EasyOCR-YOLOv5 based Solver for Google's reCAPTCHA v2

Shubo Sun
UIUC Statistics Department
shubos2@illinois.edu

Yizhen Jia
UIUC Economics Department
yizhenj2@illinois.edu

Vaughn Ellison Hilpp
UIUC Statistics Department
vhilpp2@illinois.edu

***Abstract*—In this paper, we present an OCR-and-Yolov5-based algorithm for object detections on Google's reCAPTCHA v2. This report is organized as follows: We introduce the origins and need for CAPTCHA. We describe the need for research to be done on solving the CAPTCHA's in order for development of new methods to be created to stay ahead of fraudsters. This report will detail our method for solving both alphanumeric and image CAPTCHA's and the data that is behind it.**

I. Introduction

CAPTCHA, abbreviated for "Completely Automated Public Turing test to tell Computers and Humans Apart", is a computer test for distinguishing between humans and robots. Since the invention of Internet and digital marketplaces in the early 1990s, there was a growing need for fraud protection against hackers armed with their own computers. These hackers attacked computer systems and posted about sensitive information or scams that were being automatically monitored by keyboards. A need for a system to disseminate between legitimate users and hacking bots grew and grew. By requiring all users to pass the CAPTCHA authentication, administrators of web pages can filter out spammers who plan to automate their activities.

Hackers and fraudsters do not rest and are constantly inventing genius software to bypass CAPTCHAs. If researchers can come up with a way to beat the current iteration of CAPTCHA, it signals a time for change in the type of test being used. This is the reason alphanumeric tests were replaced with image tests as images are more challenging to solve. Our motivation for solving CAPTCHA is to further identify the shortfalls of the current imaged-based CAPTCHA implementations in order for developers to patch the metaphorical holes in the system to stop bot attacks.

In this report, we propose a brand new network that is fast, easily trainable, and capable of creating arbitrary partitions of the text input feature space to identify the objects within Google's reCAPTCHA v2. Specifically, our model was built based on cv2, Optical Character Recognition (OCR) and YOLOv5 algorithms. The cv2 template matching is firstly used to pass the checkbox of "I'm not a robot." The OCR is to correctly identify the words lying on the top of reCAPTCHA images, and YOLOv5 is to perform object detections within each image. To make it easy, We used the EasyOCR [11] Python library to handle the OCR framework. Fig. 1. shows an example of a 3x3 reCAPTCHA v2. and Fig. 2. shows an example of a 4x4 reCAPTCHA v2.

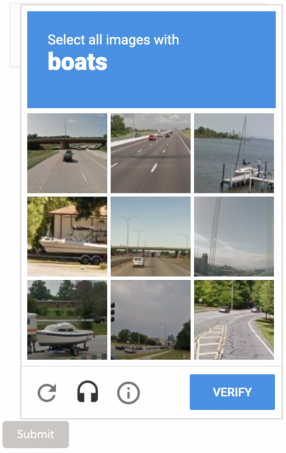

Fig. 1. An example of 3x3 reCAPTCHA v2. EasyOCR is used to identify and record "Select all images with" and "boats" on the up-left corner. EasyOCR will learn the task is to choose grids with boats. Then, our proposed YOLOv5 network will detect if boat(s) exist in any of the grids.

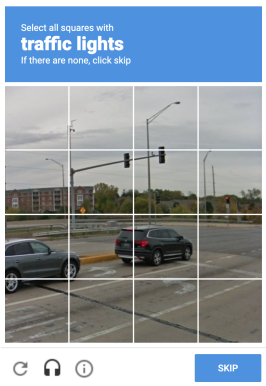

Fig. 2. An example of 4x4 reCAPTCHA v2. Different from 3x3's, our network will detect if traffic light(s) exist in the whole image.

II. Related Works

While there are many works focusing on text-based captchas, works to solve imaged-based captchas are limited.

For previous works on all versions of reCAPTCHA, Akrout et al. [1] mimicked human's mouse movements during the "I'm not a robot" stage to bypass reCAPTCHA through reinforcement learning. Goodfellow et al. [2] proposed an 11-layer CNN with a softmax layer to identify the numbers appearing on each ReCAPTCHA image. Sivakorn et al. [3] developed a tag classifier to attack the oldest version of reCAPTCHA through analyses on users' browsing history and websites restriction. Zhao et al. [4] presented an attack system to all versions of reCAPTCHA based on CNN and RCNN. Zhou et al. [5] used GoogLenet for image classifications of reCAPTCHA v2. Wang et al. [6] and Hossen et al. [7] performed object detections on reCAPTCHA v2 with YOLO v3, and [6] also mimicked human's mouse movements to pass "I'm not a robot" stage. See Fig. 3. as an example of this.

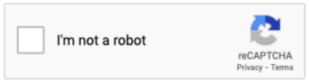

Fig. 3. An example of "I'm not a robot".

In addition to image-based CAPTCHA solving, the construction of adversarial image-based CAPTCHA networks has become one of the most popular fields, see Acien et al. [8], [9] for proposed GANs to protect BECAPTCHA from robots' attack.

Different from previous works, we are the first to combine the cv2, YOLOv5 and EasyOCR algorithms for object detections on image-based CAPTCHAs. While some of the papers [6], [7] also used YOLO-based algorithms, their models were YOLOv3-based and did not contain cv2 or EasyOCR.

III. Model Architecture

In this section, we propose an algorithm for objection detections on reCAPTCHA v2. Our model contains a checkbox locating algorithm using cv2 template matching, a text detection network using EasyOCR [11] and an objection detection network using YOLOv5s [10] on reCAPTCHA v2 images. YOLOv5s is an updated version of YOLOv5.

Specifically, our model is organized in three steps. For the first step, we use the template matching in Python cv2 library to locate the checkbox of "I'm not a robot" and click on this checkbox based on brute-force search. The position and the size of the checkbox never change, which means the brute-force search can always locate the checkbox.

For the second step, We identify the task type (3x3 or 4x4) through a self-trained YOLOv5s network. For the third step, we take a screenshot of the webpage and identify the text above the reCAPTCHA images using EasyOCR. Learning what the target is, our network performs object detection using two YOLOv5s models. Each YOLOv5s model is applied to different classes and We will explain this in Section V detailedly. Also, YOLOv5s will detect objects within each grid for the 3x3 tasks directly, and integrate the 16 grids to detect objects within the whole image for 4x4 tasks. Our model will repeat the last two steps until all target objects are located. Fig. 4. offers an overview of how our model functions.

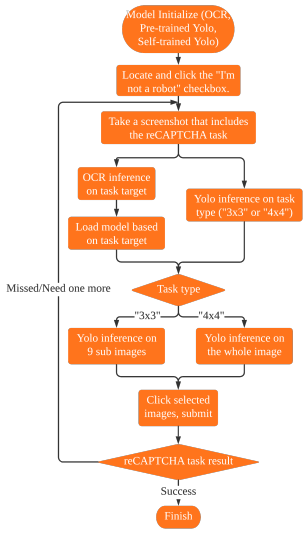

Fig. 4. A flowchart of our network. We first pass "I'm not a robot" checkbox using cv2 template matching. Then, we use YOLOv5s to determine the reCAPTCHA task type (3x3 or 4x4). Finally, we perform text detection to learn the class target above reCAPTCHAs using EasyOCR, and detect and localize the objects based on the task types using YOLOv5s. Note that this process will repeat until all target objects are detected and located.

A. EasyOCR

We perform text detections through EasyOCR to learn the target category in reCAPTCHAs for further object detections. See Fig. 5. for an overview of the EasyOCR framework.

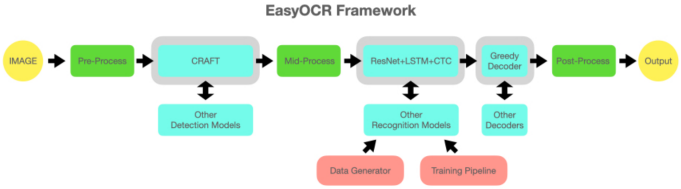

Fig. 5. The framework of EasyOCR. EasyOCR locates texts through CRAFT,

After the input image pre-processing, EasyOCR first locates the target text using CRAFT [16] and record the bounding boxes. With some mid-processing of data, a combination of trained Resnet and bidirectional LSTM modules is applied to recognize the located texts using the CTC [17] loss. Since CTC loss can generate a different length of the output texts compared to the input texts, EasyOCR uses an decoder to remove duplicated and blank characters for better text prediction accuracy. We have shown how CTC works in the progress report. The output of EasyOCR is a list of the texts as well as the coordinates of their bounding boxes.

B. YOLOv5

We trained a 126-layer YOLOv5s model for object detections. YOLOv5 consists of three sections: Backbone, Neck and Output. Fig. 6 provides an overview of YOLOv5.

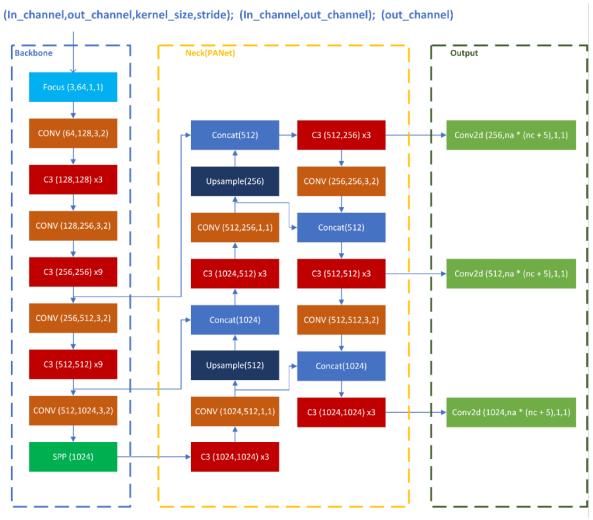

Fig. 6. The architecture of YOLOv5s. It consists of three parts: (1) Backbone: CSPDarknet53, (2) Neck: PANet, and (3) Head: Yolo Layer. The data are first input to CSPDarknet53 for feature extraction, and then fed to PANet for feature fusion. Finally, Yolo Layer outputs detection results (class, score, location and size). One C3 block consists of three convolutional layers and bottleneck modules, and a "$\times n$" means this C3 has n bottleneck modules. SPP is a pooling layer that removes the fixed-size constraint of the network. Our YOLOv5s has a similar structure to YOLOv5 while the number of bottleneck layers within each C3 block is different.

1) Backbone: The Backbone section is a CSPDarknet53 [12] network for feature extraction. It contains 4 regular convolutional layers connected by C3 blocks. The first C3 block contains 3 convolutional layers followed by 3 bottleneck layers; the second C3 block contains 3 convolutional layers followed by 7 bottleneck layers; the last C3 block contains 3 convolutional layers followed by 11 bottleneck layers. Note that although YOLOv5s and YOLOv5 are different in the number of bottleneck layers within C3 blocks, both their total number of bottlenecks in the Backbone section is 21. Each bottleneck layer consists of two convolutional layers with the same number of output channels as input channels. These bottleneck layers can reduce the number of neurons, and compress feature representations to best fit in the network and get a smaller loss during training. Followed by each regular convolution layer and bottleneck layer is a Sigmoid Linear Unit [13] (SiLU) activation function. For SiLU, if we borrow the expressions from Multilayers Perceptrons, and suppose $\mathbf{z}^{(i)} = \mathbf{W}^{(i)}\mathbf{a}^{(i-1)}$ is the output at layer i, then the output with activation is

$$\mathbf{a}^{(i)} = SiLU(\mathbf{z}^{(i)}) = \mathbf{z}^{(i)}\sigma(\mathbf{z}^{(i)}). \tag{1}$$

In this case, we can see SiLu as a scaled version of the Sigmoid function. For kernels, strides and paddings, we first start with a convolution layer with a kernel size of 6 and a stride and padding of 2 to transfer the 3 channels' RGB into 64 output channels. After we dive into the Backbone, all the regular convolutional layers have a kernel size of 3, a stride of 2 and a padding of 1. For convolutional layers in a C3 block but not in a bottleneck, the number of its output channels decreases by 2 under a kernel size of 1 and a stride of 1. For the two convolutional layers inside the bottleneck, the first one has a kernel size and stride of 1, and the second one has a kernel size of 3 with a stride of 1.

YOLOv5s concatenates the 2^{nd} and the 3^{rd} C3 blocks in the Backbone section with layers in the Neck section respectively for feature fusion. The ending of Backbone is a Spatial Pyramid Pooling [14] (SPP) block containing 2 convolutional layers and a 1-dilated maxpool layer with a kernel size of 5 and a padding of 2. SPP can maintain the spatial information from previous layers, and adjust the output to a fixed size that works as the input for the starting layer in the Neck section.

2) Neck: The Neck section is a Path Aggregation Network [15] (PANet) aiming at feature maps collection and object localization in images. PANet first extracts features through bottom-up path augmentation which shortens the information path to make PANet less computationally expensive, and improves the feature fusion with a "pyramid" structure. See Fig. 6. for this "pyramid" structure: in the left of the Neck section, the bottom convolutional layer has an input size of 1024, while the top convolutional layer has an input size of 512. PANet will also concatenate the processed feature maps with C3 blocks from the Backbone section to achieve better object localization.

The Neck has a structure as follows: 2 C3 blocks are followed by a regular convolutional layer with an up-sampling layer to concatenate with layers from the Backbone. After the up-sampling layers, it has three C3 blocks connected by two convolutional layers. The C3 blocks in the Neck section have the same hyperparameters and are designed the way as those from the Backbone section. For the regular convolutional layers, The left ones have a kernel size and a stride of 1, and the right ones has a kernel size of 3 and a stride of 2. After concatenations between layers from the Neck section, our model moves on to the Head section.

3) Head: The last section is the Head section for prediction, which is composed of a vector containing the coordinates of the predicted bounding box (center, height, width), the confidence score of the prediction and the label. Based on different concatenations from the Neck section, the predictions were performed on parallel convolutional layers with three different detection heads (input size). As can be seen in Fig. 6., the second detection head is twice the size of the first detection head, so it is better able to detect medium objects. Similar conditions can be applied to the third detection head to detect small objects.

C. Other Hyperparamters and Loss Function

Our loss functions are Binary Cross Entropy (BCE) loss with logits and a bounding box regression loss proposed by Song et al. [18]. The BCE loss with logits combines a Sigmoid layer and the BCE loss, and outperforms the single BCE loss for multi-label classifications. The bounding box regression loss outperforms regular IoU loss due to the consideration of the overlapping directions of the bounding boxes.

For hyperparameters, we trained our model with a learning rate of 0.01 and a weigh decay of 0.0005. Our model ran over 300 epochs using the SGD optimizer with a batch size of 64 and predict the bounding boxes with a 0.2 IoU threshold. For the remaining hyperparameters, we keep the same as defaults in [10].

IV. Data

We have two sets of image data. The first set is the *Task classification set*. We accessed to the reCAPTCHA v2 through Google developer tools, screenshotted multiple times to get 329 images and divided them into 3x3 and 4x4 tasks. These collected images were originally in 1920x1080 resolution but randomly cropped before the data augmentation. The second *bbccps* set contains reCAPTCHAs from Kaggle [19], [20] for boats (307), bridges (302), chimneys (294), crosswalks (205), palm trees (306) and stairs (278). The resolutions of *bbccps* vary since these images are from different Kaggle webpages.

We performed a 70%/20%10/% split for Train/Dev/Test data. Table I displays our Train/Dev/Test splits for these sets. We also performed different data augmentations for each set. Table II. provides a detailed explanation of data augmentations. For samples of each set after data augmentations, see Fig. 7. and Fig. 8. for *bbccps* and *Task classification set* respectively. We also labeled each all images. Fig. 9. provides graphical summaries of the two sets with label embeddings.

TABLE I
Number of Images

	Bundle Types	
	Task classification set	***bbccps set***
Training (70%)	230x3 (for augmentations)	1,181x3
Validation (20%)	66	337
Test (10%)	33	169
Total	789	4,049

Fig. 7. bbccps samples.

TABLE II
Data Augmentations

Bundle Types	
Task classification set	***bbccps set***
Random crop (0∼10% zoom in)	Flip (horizontal)
Grayscale on 5% of images	Random crop (0∼20% zoom in)
Hue (-20° ∼ +20°)	Rotation (-11° ∼ +11°)
Blur (up to 0.75 px)	Shear (±10° Horizontal, ±10° Vertical)
Noise (up to 2% of pixels)	Exposure (-11% ∼ +11%)

The blur and the noise were also applied to the bounding boxes.

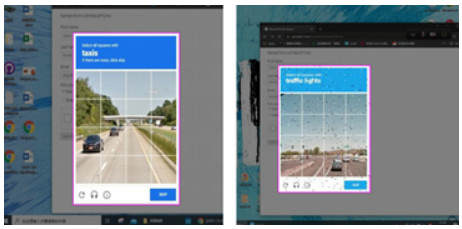

Fig. 8. Task classification set samples.

V. Experimental Designs and Results

Depending on different characteristics of the classes of our images, we applied different weights initialization methods. For buses, taxis, cars, bicycles, traffic lights, tractors, fire hydrants, and motorcycles, we utilized the pretrained YOLOv5s weights, while for boats, bridges, crosswalks, chimneys, palm trees and stairs, we self-trained the YOLOv5s. Also, we considered taxis as cars since taxis' shape is similar to cars'. We discarded the "mountains or hills" class as their features were hard to capture.

A. Experiments on the task classification model

As we have stated in Section III, we self-trained a YOLOv5s task classification (e.g. if it is 3x3 or 4x4) model. we examine the performance of this model below.

Fig. 10. shows the loss and mean average precision (mAP) history for the 3x3 or 4x4 task classification model. The loss, precision, recall, and the mean average precision become

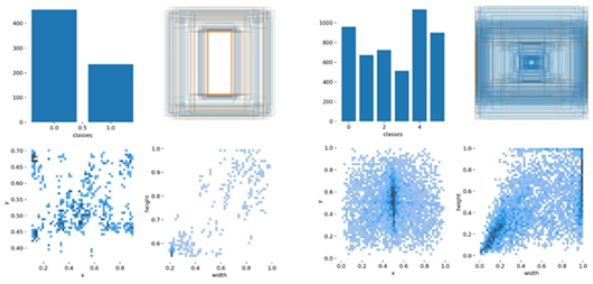

Fig. 9. The Graphical summaries of labels for *Task classification set* (left) and *ccbbps* training set (right). For each summary: (Upper Left) Histogram of classes, (Upper Right) Label shapes, (Bottom Left) Label centers (relative), (Bottom Right) Label width-height (relative) graph.

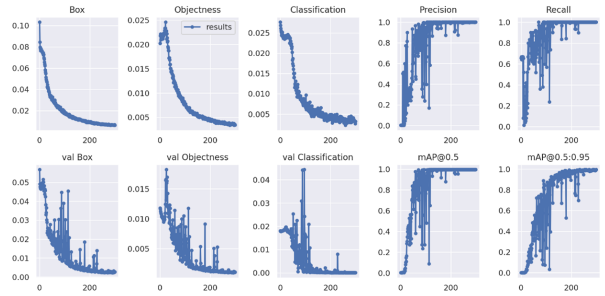

Fig. 10. Loss & mAP over epochs for task classifications.

stable after 200 epochs. Training on 300 epochs, we find the prediction on the test data has a precision of 0.999 and a recall of 1 for both 3x3 and 4x4. The mAP is approximately 0.996 under an IoU threshold of 0.5, and the average mAP is also close to 0.996 when we vary the threshold from 0.5 to 0.95. This suggests our self-trained task classification model works well. Fig. 11. shows some sample output of this model. Table III presents the performance of this model on the test data.

Fig. 11. Sample output of our task classification model.

TABLE III
PERFORMANCE OF TASK CLASSIFICATION MODEL ON THE TEST DATA

Classes	Images	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Both	66	0.999	1.0	0.996	0.996
3x3	36	0.998	1.0	0.996	0.996
4x4	30	1.0	1.0	0.996	0.996

B. Experiments on the object detections model

In this section, we examine the performance of our self-trained YOLOv5x model on object detections. Surprisingly, our self-trained model behaves decently for some classes with data augmentations on *bbccps* reCAPTCHAs. Fig. 12. shows the loss and mAP history of our model.

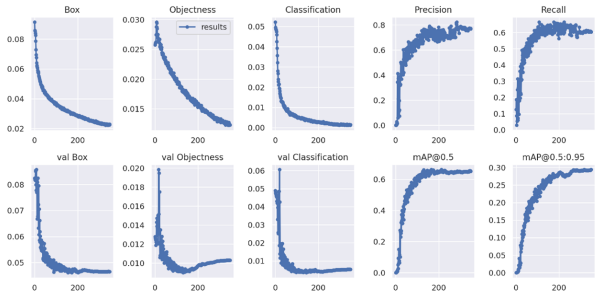

Fig. 12. Loss & mAP over epochs for object detections.

TABLE IV
PERFORMANCE OF OBJECT DETECTION MODEL ON THE TEST DATA

Classes	Images	Precision	Recall	mAP@0.5	mAP@0.5:0.95
All	454	0.768	0.604	0.652	0.294
Boats	80	0.885	0.625	0.695	0.318
Bridges	57	0.976	0.719	0.890	0.433
Chimneys	82	0.854	0.780	0.860	0.419
Crosswalks	52	0.669	0.596	0.578	0.224
Palm trees	114	0.725	0.526	0.57	0.262
Stairs	69	0.496	0.377	0.319	0.110

Fig. 13. Sample output of our object detection model.

For our model's performance on *bbccps*, the precision, recall, and the mean average precision become stable after 200 epochs. Training on 300 epochs, we achieve a precision of 0.768 and a recall of 0.604 on the test data for all classes. The mAP is 0.652 with an IoU threshold of 0.5, but the average mAP is only 0.294 when the threshold varies from 0.5 to 0.95. In other words, the mAP decreases significantly when the threshold increases. This is also shown in Fig. 13.

for a sample output that our model only has a 0.4 confidence score for one of the crosswalks. Our model even missed one crosswalk image.

The reason for this unsatisfactory result might relate to the dataset size. We only have about 300 images for each class in *bbccps* set, whereas an ideal number would be more than 1,500 images per class. Also, the characteristics of some classes might be hard to distinguish. This can be seen from Table IV that the mAP for stairs is very low.

C. Algorithm discussion

1) Click compensation improvement: The 3x3 reCAPTCHAs have two sub-types: "select all at once" and "click until none left". For the former type, we noticed that it always contains 3 correct images out of 9, and requires users to select at least 3 images before submission. However, we cannot guarantee our algorithm will select the exact 3 correct images from 9 images. Sometimes, our algorithm can only detect fewer correct images, and we have to pick some other images to meet the submission requirement. We call this process "click compensation". Currently, the click compensation is random, and we will improve the click compensation in the future. For example, when an image shows a person on a bicycle, the models may only detect the person but not the bicycle. In this case, if our algorithm only selected two correct images but missed this special case, the click compensation will include the missed one by searching for images that contain persons. This can improve the performance of our algorithm.

2) Algorithm performance: We use the success rate as the measurement currently; however, we realized that our algorithm's performance heavily relies on YOLOv5's performance. Even if the algorithm clicked on all correct images, the reCAPTCHA system may still ask us to complete another round of tasks. This happens due to the design of reCAPTCHA - preventing robots or massive queries in a short time. If reCAPTCHA has noticed that we tried too many times in seconds, the difficulty for our algorithm to finish tasks will increase drastically. In this case, we cannot come up with a proper measurement to evaluate the performance of our algorithm.

VI. Conclusion and Future Work

In this project, we developed an algorithm to solve Google reCAPTCHA v2 tasks. We utilized three different techniques: checkbox pass using cv2 template matching, text detection using EasyOCR and object detection using YOLOv5. For YOLOv5 implementations, we collected images from Kaggle and our screenshots, and labeled them for training. We came up with some minor techniques such as "click compensation" to improve the algorithm performance. Our algorithm can solve more than half of the reCAPTCHA v2 tasks smoothly, but some further improvements are necessary for our algorithm to become a well-developed one.

To discuss what we have learned, this project improves our Python programming skills a lot. We also have a better idea about some advanced OCR or object detection techniques. More importantly, by investigating on reCAPTCHA v2's images and design, we are clearer on how anti-robot systems work. A specific experience would be denoising is not always a good choice before doing the object detection. For example, the model can predict the traffic sign in a noisy image successfully, but this sign could be predicted as a clock if denoised. This may not be a big issue in our project; however, it will be a disaster when this happens to an autopilot algorithm.

Due to the time limits, it is hard for us to label more data for training. For instance, our labels of taxis as cars can lower the algorithm performance. In this case, collecting and labeling more classes as well as more images for each class to improve the self-trained model is a reasonable improvement for this project because this will increase the depth and width of the dataset).

Also, the limited computational power restricts our model. Specifically, training a more complex YOLOv5x model may give us a better result but it will take too much time (we have to focus on other courses' projects and exams). For instance, On an Nvidia Tesla P100-PCIe-16GB GPU, training a YOLOv5s model with our *bbccps* dataset only takes about 20 seconds per epoch, while training a more complex YOLOv5x model takes around 2.3 minutes per epoch. Once the computational power issue is solved, we can improve our algorithm performance by having an even larger model. What is more, coming up with an idea to predict mountains and hills (recall that our algorithm gave up this class in Section V), improving the click compensation module, and coming up with a proper algorithm performance measurement will be really helpful for this project.

References

[1] Ismail Akrout, Amal Feriani and Mohamed Akrout, "Hacking Google reCAPTCHA v3 using Reinforcement Learning," arXiv:1903.01003.

[2] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet, "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks," arXiv:1312.6082.

[3] S. Sivakorn, J. Polakis, and A. D. Keromytis, "I'm not a human: Breaking the google recaptcha," Black Hat,(i), pp. 1–12, 2016.

[4] Binbin Zhao, Haiqin Weng, Shouling Ji, Jianhai Chen,Ting Wang, Qinming He, Raheem Beyah. 2018. Towards Evaluating the Security of RealWorld Deployed Image CAPTCHAs. In AISec '18: 11th ACM Workshop on Artificial Intelligence and Security Oct. 19, 2018, Toronto, ON, Canada. ACM, NY, NY, USA, 12 pages. https://doi.org/10.1145/3270101.3270104

[5] Yuan Zhou, Zeshun Yang, Chenxu Wang, Matthew Boutell, "Breaking google reCaptcha V2," Journal of Computing Sciences in Colleges, Volume 34, Issue 1, October 2018, pp 126–136.

[6] D. Wang, M. Moh and T. Moh, "Using Deep Learning to Solve Google reCAPTCHA v2's Image Challenges," 2020 14th International Conference on Ubiquitous Information Management and Communication (IMCOM), 2020, pp. 1-5, doi: 10.1109/IMCOM48794.2020.9001774.

[7] Md Imran Hossen, Yazhou Tu, Md Fazle Rabby, Md Nazmul Islam, Hui Cao, Xiali Hei, "An Object Detection based Solver for Google's Image reCAPTCHA v2," arXiv:2104.03366.

[8] Alejandro Acien, Aythami Morales, Julian Fierrez, Ruben Vera-Rodriguez, "BeCAPTCHA-Mouse: Synthetic Mouse Trajectories and Improved Bot Detection," arXiv:2005.00890.
[9] Alejandro Acien, Aythami Morales, Julian Fierrez, Ruben Vera-Rodriguez, "BeCAPTCHA: Detecting Human Behavior in Smartphone Interaction using Multiple Inbuilt Sensors," arXiv:2002.00918.
[10] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, Ayush Chaurasia, TaoXie, Liu Changyu, Abhiram V, Laughing, tkianai, yxNONG, Adam Hogan, lorenzomammana, AlexWang1900, Jan Hajek, Laurentiu Diaconu, Marc, Yonghye Kwon, oleg, wanghaoyang0106, Yann Defretin, Aditya Lohia, ml5ah, Ben Milanko, Benjamin Fineran, Daniel Khromov, Ding Yiwei, Doug, Durgesh, and Francisco Ingham. ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations, Apr. 2021, https://github.com/ultralytics/yolov5
[11] rkcosmos with 67 contributors in total, rkcosmos/EasyOCR: Version 1.4.1, 11 September 2021, https://github.com/JaidedAI/EasyOCR
[12] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," arXiv:1911.11929.
[13] Stefan Elfwing, Eiji Uchibe and Kenji Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning," arXiv:1702.03118.
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 37(9):1904–1916, 2015.
[15] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia, "Path Aggregation Network for Instance Segmentation," arXiv:1803.01534.
[16] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee, "Character Region Awareness for Text Detection," arXiv:1904.01941.
[17] Alex Graves, Santiago Fernandez, Faustino Gomez, Jurgen Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," Appearing in Proceedings of the 23 rd International Conference on Machine Learning, Pittsburgh, PA, 2006.
[18] Qisong Song, Shaobo Li, Qiang Bai, Jing Yang, Xingxing Zhang, Zhiang Li and Zhongjing Duan, "Object Detection Method for Grasping Robot Based on Improved YOLOv5," Micromachines 2021, 12, 1273. https://doi.org/10.3390/mi12111273.
[19] https://www.kaggle.com/alerem/recaptcha
[20] https://www.kaggle.com/benjaminmissaoui/recaptcha-data