CS230

Automating Simple Web Tasks using Deep Learning

Mohammad Mahbubuzzaman
Department of Computer Science
Stanford University
tachyon@stanford.edu

Abstract

In this project, we aim to build an AI system that can automate simple web-browser based tasks using deep learning based models. In addition to assisting people with disabilities or limited computer skills, it can enable users to automate repetitive tasks that are difficult to automate via traditional tools. Existing automation approaches rely on custom software and interfaces. Our system will utilize a combination of deep-learning based computer vision and language processing models to automate tasks without requiring website specific software engineering. We break down the problem in two major tasks. First task is to take rendered web-pages as images and apply scene understanding approaches to make sense of the content. Next, it will create an action plan based on the scene and given task description utilizing both computer vision and natural language understanding.

1 Introduction

Automation in computer typically relies on modules interacting via agreed upon protocols - which requires support from the system that is being automated. We refer to this type of automation as "*traditional automation*" or "*programmatic automation*" in the rest of this paper. In this project, we approach automation from the perspective of a human as opposed to that of a system as in traditional automation. People use computers by understanding the graphical output rendered on the screen and by interacting with it via peripherals such mouse and keyboards. For many problems, programmatic automation makes sense as it can execute fast and reliably. However, there are scenarios where traditional automation is prohibitively difficult, expensive and not scalable. Following are two specific instances that highlight such use cases.

For the first example, consider the task of logging in to a site. A human with moderate computer literacy can log in to a new site without any explicit instructions, even though the exact location of the login widgets or the user interface is not known beforehand. However, for an inexperienced user or user with disabilities who needs to minimize manual work, it can be useful if the system does it for them. Implementing such an automation via traditional approaches is practically impossible because each site is free to implement the login interface in their own way.

For the second category, consider web tasks that are moderately complex. In this scenario the user can be experienced but the challenge is that the task needs be performed for a rather large number of sites. Take for example, the task of updating passwords for a list of websites, where the user have accounts and these sites have been compromised or the passwords used are weak and at risk of being compromised. This can be a tedious exercise and something that is challenging to automate using programmatic approaches for the same reasons outlined above.

For both of these scenarios and a number of other broad use-cases, an AI assisted automation for web tasks or computer tasks in general can be a useful tool. To limit the scope of this project we aim to

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

automate one such web task as a proof of concept. In future we can extend the scope to include more common tasks as well as contexts outside of web browsers.

For this project, the input will consist of a) Stream of images capturing the rendered screen for web pages and b) A desired high level task such as "Login" or "Update password". In terms of output, the scope for this project will be limited to generating a sequence of unambiguous, elementary tasks that are environment and instance specific. This could be actions like "Move mouse to coordinate x, y", "Left Click" etc. Note that, these tasks are to be fed into an execution engine that can perform the tasks and these are not meant to be human readable. This execution engine is not part of the scope of this project.

2 Related work

While we are yet to find any existing work that significantly matches with our problem description, a recent work[1] claims to use natural language interface, to map user commands into website interactions. To train and evaluate, the dataset used is limited to a handful of websites. There are several other works [2] that could be potentially related which we are still going through.

3 Dataset and Features

We are utilizing the Roboflow Website screenshot dataset `https://public.roboflow.com/object-detection/website-screenshots` for this project. It is a dataset composed of screenshots from over 1000 of the world's top websites. It consists of 1206 images with a total of 54, 215 annotations. Original images are of resolution 1024x768. All images are pre-processed and exported to Coco format. These images have been automatically annotated to label the following 8 classes:

1. button - navigation links, tabs, etc.
2. field - to enter for search or comment
3. heading - text that was enclosed in <h1> to <h6> tags.
4. iframe - ads and 3rd party content.
5. image - <img>, <svg>, or <video> tags, and icons.
6. label - text labeling form fields.
7. link - inline, textual <a> tags.
8. text - all other text.

Among above eight classes, original dataset has under represented data for classes: 'label', 'iframe'. After baseline, we plan to look at the dataset and improve annotations for those two classes. We might also need to add more images to increase representation of the above two classes as well as experiment with augmented images of varying orientations.

3.1 New Dataset

This project requires one more dataset for training the classifier on login page screenshots. This is something that had to be manually created. We individually downloaded about 150 images of login screens using google search and used them as positive examples of the login webpage classification problem. A subset of this newly created dataset is available here Sample Dataset

4 Methods

Code https://github.com/tachyon77/web$_automation$

Figure 1 gives a high level view of the architecture. The intuition behind our algorithm is based on two key observations. To understand a given webpage we make use of three types of information. **First** we look at the visual layout of the UI elements - such as what kind of UI element it is, whether a textbox, and image or link and so on. **Second**, we consider how the elements are placed relative to each other. A text appearing closer to a textfield is likely associated with the textfield whereas a text

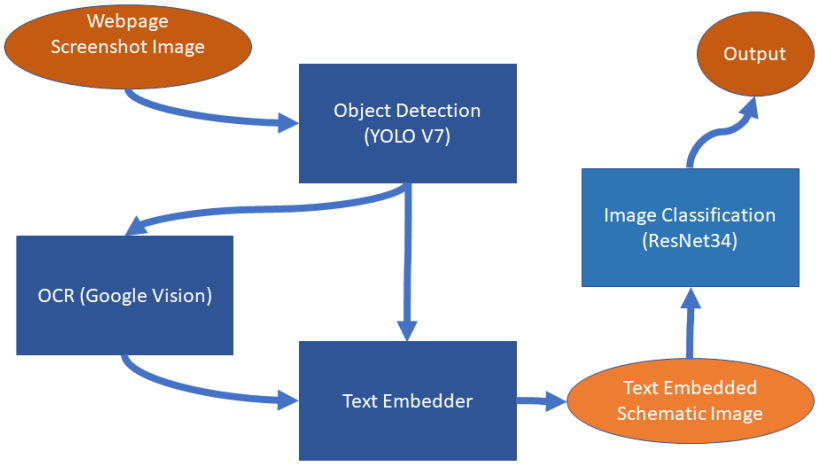

Figure 1: Algorithm

appearing further away in the page is probably unrelated to the textfield in question. **Third** we read any textual content that is associated with an element.

Our model uses all three of these signals. The first two types of signals are solvable with object detection and localization models which can tell us what kind of UI element it is and what are its spatial properties. For this we use state of the art segmentation and object detection algorithm Yolo v7. We start with a pretrained Yolo v7 model trained on standard COCO dataset. However, since our training classes are different and not part of common natural objects, we fine tuned this pretrained model using a dataset called RoboFlow webschreenshot which is available freely.

For the third type of signal, which is textual information, we used an Optical Character Recognition (OCR) model. In particular, we utilized Google's cloud vision api to get textual content from a given image. To feed into this step, our previous step crops individual elements of UI using the bounding boxes produced by YOLO.

After we have detected the UI elements with their bounding boxes and also detected any text present on them, we employed a novel idea to fuse these two types of information. The intuition is that, NLP based algorithm have no notion of visual or spatial context. In this case two different context matters simultaneously. It not only matters what the text is; but it also matters where in the scene it is located and what other elements are int the neighborhood. For cases where the spatial locality is important while absolute positions do not matter (a login box can be anywhere on the screen), the type of algorithm that is suitable is CNN based models such as Resnet. So, to combine these two types of information - textual content and spatial layout, we create a new feature representation using object localization as well as OCR output. Since Resnet needs an image like input, we embed the textual content right at the place they are detected. We also blank out the image everywhere except the UI elements that are detected. The benefit of extracting text and writing them back again is that, in this process, all the font, color and other visual attributes of the text is stripped out, keeping a constant visual look for a given text. This should help the classifier by not having to deal with all those additional visual features, which we can argue safely that they are not relevant for scene understanding. And the point of treating the text content inside an image feature as opposed processing with traditional NLP type layers is that, this way we retain the spatial context of the text.

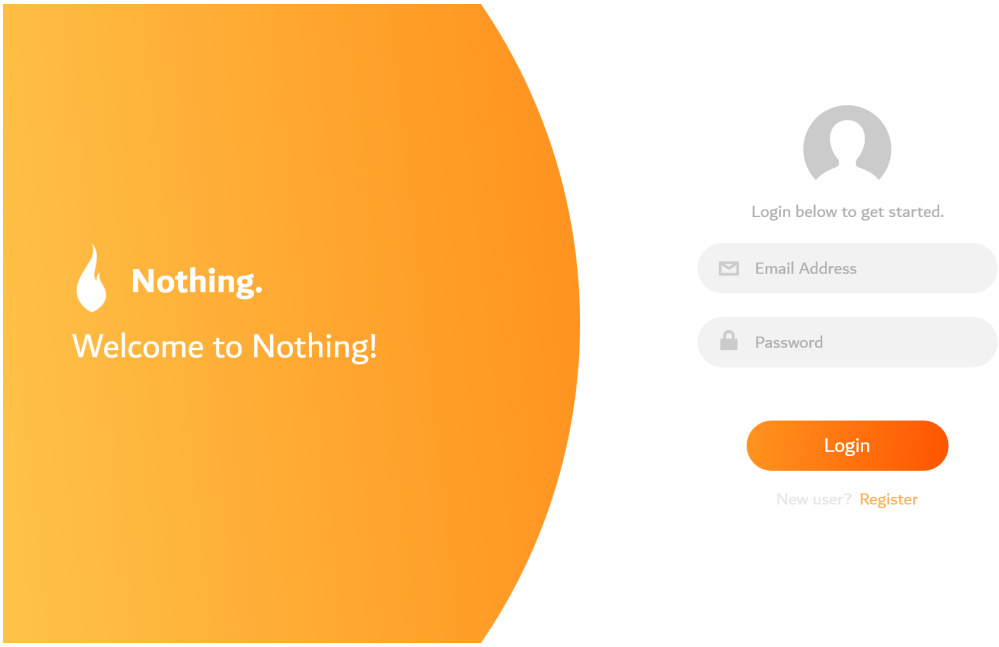

Figure 2: An input image

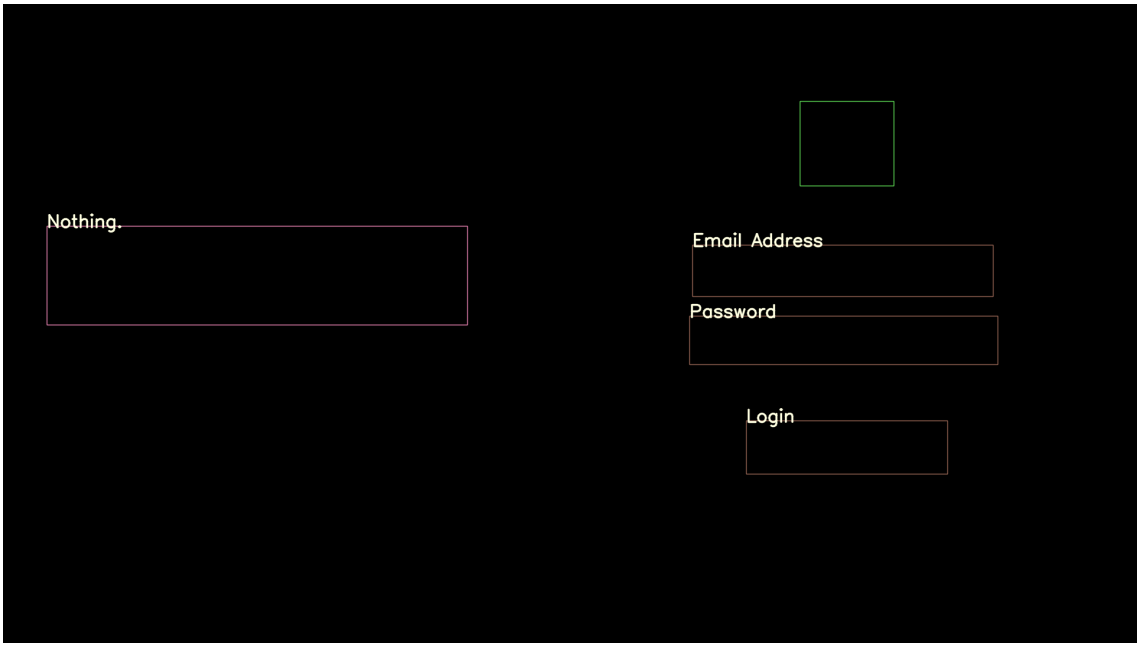

Figure 3: New Feature Representation for use in Resnet

5 Results

Following is an example of how the output of Text Embedder step look, after UI elements are detected with bounding boxes and OCR detected text is written back in place. Figure 3 shows an actual example of what is fed into Resnet34.

Since we used a very tiny dataset, our model was able to converge rapidly with high accuracy. After only 10 epochs we achieved:

Scenario	Loss	Accuracy
Train	0.09	97.69
Validation	1.10	94.44

Please note, figure 4 and 5 are from the training of the YOLO v7 model with the Roboflow dataset.

For test statistics we were not able to finish the implementation of infrastructure to collect the data programmatically. However, we created a dataset of 53 unseen examples, and manually looked on the inference output for them. Initial study suggests, our model has an accuracy of at least 80%.

We also experimented multiple existing OCR models include tesseract and found most of them to be having an unsatisfactory accuracy rate. Google cloud vision did the best but had some smaller issues as well which we applied some fixes to work around.

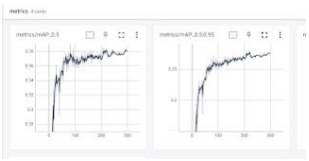

Figure 4: mAP 0.5 and mAP 0.5:0.95

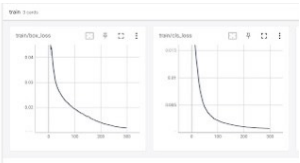

Figure 5: train box loss | train class loss

6 Conclusion/Future Work

Scene understanding in a global context is a challenging and interesting problem. In this project we are essentially addressing scene understanding but with a limited scope. Given, the input space is more structured, we expect this effort to result in a practically viable software that can perform simpler tasks. In future we would like to experiment with other browser based tasks, possibly of higher complexity. The vision is to build a solution that an end user can use within a browser for any website. If we have more resources, we would like to complete the remaining pieces of the complete system including integration of screen capturing and mouse and keyboard automation. A more ambitious goal is to have a hub/website of trained models that can users can download as plug and play modules that automates various specific tasks. Or take it even further to be able to train models on new tasks in a distributed setup, by real users while they perform activities in real-time.

7 Contributions

Mohammad Mahbubuzzaman is the sole contributor to this project.

References

[1] Sahisnu Mazumder and Oriana Riva. Flin: A flexible natural language interface for web navigation. 2021.

[2] Jieshan Chen, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. Object detection for graphical user interface: Old fashioned or deep learning or a combination? 09 2020.