirectional LSTMs can be helpful in the CAPTCHA recognition problem. For instance, in problems like "rrn", the output could be ['r', 'm'] or ['r', 'r', 'n'], so it is useful to predict if the last character is 'm' or 'n' first and then predict the previous characters.

2.2. TensorFlow

In machine learning research and production field, TensorFlow is the most popular open-source library, which provide APIs for different levels of users, both beginners and experts can get functions they need. TensorFlow enable users to train their machine learning model by CPUs or GPUs in personal or professional devices and build their machine learning model without writing underlying code.

In addition, TensorFlow also provide high level APIs such as Keras for constructing, training and testing deep learning models, and Estimators which provide models for large-scale training and learning. A large amount of pre-trained models and dataset built by community for many different application scenarios, so users of TensorFlow do not need to build their model from the very beginning.

2.3. Docker

Docker is a container platform designed to make it easier to create, deploy, and run applications using containers. Containers can be used to pack up an application with all of requirements it needs, including environment settings, libraries and dependencies installation, and migrated to out as one image package. By using container technique, we can rest assured that the application can successfully run on any other machine regardless any other customize environment that machine has. Since it saves us a lot of effort from environment settings, we can pay more attention on application development without worrying dependencies settings every time we move the application to other machine or platform.

2.3.1. Docker and virtual machine

In some extend, Docker is like a virtual machine. Unlike a virtual machine however, rather than creating the whole virtual operating system, multiple applications, each with different environment settings within their own containers, are able to run in the same Linux machine, which gives a significant performance boost and makes use of resource in a more sufficient way.

2.3.2. Virtualization technique

Virtualization is technique that allows users to create multiple simulated environments from a single physical hardware system. Software uses hypercall (hypervisor call) to directly connect to the hardware and allows you to split one system into separate, distinct, and secure environments known as virtual machines (VMs) [11]. Hypervisor is responsible for separating a machine's resource from hardware and redistributing them to different virtual machines.

2.3.3. Docker container

Apart from virtual machine, which is a form of visualization technique, Docker container is another form of virtualization which is popular and widely used because of its efficiency and convenience. As shown in **Figure 11**, Docker virtualizes operating system and splits them into multiple virtualized parts and run them on different containers. By using this approach, we can easily distribute and strip an application into several specific functions so that each function can be maintained in a more efficient way.

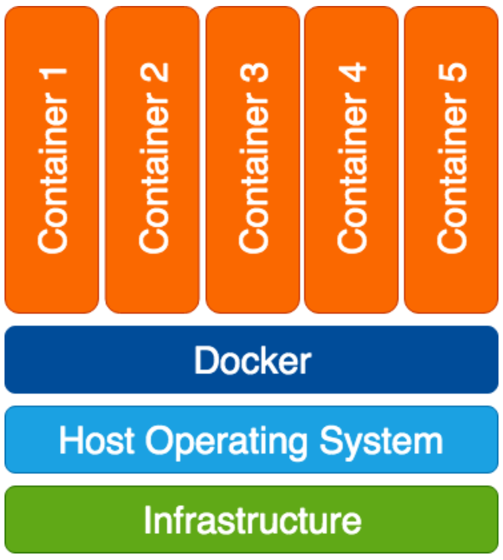


Figure 11. Docker Container.

2.3.4. Docker image

Docker image is a file that combines file system and its parameters. Docker image is similar to snapshot when it comes to virtual machines. A Docker image can run on a Docker container. Docker container can be packed into a Docker image. Docker image is a static file that contains information how an operating system is defined including its running environment. In this project, we run a Linux operating system in a container, set a specific running environment as required by our captcha recognition server, install all libraries and dependencies needed, and export the container to a Docker image using Docker commit command. With this image, we can easily migrate an application to any other

2.3.5. Docker hub

Docker Hub is a cloud-based repository where Docker users create, test, store and distribute their container images. Through Docker Hub, we are able to access open source image repositories which is public to all Docker users. Docker hub can also be used as a warehouse to store private container images. In our project, we push our Docker image into Docker hub, pull the image down and run the image on Google Kubernetes.

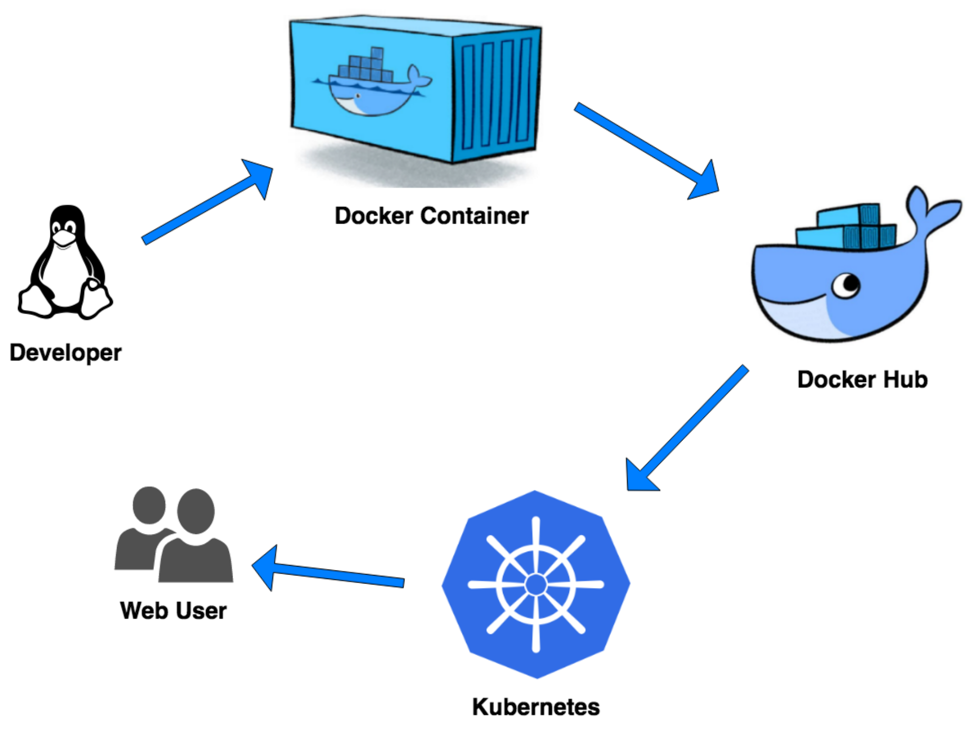


Figure 12. Relationship between Docker container, Docker hub and Kubernetes.

The figure above (**Figure 12**) shows the relationship between Docker container, Docker hub and Kubernetes. Developer create running environment, implement and deploy application in a Docker container. A container is packed into a Docker image and pushed to Docker hub. In Kubernetes, image is pulled down from Docker hub. Application image runs on Kubernetes and Kubernetes provides service for web users.

2.4. Kubernetes

Kubernetes (K8s) [12] is an open-source system which is designed by Google. Using this system, developers can deploy, scale and manage their containerized applications automatically.

Nowadays, most of information technology services are deployed on web server and are available to users at any time. At the same time, developers need to update their deployment’s version sometimes. In order to satisfy these requirements, containerization can help developers to deploy their applications in a short time without influencing the web services which are accessible to users. With Kubernetes system, containerized applications can be easily managed by developers and find the resources they need to work normally.

Kubernetes is a distributed system that are deployed in several computers, it integrates a cluster of computers and make them work as a single working unit. In the past deployment model, developers had to install their applications directly on each single nodes of the cluster. While in the Kubernetes deploying model, Kubernetes will distribute and schedule the applications to the whole cluster automatically.

2.4.1. Cluster structure

In a Kubernetes cluster (**Figure 13**), there are two types of nodes: Master node and worker nodes. The master node works as a coordinator, developers mainly interactive with master node so as to manage the whole Kubernetes cluster. The other nodes are worker nodes, where application actually running on. For each worker node, there is an agent process called Kubelet which directly communicate with the master node by Kubernetes API and manage the specific node, and the kube-proxy process takes charge of Kubernetes networking services.

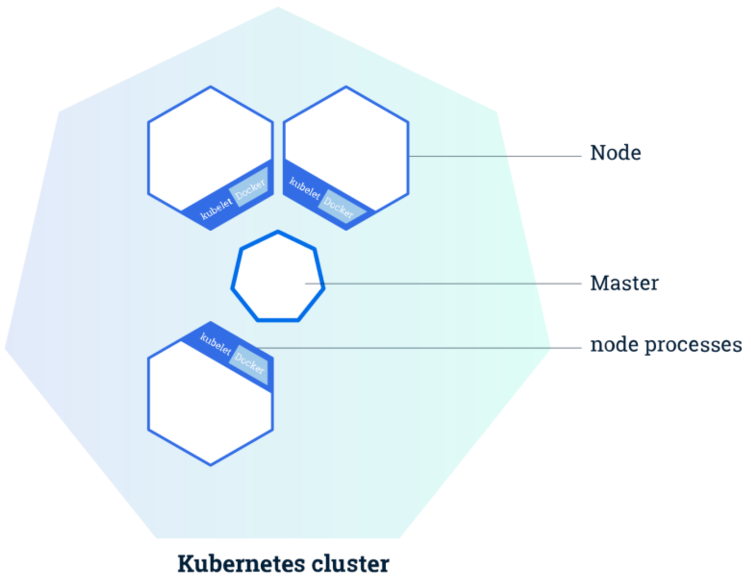


Figure 13. Kubernetes Cluster Diagram

In addition, Kubernetes System also contains several concepts that describe the running state of the cluster. Some concepts are closely related to our project.

- **Pod** is the smallest application running unit in the Kubernetes running model, it represents a running process in Kubernetes worker node. Commonly, one Pod runs a single container or several containers if they are working simultaneously, and docker mentioned previously is the most popular container to be used in K8s.
- **Service** is a special mechanism that can make the applications running in Pods to be exposed as a network service. This mechanism can free developers from configuring their applications to adapt different service discovery mechanism. In Kubernetes system, IP addresses and DNS name would be distributed to Pods automatically.

2.4.2. Deployment architecture

Figure 14 shows the architecture of applications which deployed on a Kubernetes cluster. For developers, they directly interact with Kubernetes Master node via API server and send commands to the Kubernetes cluster such as deploy, start and restart applications, increase and decrease the number of replicas. When API Server receives commands, it sends these messages to the Controller Manager and Scheduler, which take charge of actually scheduling and managing the cluster. After processing developer’s commands, the Kubernetes Master node broadcasts messages to other Kubernetes worker nodes’ Kubelet, which is the manager agent process of working node. The Kubelet process controls the Pods running in the worker node, and Pods represent the smallest running unit of the application. Kube-proxy provides the Kubernetes network service and exposes the applications to external network. Thus, users can use the application deployed on Kubernetes cluster.

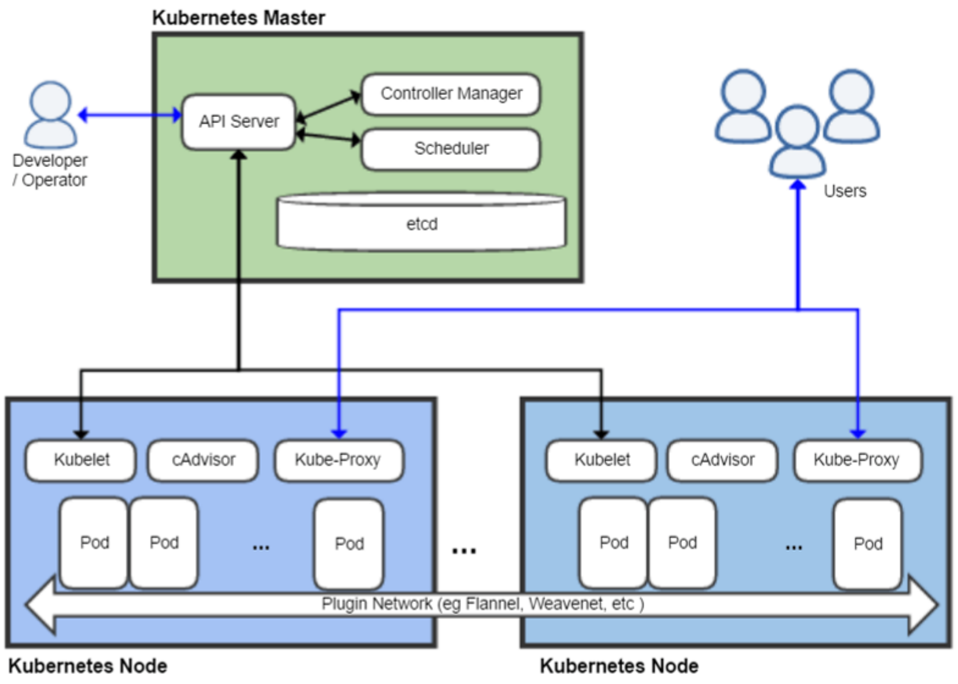
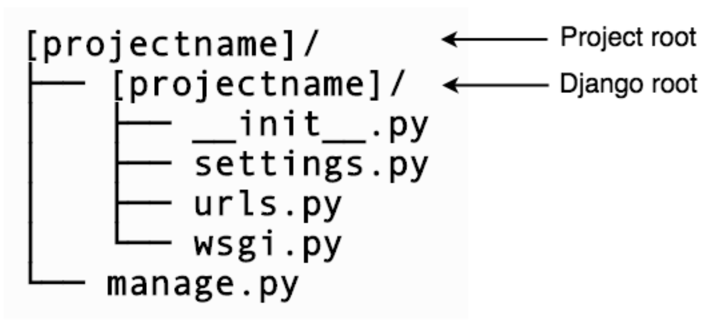


Figure 14. Kubernetes architecture diagram

2.5. Django

Django is a python-based open-source web framework that encourages rapid web development [13]. Django was first released in 2015 in the effort of a group of experienced web developers [14]. As stated on the Django official website, there are three main characteristics in Django, fast, secure and scalable. It is designed to help developers develop their web application as quickly as possible. What’s more, it helps developers to avoid common security issue during development. Also, it is flexibly scalable, which means it also suitable for business web application. In this project, we are going to use Django framework to build our captcha recognition server.

Same as many other application frameworks, there is a basic project structure in Django. As shown in **Figure 15**, once a new Django project is created, there are project root and Django root. Django root sits under project root. An interactive interface file named manage.py is right under project root. It plays an important role in Django project that enables us to interact with Django project. Under Django root, there are configuration file settings.py, URL declaration file urls.py, WSGI entry-point wsgi.pg, and other customized files.

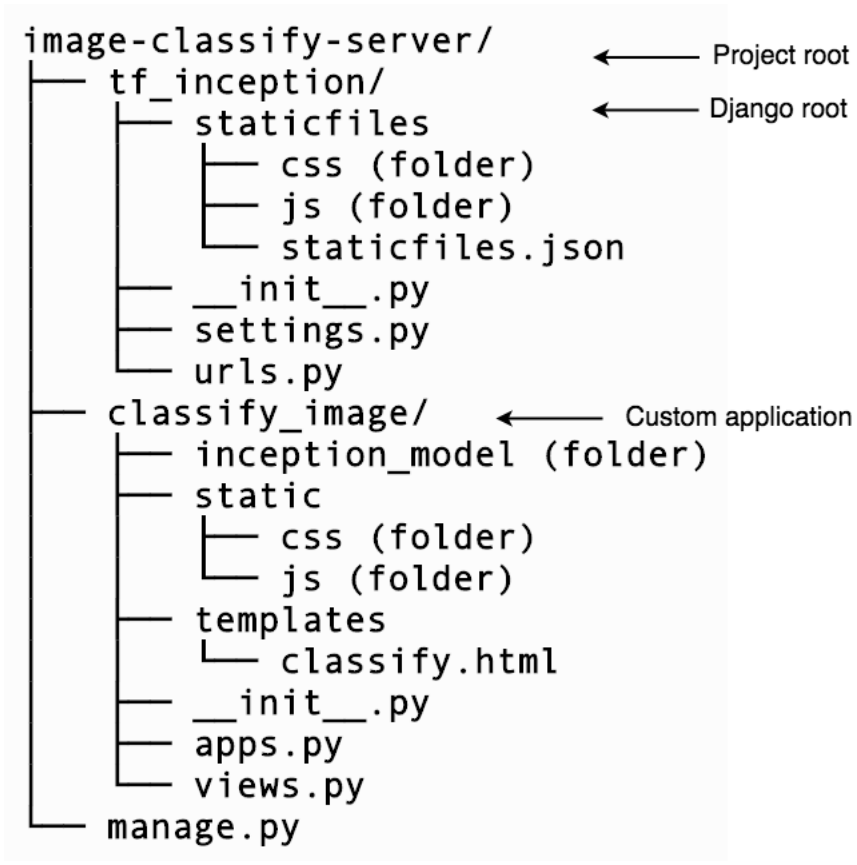


```
[projectname]/
├── [projectname]/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Figure 15. Structure of a New Django Project

As you can see in **Figure 16**, in our Django project, interactive interface file manage.py sits under project image-classify-server. Besides manage.py, there are Django root tf_inception folder and custom application classify_image folder. Django root tf_inception is the project's connection to Django. Django settings file settings.py contains all configuration about this project. URL configuration file urls.py is a mapping between URL and views.

Apart from Django root tf_inception folder, there are another folder in project root named classify_image, which acts as custom application. All our customize function will be located in this folder. Inside the custom application folder, there are front-end HTML file in templates folder which define how the user interface looks like. There are CSS and JavaScript file in static folder. In the first time for deployment, we need to collect static file from static folder. A staticfiles folder will be created under Django root tf_inception. Pretrain model is stored as protobuf(.pb) format in inception_model folder. View file views.py in custom application folder is responsible to handle JSON response, load pretrain model, deal with API request and return text result.



```
image-classify-server/
├── tf_inception/
│   ├── staticfiles
│   │   ├── css (folder)
│   │   ├── js (folder)
│   │   └── staticfiles.json
│   ├── __init__.py
│   ├── settings.py
│   └── urls.py
├── classify_image/
│   ├── inception_model (folder)
│   ├── static
│   │   ├── css (folder)
│   │   └── js (folder)
│   ├── templates
│   │   └── classify.html
│   ├── __init__.py
│   ├── apps.py
│   └── views.py
└── manage.py
```

Project root
Django root
Custom application

Figure 16. Structure of Django Captcha Recognition Project

2.6. Extension

Extensions are browser plug-ins that contain a small software program to a better browsing experience. An extension is customized for a specific function and tailor-made for a specific type of browser. Extensions are commonly written in JavaScript, HTML and CSS [15]. Since Google Chrome is widely used worldwide, we take Google Chrome for example to develop our extension.

Although extensions are made of various components with respect to the functionality of the extension, there are generally some components that often included in an extension, components including background scripts, content scripts, an option page, UI elements (UI interfaces), a must-have file named manifest, and other logic files that keep the extension running in a normal way as expected [15].

2.6.1. Manifest

Manifest file is compulsory for any extension. It is a file in JSON format that clarifies extension name, version, and a short description about what the extension is designed for. Apart from all the required important information mentions above, it also contains other functional related information like browser action, the location of background scripts, the location of web accessible resources, icons of the extension, and declaration of permissions. Short cut keyboard commands can also be set in this manifest file.

Background scripts are written in JavaScript that in charge of evens managing. Events are triggers within a browser, including opening a new tab, navigating to another tab, closing an existing tab, setting a bookmark, sending a message and so on. In our extension, screenshot cropping function will be implemented on background scripts. The background script is loaded when it is needed, otherwise it will go in an idle state when it is unloaded.

2.6.2. Content

Content scripts are files with the access of Chrome API. They run in the context of a web page and they have the ability to read the detail information of a web page. Content scripts can be written in HTML with inject JavaScript. The HTML code of content script is isolated from the current web page that serves as an independent section web page. Injected JavaScript in content specifies how to extension works. A HTML script is not compulsory for content in extension. In order to simply the usage of our extension, in our project, we don't need an additional web page section. AJAX is used in JavaScript content with POST request. Image data is encoded into image64 format. Return data type is set to text because we want to return the recognition result of a captcha image. If POST method goes successfully, the result text will be copied to clipboard.

3. Design and Construction of System

According to our objectives, we design two versions of deep learning model for recognition, the naïve version for fixed-length characters while the advanced version for variable-length characters, both of which requires no pre-segmentation on input captcha images.

As well as two kinds of user-interfaces are supported in our system, a web page for uploading captcha images and a google chrome extension with screen-shot function for convenience.

The backend recognition server and frontend user-interface are connected via Django web server. HTTP Post would be sent asynchronously using jQuery Ajax from frontend. Then whole service above would be encapsulated into docker image, uploaded on docker hub and easily deployed onto Kubernetes.

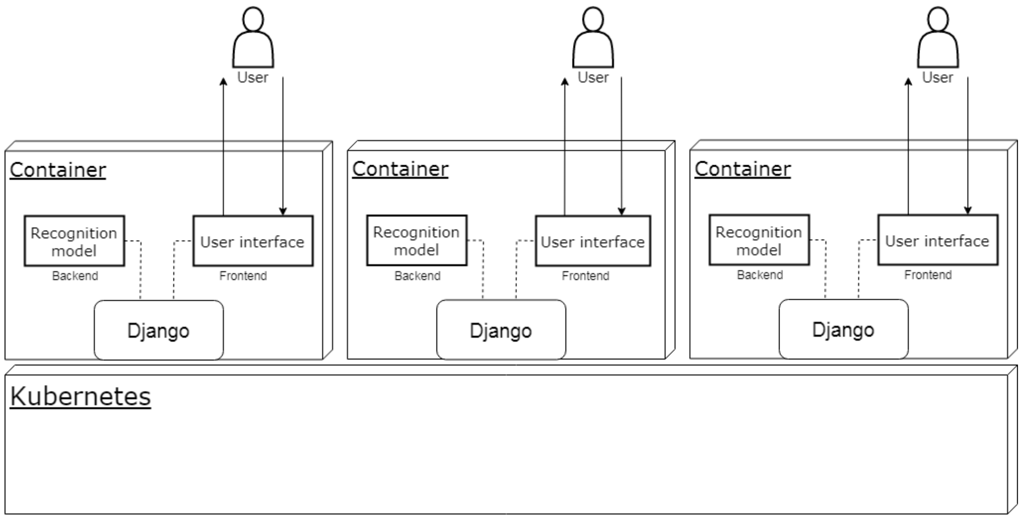


Figure 17. Overall System Design

3.1. Fixed-length captcha model design

3.1.1. Network Model

For fixed-length recognition version, we propose a Convolutional Neural Network solve the whole sequence of a CAPTCHA image, no segmentation needed before. The network model is composed of four convolutional, four pooling and two fully-connected layers, shown in **Figure 18**. Outputs of the last layer represents the probability distributions for all digits, from which we can inference the prediction and compute the uncertainties of characters.

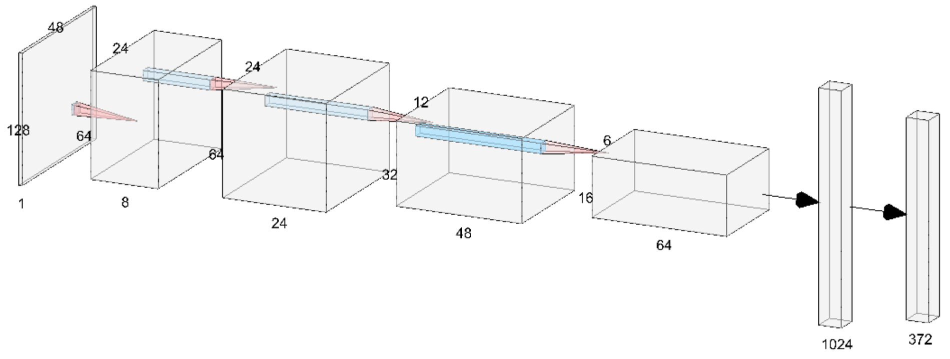


Figure 18. Fixed-length version network model used in our project.

We focus on CAPTCHAs with 6 digits. Each of which is represented by 62 neurons in the output layer, so 372 output probabilities in total. We define a bijection Θ(x) that maps a character x ∈ {‘0’,...‘9’, ‘A’,..., ‘Z’, ‘a’,..., ‘z’} to an integer $l \in \{0, 1, 2, \dots 61\}$:

$$\Theta(x) = \begin{cases} 0 \dots 9, & if\ x = '0' \dots '9' \\ 10 \dots 35, & if\ x = 'A' \dots 'Z' \\ 36 \dots 61, & if\ x = 'a' \dots ' z' \end{cases}$$

We assign the first 62 output neurons to the first digit of the sequence, the second 62 neurons to the second digit and so on. Thus, for a digit x_i the neuron index n is computed as $\mathrm{n} = \mathrm{i} \cdot 62 + \Theta(x_i)$, where i ∈ {0, ..., 5} is the index of the digit. To predict a digit, we consider the corresponding 62 neurons and normalize their sum to 1, using softmax regression function. Fig. 4 shows an example of our network output. Here, the predicted character index for the first digit is $c_0 = 52$ and the predicted label is x = $\Theta^{-1}(c_0) = 'q'$.

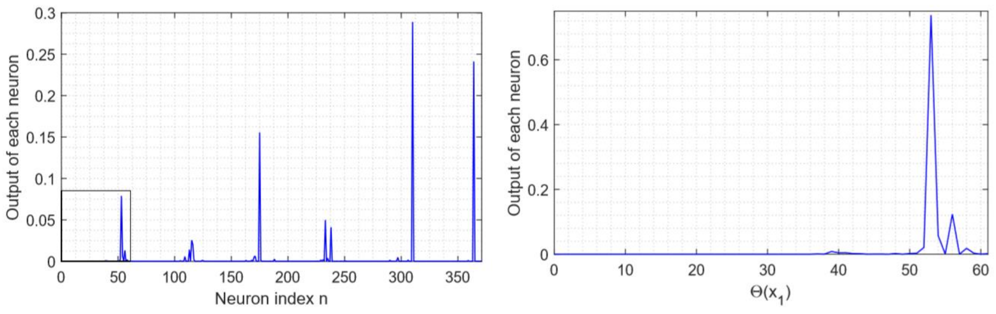


Figure 19. Output of the last softmax layer in CNN. Left: Each sub-continues box consists of 62 output values, representing one digit. **Right:** Distribution of probabilities shown in the first box. Softmax function guarantees the sum of each box is normalized to 1.

3.1.2. Loss function

Then we need to define a loss function for our proposed fixed-length recognition network model. As illustrated above, we compute the predictive probabilities distribution of each digit, and normalize the sum of each box to ONE. According to this objective, we use one-hot vector representing the output and the overall loss function is defined as cross-entropy:

$$softmax: a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

$$\mathcal{L} = -\frac{1}{d} \cdot \sum_{i=1}^{d} (y_i \cdot \log(h_i) + (1 - y_i)\log(1 - h_i))$$

Where h is the predicted probability and y’ is the true label.

3.2. Variable-length captcha model design

Since training process is a time-consuming process, also unacceptable when we have to train several different models respectively for characters in different length. An ideal recognition model should be able to handle captchas with different length characters, and we proposed such a network model combining CNN and LSTM, shown in **Figure 20**.

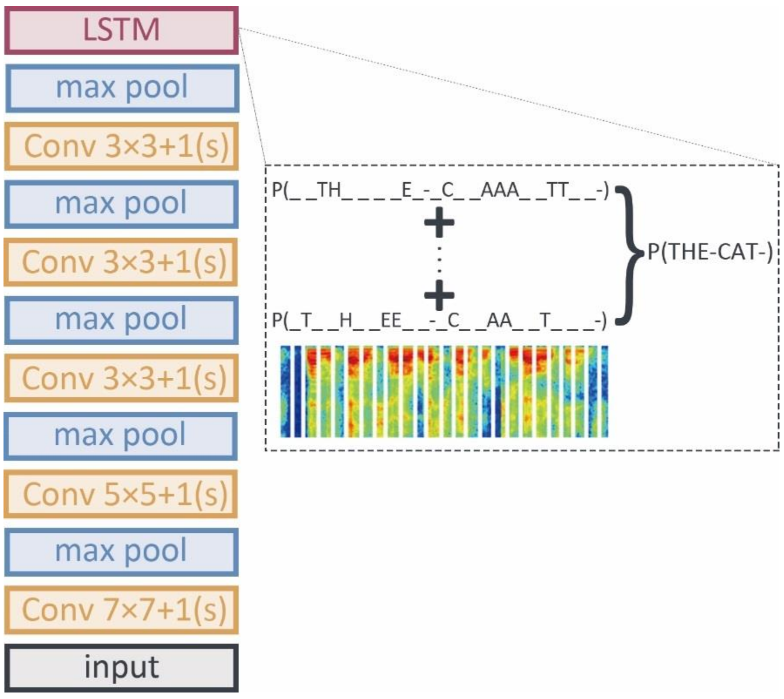


Figure 20. Combination of CNN and LSTM. Each convolutional layer is followed by an activation and max pooling layer. The output of CNN is input of LSTM.

Almost same as fixed-length version model, we use CNN as the feature extractor. The CNN consists of 5 convolutional layers, using 7×7, $5 \times 5, 3 \times 3, 3 \times 3, 3 \times 3$ sized kernel respectively, and each followed by max pooling layer as well as ReLU activation layer. The last output layer of CNN is flattened then as the input of LSTM network.

In terms of LSTM, it is one kind of RNN which is suitable for sequence signal processing. For our captcha recognition task, we introduce a loss function from Baidu research, widely used in the area for speech recognition, called Wrap-CTC.

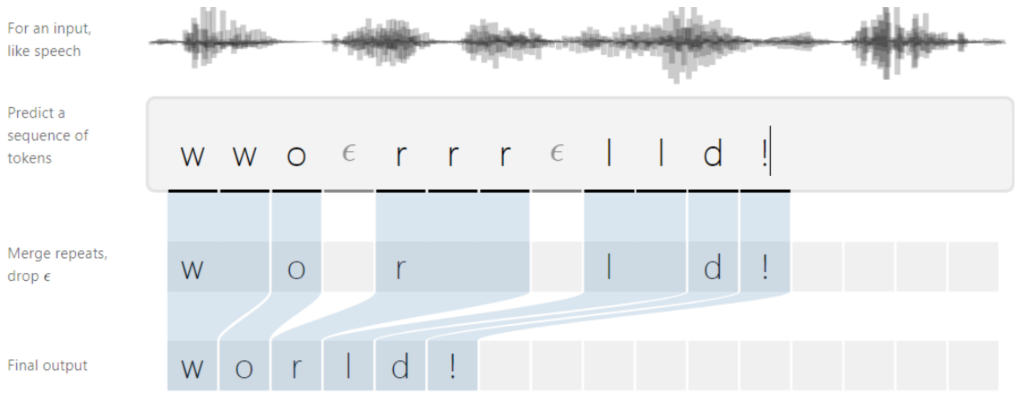


Figure 21. CTC loss used in speech recognition task.

Connectionist Temporal Classification (CTC) is a loss function useful for performing supervised learning on sequence data, without needing an alignment between input data and labels. Since both speech and captcha characters are sequence data, we find the common feature here and try to make some change to this loss function and apply it.

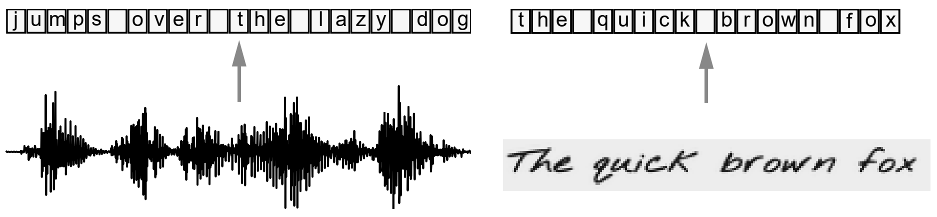


Figure 22. Common feature between speech recognition and characters recognition in image.

For example, CTC can be used to train end-to-end systems for characters recognition, which is how we have been using it in our system. Specifically, CTC loss can be defined as:

$$\mathrm{P}(X) = \sum_{A \in \mathcal{A}} \prod_{t=1}^{T} p(x_t \,|a_t) \cdot p(a_t \,|a_{t-1}).$$

Where $p(x_t \,|a_t)$ is the emission probability and $p(a_t \,|a_{t-1})$ represents the transition probability. P(X) is the probability of the input, which is the marginalization over alignments. The loss function could be more simplified as:

$$\mathrm{P}(X) \propto \sum_{A \in \mathcal{A}} \prod_{t=1}^{T} p(a_t \mid X).$$

3.3. Data Preparation

Python captcha is a python open-source library to generate image and audio CAPTCHAs. It was first released on 25th November 2014. Now we are using the newest 0.3 version. Since our CAPTCHA recognition problem is a machine learning problem, we need a huge amount of data to train our models. In our project, we use this python library to generate captcha images dataset. There are two phases in data preparation, the first phase is captcha generation, the second phase is TFRecords conversion.

3.3.1. Captcha generation

In captcha generation phase, our goal is to generate a huge amount of captcha image data using python captcha library. We set image height to 48, image width to 128 and font size to 40. The generated string is from a set of upper-case alphabets, lower-case alphabet and a series of numbers from 0 to 9. Different length of captcha images are generated, with length of 4, 5, and 6 characters or numbers. 10,000 images will be generated for each type of length. Captcha generation information is shown in **Table 1** below.

Attribute	**Value**	
Image Height	48	
Image Width	128	
Font Size	40	
Image Format	JPG	
Character Sets	ABCDEFGHIJKLMNPQRSTUVWXYZ abcdefghijklmnpqrstuvwxyz 123456789	
Data size	Length = 4	10,000
	Length = 5	10,000
	Length = 6	10,000

	Total	30,000

Table 1. CAPTCHA data

Table 2 shows how the captcha data looks like. Colour of images is randomly generated. Noises are added into captcha images. File name of each captcha image follows a naming rule, that is captcha string plus a 1 to 5 digits random number. The random number is to prevent data duplication. Take captcha "ANHMI" for example, the image will be named by "ANHMI_num46892", where "46892" is a random number.

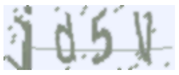
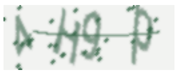
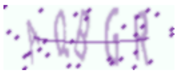
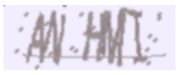
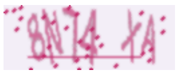
Length = 4		**Length = 5**		**Length = 6**
				
jd5v	4H9P	AQBGR	ANHMI	8N74XA

Table 2. Captcha Image

3.3.2. TFRecords conversion

The second phase is to convert captcha image from JPG format to TFRecord format. The TFRecord format is a simple format for storing a sequence of binary records [16]. It is a useful data format for TensorFlow to read data efficiently, especially when linear reading is performed in the serialized storage dataset. It is widely used in TensorFlow for data streaming.

There are 2 steps in TFRecords conversion. The first step is to create data list. Zip function is used int step 1 to zip images and their labels into a tuple. The second step is to convert data tuple in step 1 into TFRecords. Suffix of a TFRecords file is "tfrecords". Thousands of captcha images are pached into a TFRecords file with file name "train.tfrecords".

3.4. Online Captcha Recognition Server

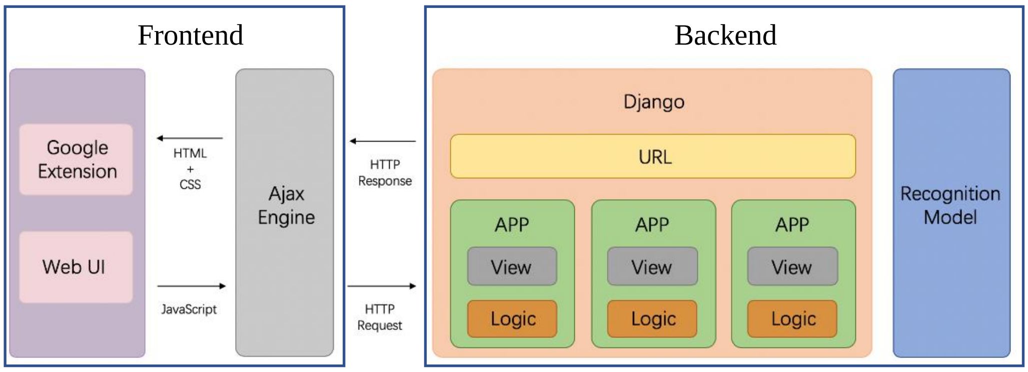
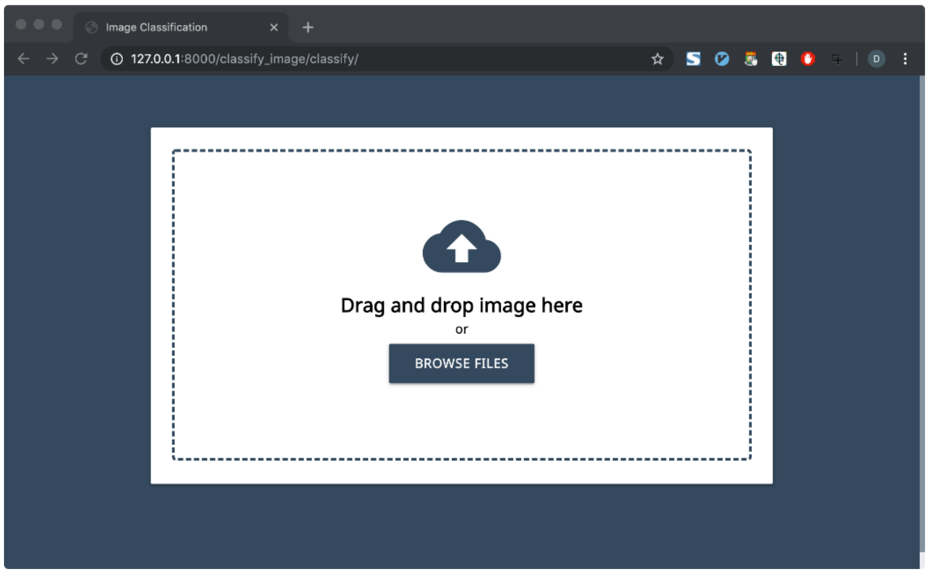


Figure 23. The online captcha recognition server structure

Aiming at effectively development and clearness of work assignment, we separate our system into two main parts. The first part is frontend, including user interface and Ajax engine, while another part is backend, which consists of Django and recognition model.

In detail, we develop two kinds of user interfaces in frontend, web UI and google extension respectively. The web UI shown in Fig.24 is developed with HTML and CSS, where users could drag and drop captcha images from their own laptops or mobile devices. Then we start Ajax engine, written in JavaScript, to wrap up the uploaded images and send HTTP POST requests to the specific URL assigned in backend, which would be caught by Django.



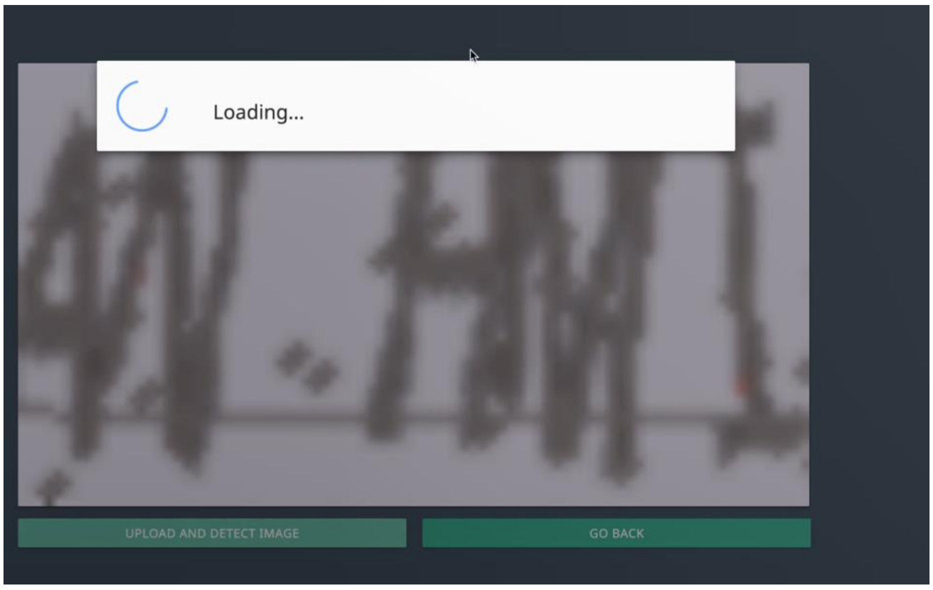


Figure 24. Web UI interface of our online system. Users could either click and choose or drag and drop images from their PCs and mobile devices.

When catching the HTTP requests from frontend, the Django would distribute these requests among APPs. Each APP is corresponded with one URL, and we develop several APPs with different functions, e.g. for solving requests with fixed-length captcha from Web UI and for solving requests with variable-length captcha from Google extension. Views in APPs should be designed respectively for different objectives, as well as internal logics.

After analysing the requests by Django, we successfully decode the initial images from Web UI, which would be sent to our recognition model trained before. The variable-length version model could support different types of APPs. In order to avoid time-consuming, we start a TensorFlow session at very beginning, and do not release this session until manually close it. Since it would take long time to initialize the computing graph and load the parameters to model from local disks, keeping the model in memory once started is a good choice. Due to the unreleased session, the inference process of our model shall be finished in millisecond, so that users might not feel the delay.

Getting the recognition results, Django generates a HTTP response carrying these results and sends back to frontend. Ajax engine then will solve the HTTP responses and decode them, inserting the results into correct locations.

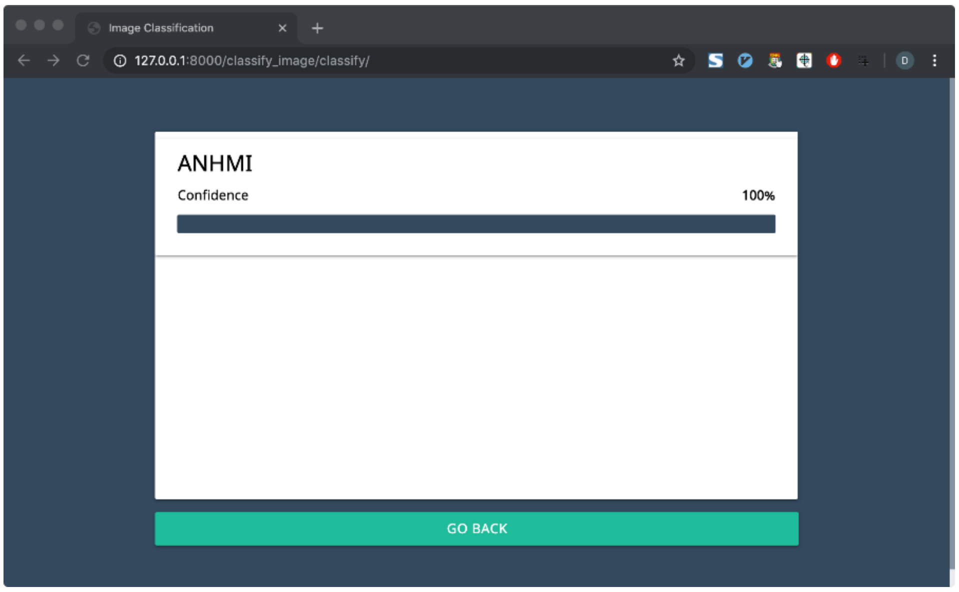


Figure 25. Recognition result in Web UI interface

The process above is mainly focused on Web UI interface, much similar to Google Extension interface which would be detailed in the following section.

3.5. Google Chrome Extension – ByeCaptcha

In many cases, web users may prefer to solve captcha test directly on their current web page rather than uploading the captcha image on other interface. In this project, we develop and implement a Chrome extension - ByeCaptcha as another easy-to-use user interface for captcha recognition. Once installed, user can click the captcha extension icon to activate the extension, take a screenshot by cropping on a specific region where a captcha image is needed to be solved. After cropping, captcha screenshot will be uploaded immediately to our captcha recognition server which has been discussed on *3.4 Online Captcha Recognition Server*. Captcha image will be sent to our captcha recognition server by POST request and processed by the pre-trained CNN model as discussed on *3.1 Fixed-*

length captcha model design and *3.2 Variable-length captcha model design*. Recognition result will then be sent back and copied to the user clipboard. Last thing to do is to paste (with keyboard short cut CTRL+V) the recognition result to the input, and you can easily pass the recognition test.

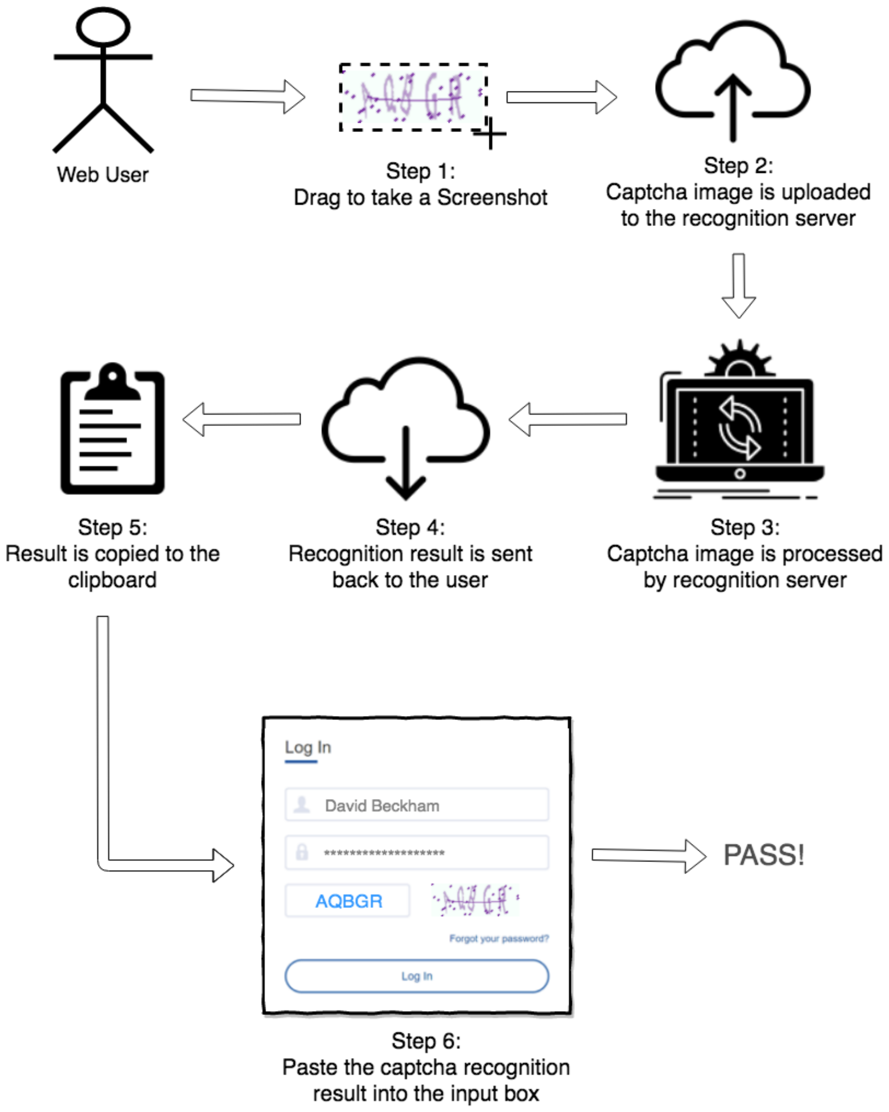

Web User
Step 1:
Drag to take a Screenshot
Step 2:
Captcha image is uploaded to the recognition server
Step 3:
Captcha image is processed by recognition server
Step 4:
Recognition result is sent back to the user
Step 5:
Result is copied to the clipboard
Log In
David Beckham
AQBGR
Forgot your password?
Log In
PASS!
Step 6:
Paste the captcha recognition result into the input box

Figure 26. From Server's Perspective

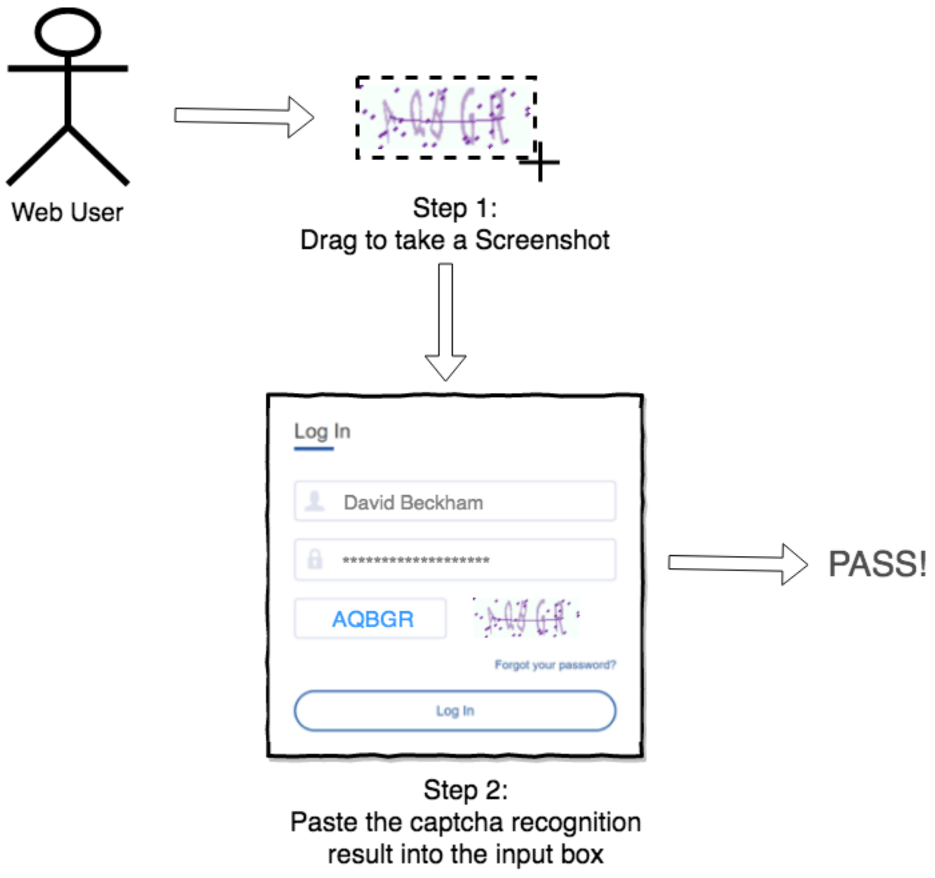


Figure 27. From User's Perspective.

Figure 27 shows that user only needs 2 steps to solve a captcha test with the help of our extension.

As you can see in, from server's perspective shown in **Figure 26**, it involves many steps which makes things seem complicated. However, from the perspective of users in **Figure 27**, it contains two steps only.

3.6. Distributed Server Deployment

As we mentioned above, the captcha verification is widely used in login interface and we cannot control the time when users will encounter a captcha verification. So, our captcha recognition service should be available at any time. In addition, there is a great possibility that large amounts of users need to pass a captcha verification simultaneously, and this require our service has high throughput. In the daytime, we believe that there are more users need to pass captcha verification than those at night and midnight. If servers running at midnight have the same number of which running in the daytime, it would be a waste of energy and resources. So that scaling the application according to the peaks and troughs of users' request would be a good way to save resources, which requires good scalability of the service system.

In order to improve the availability, throughput and scalability of our captcha recognition service, we deploy our captcha recognition server on a distributed computation cluster and choose Kubernetes to manage our captcha recognition application.

Figure 28 shows the architecture of the captcha recognition application. From this figure, we can see that developers interact with the Kubernetes Master node via API server directly. After the Controller Manager and Scheduler processing the task of developer, the Master node interacts with other worker nodes. As we have packaged our application server to a docker image, each pod runs a docker container based on our image. Kuber-Proxy provide Kubernetes network service for our application and the captcha recognition API to our web user interface and chrome extension. For both web interface and chrome extension, they use the same service API, but the requests of recognition would be distributed to several worker nodes, which avoid aggregating all the requests to one single server and leading to network congestion.

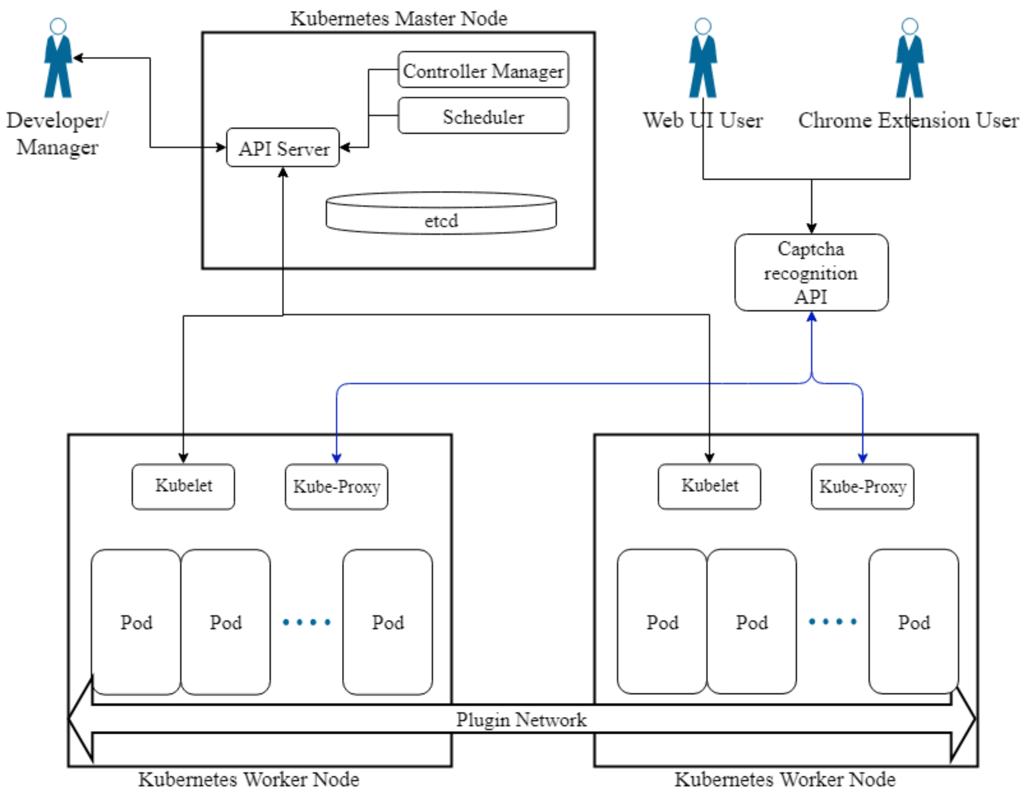


Figure 28. Application deployment architecture

4. Experiment

We trained our model on GPU Farm provided by MSc CS program office. Detailed configuration and environment are shown as follows:

4.1. Hardware configuration

Items	**Details**
System version	Ubuntu 18.04.2 LTS

Memory	57 GB
CPU Info	Intel(R) Xeon(R) Silver 4108 CPU @ 1.80GHz
CUDA / CuDNN	CUDA Version: 10.1 / CuDNN version: 7.3.1
GPU Driver	Driver Version: 418.40.04
GPU Type	GeForce GTX 1080 Ti

Table 3. System hardware configuration of our machine used for training and developing.

Ubuntu 18.04 LTS is one a outstanding system built in with Linux Kernel. It's easy-to-use apt-get tools helps us collect and install software automatically and quickly. 57GB memory is sufficient for complex deep learning model loaded, which allows our team try multiple networks and large training batches. Configuration of GPU Farm provided by CS department is at the cutting edge of education area, installed with latest version of CUDA and CuDNN. GeForce 1080 Ti gives us powerful computing acceleration.

4.2. Software environment

Software	**Version**
Anaconda2	4.6.11
Python	3.6.8

Django	2.2.1
Pillow	6.0.0
Tensorflow-gpu	1.12.0
WhiteNoise	4.1.2
Opencv-python	4.1.0.25
Captcha	0.3
PyYAML	5.1
Gunicorn	19.9.0

Table 4. Selected software requirements in our project.

We mainly use Tensorflow-GPU in version of 1.12.0, a stable release version which we can easily find solutions on Github when we were encountered with trouble shootings. Anaconda is used to create new Python environments. We use Django 2.2.1 as our server architecture. Captcha library is used to generate training and testing samples.

4.3. Network Model

Optimizers	**Adabound**	**Adam**	**Momentum**	
CNN Networks	CNN5	ResNet	DenseNet	

Recurrent Networks	BLSTM	LSTM	SRU	GRU

Table 5. Several networks and optimizers implemented in our project.

We implement several neural networks, including CNNs and RNNs. CNN5 is a classical network widely used in image recognition. ResNet is an advanced efficient type of CNN which decreases number of parameters while keeping high accuracy. DenseNet is naïve neural network which consists of fully connected layers. We use CNNs to extract features from captcha images and deliver the extracted features to RNNs.

Bi-LSTM is an extended version of LSTM, which combining both forward and backward propagations together, and thus taking both prior and later sequence into account. While LSTM only accepts information from prior sequence. SRU and GRU is initial version of RNN, where advanced structures, e.g. Forget Gate, do not exist.

4.4. Results

Length	**Network**	**Optimizer**	**Learning Rate**	**Acc on Web UI**	**Acc on Extension**
Fixed	CNN5	Adam	0.01	99.2%	
	ResNet	Adam	0.01	98.7%	

Variable	DenseNet + LSTM	Adam	0.01	84.9%	68.3%
	DenseNet + BLSTM	Adam	0.001	85.7%	69.2%
	CNN5 + BLSTM	AdaBound	0.001	95.6%	78.4%
	ResNet + BLSTM	AdaBound	0.001	93.5%	77.9%

Table 6. Experiment results

We did many experiments testing different network models, and list some of the results in Table 6. For fixed-length captcha recognition task, both CNN5 and ResNet achieves high accuracy. That’s because the network was not that complicated and training samples were sufficient. Valuable features are extracted well from CNNs.

For variable-length captcha recognition task, we tried more complex network model combining CNNs and RNNs, since RNN is suitable for solving problems encoded in sequence. Through comparing, we noticed that BLSTM performs better than naïve LSTM, while CNN and ResNet both better than DenseNet. As combined networks have more parameters and more complicated loss functions to fit, the final accuracy is lower than fixed-length models.

Also we tried different optimizers. Both AdaBound and Adam perform better than simple SGD, which we do not list in the table above.

5. Trouble Shooting

5.1. Captcha data collection

A huge amount of data needs to be collected for captcha recognition model training. At the very beginning, we plan to collect captcha data by web crawling. However, it is difficult to collect a big volume of data since the web crawler itself needs to pass anti-robot test. Therefore, collecting data in this way is not efficient enough to satisfy the need of our model training.

Fortunately, we come across an open-source python library named "captcha" as mentioned in *3.3 Data Preparation*. Using this captcha generation package, we can generate a huge amount of training data to feed our captcha recognition model. Moreover, we can use the package to generate captcha image with random text value and variable text length. The open-source captcha library saves us a lot of effort in captcha data collecting.

5.2. Variable-length captcha recognition

An ideal captcha recognition model should be able to handle captchas with variable length characters. In practical, different website has different kinds of captcha test with variable-length characters. A fixed-length captcha recognition model may not satisfy general use. However, the fixed-length recognition model cannot handle variable-length captcha as we expected. In order to solve variable-length captcha recognition, we propose a network model combining CNN with LSTM as discussed in *3.2 Variable-length captcha model design* is very difficult to deal with. Although the recognition accuracy is not as high as fixed-length recognition model, the method we propose in *3.2 Variable-length captcha model design* can successfully handle variable-length captcha.

5.3. Chrome extension post request

As mentioned in *2.6 Extension* and *3.5 Google Chrome Extension – ByeCaptcha*, our chrome extension is designed to interact with captcha recognition server using POST request. In the very beginning, we have no clue of how to deal with POST request in chrome extension. After searching and learning, we find out that we can make use of AJAX technology and implement POST method in extension content file.

After solving POST method implementation, we encounter with another issue which is related to POST data type. At first, response data type is set to default data type JSON. However, it returns an error "Cross-origin Read Blocking (CORB) blocked cross-origin response with MIME type application/json". Problem solved after setting response data type to TEXT.

5.4. Kubernetes migration

In our early experimental phase, we have successfully deployed our server locally. In order to improve the availability, throughput and scalability of our service, we migrate our application server to Kubernetes cluster on cloud.

However, when we tried to run pods with our docker image, we encountered the error of “CrashLoopBackOff”, while the correct status of pods should be “running”. The “CrashLoopBackOff” means that our pod is starting, crashing then starting and crashing again. Usually there are three reasons may cause a pod keep crashing and restarting. First reason is that the application inside the pod is keep crashing, the second one is because some parameters of the pod have been set incorrectly, and the other one is something wrong when building the Kubernetes cluster. After checking our configuration, the setting of the Kubernetes system should be correct. As mentioned before, our application server has been tested on single machine, and we need to start the server manually. According to the tutorial of Kubernetes, the pod will be in running status after execute command “kubectl run application-name --image=image-name --port 8080”, but our application server will not start automatically. We thought this problem caused the Kubernetes system to misjudge the application running in pod is crashed.

In order to run our server before the system misjudging it as crashed, we added “-it /bin/bash” parameter when creating a pod, so that we can access to the command line interface of pod. After starting our Django server manually, we find that pods of our application are all in running status. In addition, there is another way to solve this problem. Based on the docker image we have, we can create a new Dockerfile and append some commands to make the server start automatically after the pod initialized. With the a new Dockerfile, we can update our docker image on docker hub.

5.5. Django settings

Everything goes normal when Django runs in localhost. However, when we run our Django project in Docker container where exposed IP address is not localhost, HTTP response return “400”. 400 Bad Request response status code in the HyperText Transfer Protocol (HTTP) reveals that the server cannot or will not process the request due to something that is perceived to be a client error [17]. After debugging, we find out that ALLOWED_HOSTS in settings.py is not set. ALLOWED_HOSTS is a list of strings representing the arrange of host or domain names that this Django server can serve. It is a security measure to prevent HTTP Host header attacks, which are possible even under many seemingly-safe web server configurations [18]. When debug mode is on, an empty ALLOWED_HOST list only serves for localhost, 127.0.0.1 and [::1]. Now it makes sense why it works well in localhost while does not work in Django container. Since the Django container will be packed into Django image and migrate to Kubernetes, the exposed IP address will be unknow for now. Therefore, we simply set ALLOWED_HOSTS to [*], which means it matches every IP address. Problem solved after ALLOWED_HOSTS setting.

6. Conclusion

6.1. Project highlights and innovations

In this project, we introduce and implement a cloud service of captcha recognition system. We have tried to recognize an image-based distorted words CAPTCHA by convolutional neural network. In order to deal with the variable length CAPTCHA, we also add recurrent neural networks on CNNs. Our model doesn’t need any data pre-processing like cleaning the noisy background of a CAPTCHA image, segmenting the image with rectangular windows and recognizing the individual characters. The model was trained on CAPTCHAs generated by python captcha library. We tried both fixed length and variable length CAPTCHAs. Both give 99.2% and 95.6% accuracy respectively.

After training our model we made an online recognition webpage. Anyone when facing CAPTCHAs, can use our webpage to drag and upload an image from their own computer. After the system recognizing the CAPTCHA results, it returns the most likely results to the user. We also made a Chrome extension version to make our system more user friendly. When anyone come across a CAPTCHA online, can use our extension to trigger the screenshot function and the extension would automatically upload the screenshot content to our system and give the result.

In the end, in order to improve our backend server’s throughput and scalability, we deploy our application server on the Kubernetes cluster, which provides automating application development, scaling and management. Besides that, for high-end users we encapsulate our backend server in a docker image and upload it onto docker-hub. Anyone can download it from docker so that they can utilize our image for developing.

6.2. Recommendation for further investigations

For project integrity considerations, we did not use different CAPTCHA libraries to generate different styles CAPTCHA. After that, we plan to expand the training data by using different CAPTCHA styles including some real data from the Internet. In terms of chrome extension, we plan to further improve its usability. First, after taking screenshot, the system can automatically help user fill in the blank. Moreover, our system can automatically detect where is the CAPTCHA and fill in the blank all by itself so that can avoid users manually triggering screenshots.

Contribution

	Yaodong Wang	Yi Wu	Wenzhao Gao	Manping Liang
Background Research	✓	✓	✓	✓
Literature Review		✓	✓	
Methodology Design	✓			✓
Proposal Writing		✓		
Data Collecting	✓		✓	
Data Pre-processing	✓			✓
Algorithm Design	✓			
Algorithm Optimization	✓	✓	✓	✓
Kubernetes Migration			✓	
Docker Image Design	✓			✓
Django Server Implementation		✓	✓	
Web UI Design	✓	✓		
Extension Design			✓	✓
Interim Report		✓		
Status Presentation PPT Design			✓	✓
Final Report	✓	✓	✓	✓
Oral Presentation PPT Design	✓	✓	✓	✓

References

[1] Von Ahn, Luis, et al. "CAPTCHA: Using hard AI problems for security." International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2003.

[2] Chellapilla, Kumar, and Patrice Y. Simard. "Using machine learning to break visual human interaction proofs (HIPs)." Advances in neural information processing systems. 2005.

[3] Kopp, Martin, Matej Nikl, and Martin Holena. "Breaking captchas with convolutional neural networks." ITAT 2017 Proceedings (2017): 93-99.

[4] Stark, Fabian, et al. "Captcha recognition with active deep learning." GCPR Workshop on New Challenges in Neural Computation. Vol. 10. 2015.

[5] Goodfellow, Ian J., et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." arXiv preprint arXiv:1312.6082 (2013).

[6] Y.LeCun, B. Boser, J. S. Denke, D. Henderson, R. E. Howard, W. Hubbard and L.D. jackel. Handwritten digit recognition with a back-propagation network. 1989.

[7] Chollet, F. (2017). Deep Learning with Python . Manning. ISBN: 9781617294433.

[8] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, Show and Tell: A Neural Image Caption Generator, 20 Apr 2015.

[9] LSTMS. http://deeplearning.net/tutorial/lstm.html

[10] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J.

Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[11] Red Hat, https://www.redhat.com/en/topics/virtualization

[12] Kubernetes(K8s). https://kubernetes.io/docs/home/

[13] Django, https://www.djangoproject.com/

[14] Unyscape, https://unyscape.com/django-a-python-based-free-and-open-source-web-framework/

[15] Chrome Extension, https://developer.chrome.com/extensions

[16] TensorFlow, https://www.tensorflow.org/tutorials/load_data/tf_records

[17] MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400

[18] Django Settings, https://docs.djangoproject.com/en/2.2/ref/settings/