Automatic Image Labelling System

Jing Peng
UNI: jp4081
jp4081@columbia.edu

Jiashu Chen
UNI: jc5664
jc5664@columbia.edu

Yi Yang
UNI: yy3089
yy3089@columbia.edu

Abstract

Image labelling has always been a consuming task in both time and human resources. Generally, the most common data labelling tools and formats include LabelMe json files[10], COCO dataset[7], lvis[5], etc,. While making image annotations, we need to repeatedly view images of similar themes and type in classes and bounding-box coordinates. Is there a way to make image labelling more efficient and interesting? In our final projects, we proposed an innovative image labelling system that will provide users with image annotations shortly after they send their customer's images via the front-end of our system. We divided our system into a Graphic User Interface for users to interact with the deep-learning engine and a back-end to generate image annotations. We evaluate with labelling accuracy and different test cases to prove the robustness and performance of our system.

1. Introduction

Deep learning is always related with data. For the most common supervised learning methods in deep learning, we usually need a large number of training data and labels to guarantee that deep learning model can learn adequate features and information to make reliable predictions. However, data labelling is never easy work. It is extremely consuming in time and human resources when faced up with millions of raw data to be labelled.

To make image labelling more efficient and interesting, we proposed an innovative Automatic Image Labelling System. Our Automatic Image Labelling System is composed of a graphic user interface and a software back-end. Users can upload their customer's images and get annotations via the GUI, and if they are not satisfied with the output annotations, they can choose to modify, drop or add annotations depending on their intentions. The back-end of whole system calls a image detection deep learning model to calculate classes and locations of targets in each image. Once the work of each image id finished, our system will absorb users' images into its training dataset and periodically start new training process to improve its precision and accuracy.

In this report, we will generally introduce our progress of the final project. In Section 2, we will take a quick look at related work of our project. In section 3, we will introduce the model, view, controllers and other implementation details of our project. In section 4, we introduce how we collect and manage out data. In section 5, we will show the current progress we have achieved. In section 6, we will introduce the experiments we conduct and the results to give a brief impression of our system's performance.

2. Related Work

Deep learning has been widely used in target bounding-box locating [2][11][12][13]. In OverFeat[11], the author employs a Fully-Connected(FC) layer to predict locations of bounding-box for single object and a conv layer for multi-objects detection. Multi-box methods[2][12] uses region proposals which can be used for Region - Convolutional Neural Network(R-CNN) for bounding-box prediction. R-CNN is based on different regions of an image and is able to detect multiple targets.

Ross Girshick *et al.*[3] introduced a high-speed Region-based Convolutional Network model (Fast R-CNN). Fast R-CNN uses convolutional layers and max pooling layers to formulate a fixed-size feature map and uses Region-of-Interest(RoI) branches to perform class-classification and bounding-box regression. Shaoqing Ren*et al.*[9] introduces a innovative two-stage Region Proposal Network (RPN) that receives feature maps from detection network and then generate high quality region proposals. Such method uses a sliding window to generate boxes that potentially contain objects and speed up the forward propagation process. Also, RPN, as a two-stage detector, is able to provide higher detection precision.

3. Automatic Image Labelling System

3.1. Front-End

We first build an electron-react based application framework, and then extend the functions to three parts: mask

detection, revise of the detection results, display of the revise history.

3.1.1 Technical Stack

Our front-end technical stack is formed by Electron, React, React-hooks, Semantic-UI, and axios. To build a cross-platform desktop application, we create the GUI of our node application with electron. Electron[8],is a framework for integrating HTML, CSS and Javascript which can be used to build a cross-platform application. To minimize the bugs that occur when building UIs and abstract some rendering works, we adopt react. React[14] is an efficient and latest front-end JavaScript library which are widely used for UI components development maintained by Meta. To reduce the barriers the class component brings and reuse more stateful logics, we choose react hooks as our lifecycle. Hooks[1], which is a new feature implemented in React 16.8, which is a really brand-new technology. With hooks, we can reduce redundant codes by adopting functions instead of writing class. And to make the UI more fancy and improve the development efficiency, we use Semantic-UI as our main UI framework. Finally, we implement axios library in Javascript to send HTTP GET and POST request to back-end. With these latest technologies, we build a cross-platform desktop application used for auto annotation.

3.1.2 Functions

Fig. 1 shows the overall preview of our desktop application. The workflow of the application can be divided into three main parts: mask detection, revise of the detection results, display of the revise history.

For the first function, a user can upload an target image by input the source url or path of the image, and click on the upload button, and we can present the detection result in 0.5 seconds. The detection results include mask detection result, bounding box and confidence. This part provides the detection results of our model, which can be revised by users to refine our model.

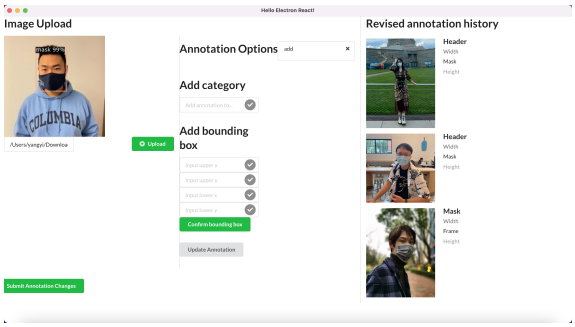


Figure 1. Application UI preview. The functions include mask detection, revise of the detection results, display of the revise history.

In the middle part of the UI, this is the second function where our users can revise the detection results of each image. Once the user finds out the detection results are not precise, we allow the user to revise the annotation to make our model more precise. And we realize this function by supporting three kinds of options: add annotations, delete annotations and update annotations. Fig. 2 displays the option of adding an annotation, the user can add a new annotation for current image by input the mask detection result and the coordinate of the bounding box. And users can delete an existing annotation if they find that there is a mistakenly recognized object. This scenario is shown in Fig. 3, where we provide a dropdown component for users to choose from. Also, we allow our users to revise an existing annotation, this is shown in Fig. 4. The user can change an existing annotation by rewriting the category (mask or unmask) or change the bounding box. These three options fit the needs of revising an annotation, which is very user-friendly. Also, by updating the annotation in the back-end dataset as training data, our model will be refreshed and therefore becomes more precise.

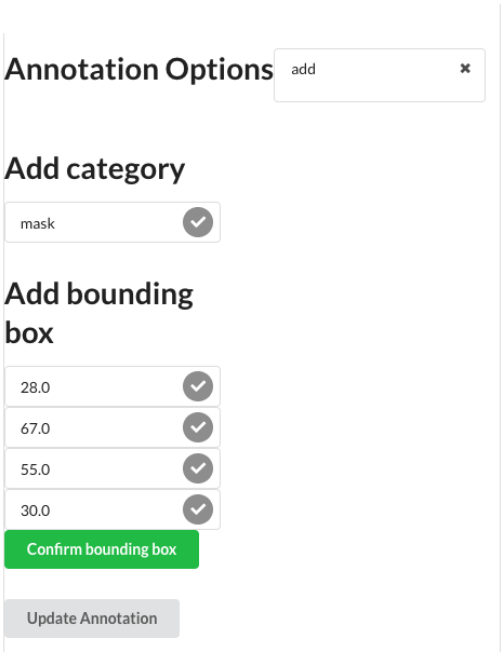


Figure 2. Add an annotation. This is an example of adding a new annotation by input mask detection result and coordinates of the bounding box.

We present the revise history of current workflow in the third part. And we can see from Fig. 5, the history change information includes the image, category and bounding box. This is useful especially when a user wants to see the history logs.

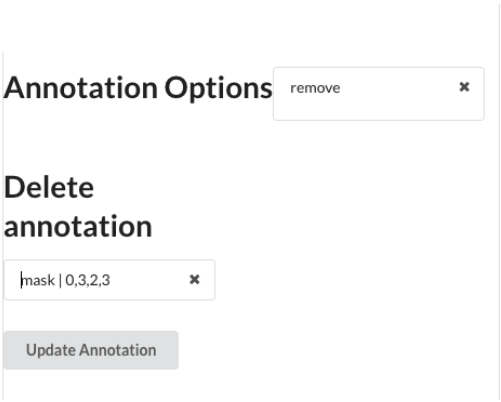


Figure 3. Delete an annotation. This is an example of deleting an existing annotation.

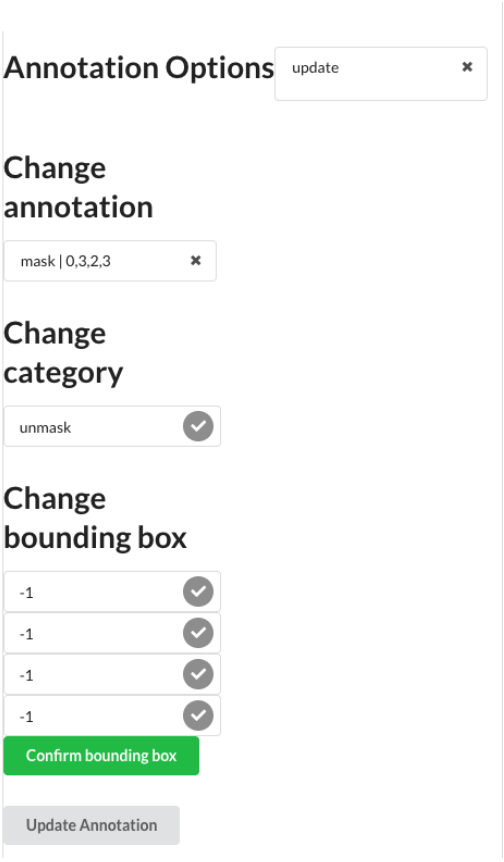


Figure 4. Update an annotation. This is an example of updating an existing annotation by change mask detection result and coordinates of the bounding box.

3.2. Back-End

We build our back-end based on Flask[4] to implement five main functions: Receiving labelling requests from front-end, Calling deep-learning model to generate annotations, Sending annotations to front-end, Receiving confirm, modification, add or drop requests from front-end and Writing annotations into label files.

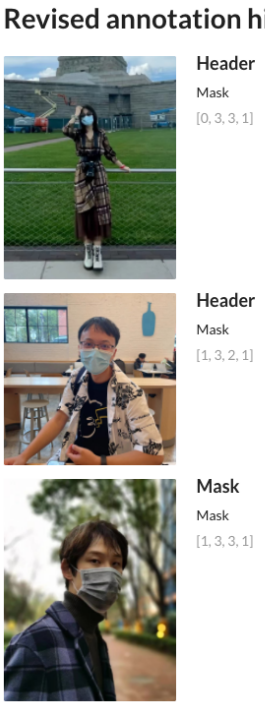


Figure 5. Revise history. This shows the history options of the current user.

Generally, our back-end is consist of a deep-learning engine and a communication module to interact with front-end.

3.2.1 Deep-Learning Engine

Our deep-learning engine is based on Faster R-CNN[9] to implement object classification and locating. Faster R-CNN uses region proposals to accurately and fast detect targets and can be divided into three cascaded networks. Here we will give a brief introduction of Faster R-CNN's composition and how it works.

1. Backbone: Backbone, just like its name, is the most important part in Faster R-CNN. Backbone can be viewed as a feature extractor whose input is RGB images and output is a feature map with fixed size. Faster R-CNN employs a Deep Residual Network(ResNet)[6] to extract a feature map for further classification and detection.

2. Region Proposal Extractor: Faster R-CNN raises a very innovative Region Proposal Network to generate region proposal as boxes that contain targets. Region Proposal Network, as known as RPN, is a typical two-stage network. The first stage of RPN uses a sliding window and generate several region proposals which

is a box of area in the input image. The second stage of RPN is trained to determine whether a region proposal is positive sample(foreground) or negative sample(background). In RPN, Non-Maximum Suppression will also be used to merge region proposals that have a high Intersection over Union(IoU).

3. RoI Head: RoI is short of Region of Interest which refers to areas we will be focusing on. Output region proposals from RPN will be sent to RoI Heads. In Faster R-CNN, there are two branches in RoI Heads that are responsible for object classification and bounding-box regression.

In the deep-learning engine of our back-end, we initially train a Faster R-CNN model using training data we have collected from the Internet. Every time the back-end receives a labelling request from front-end, it will call the deep-learning model's forward propagation to generate annotations.

3.2.2 Back-end Framework

Communication module of our back-end is based on Flask[4] to interact with the MySQL database. Flask is a lightweight Python development web framework. The back-end of this project uses the MVC framework, namely Model, View and Controller to process all requests from customers, and respond to such a framework to make the back-end more modular.

We use Flask's Blueprint feature to implement corresponding APIs in different routes, including login, registration, upload image test, modify annotations, and automatic training, upload and save operation logs and other functions, and the corresponding Controller calls business logic to control .

Dao files are responsible for corresponding database operations. When registering an account, the system automatically stores user information in the database, and assigns a unique model storage address and an address for storing annotations files to the user, so as to achieve personalized data for different user training. Passwords and other sensitive information are encrypted when stored.

Flask's session is used to ensure that any operation in the training project is limited to the data range of the account, and all information will be called through the session when the user operates. When the back-end receives the front-end image data, it will automatically call the API related to the model, process the request from the front-end, and return the data obtained after image processing.

The log record module helps users to query their past operations. The main operation of each account will be automatically recorded and saved by the system after logging in. With the help of Hook Function, the information

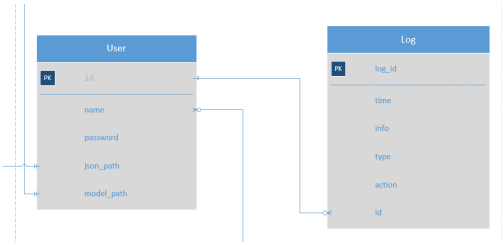


Figure 6. Relationship between User and Log.

recorded by the system includes time, operation success or failure, interaction information during operation, and so on. This helps to record the historical actions of each user. Each user can query only his own operation history through a specific API. Database interaction mainly uses the ORM nature

log_id	time	info	type	action	id
13	Sun Nov 28 15:16:20 2021	failed	http://loc…	{'username': …	2
14	Sun Nov 28 15:16:29 2021	success	http://loc…	{'username': …	2
15	Sun Nov 28 15:19:07 2021	failed	http://loc…	{'url': '/Use…	2
16	Sun Nov 28 15:19:46 2021	failed	http://loc…	{'url': '/Use…	2
17	Sun Nov 28 15:22:30 2021	failed	http://loc…	{'url': '/Use…	2
18	Sun Nov 28 15:23:08 2021	failed	http://loc…	{'url': '/Use…	2
19	Sun Nov 28 15:23:44 2021	failed	http://loc…	{'url': '/Use…	2
20	Sun Nov 28 15:26:57 2021	failed	http://loc…	{'url': '/Use…	2
21	Sun Nov 28 15:29:29 2021	success	http://loc…	{'url': '/Use…	2
22	Sun Nov 28 16:10:10 2021	success	http://loc…	None	2
23	Sun Nov 28 16:11:23 2021	success	http://loc…	None	2
24	Sun Nov 28 16:11:45 2021	success	http://loc…	{'username': …	2
25	Sun Nov 28 16:11:51 2021	success	http://loc…	{'username': …	3
26	Sun Nov 28 16:11:57 2021	success	http://loc…	None	3

Figure 7. Part of Log table.

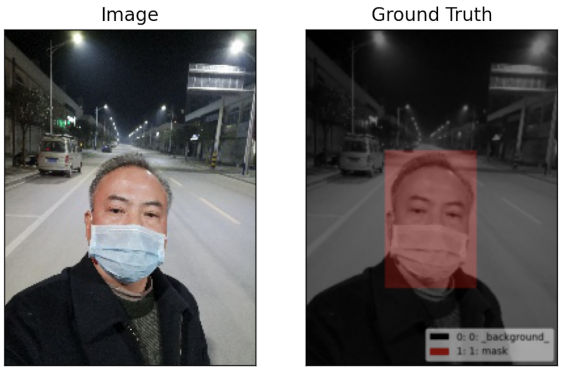
of Flask. The ORM nature of Flask realizes the interaction between the class and the database. With the nature of ORM, Flask can enable python language to interact with the database, and specific classes can correspond to specific database tables, which makes all operations between the backend and the database class-specific. The database uses the MySQL database of RDS in Amazon AWS, which is safe and reliable.

4. Data

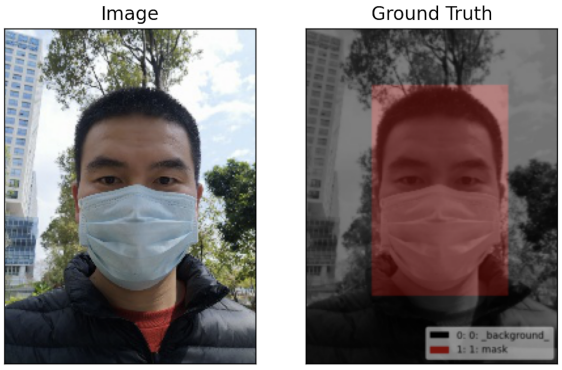
In our experiments, we use mask detection as an example. We collect our raw data from https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset and randomly select approximately 1000 images from it. We make annotations of these images with LabelMe[10]. The whole dataset is divided into a training dataset and a testing dataset and the ration is around 7:3. Some examples of original images and ground truths are shown as in 8.

5. Current Progress

We have created the framework of our desktop application and finished the basic functions for the interaction with users. Also, we trained the deep learning model for mask detection and achieved a considerable precision.



(a) Image



(b) Image

(c) Image

(d) Image

Figure 8. Image and Ground Truth

5.1. Desktop Application

In front-end, we designed the UI with our planned technical stack and realized several functions. In back-end, we programmed several APIs for the interaction with front-end and send data either extracted from database or returned after calling the propagating function with our deep learning model.

5.1.1 Front-end overview

Fig. 9 shows the initial overview of our application. Fig. 2 to 4 shows the workflow of how a user can make changes to the annotations of an image. Fig. 10 shows the confirmation message when a user want to save the changes to our data structure and upload it to the dataset for improving our deep-learning model.

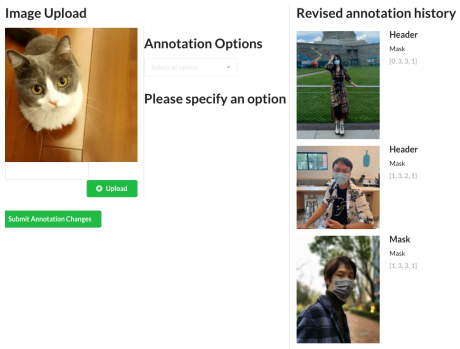


Figure 9. The initial page of the desktop.

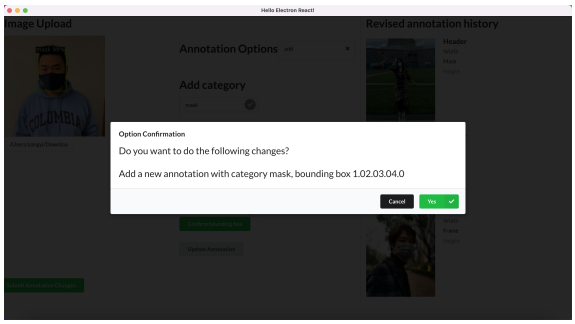
Fig. 11 displays the changes made by the user after the above options stored in our back-end json file, where we can find that a new annotation of mask has been added.

5.1.2 Back-end overview

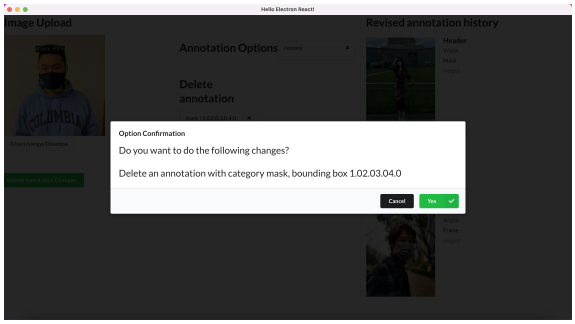
Our business logic is to allow users to implement online predictions and model training. In order to help users have a better experience, we added the function of personalized training on this basis. We use a Flask-based architecture to better implement the business logic.

At the beginning, the back-end architecture was very simple, using only two routes that could easily handle front-end requests, which made the project's scalability too low and the coupling too high. In order to improve the project, we decided to use the MVC architecture to make the project structure clearer by separating the layers, and at the same time more scalable, which of course also benefits from the flexibility of the Flask framework. Flask's ORM attributes also help us better interact with the database.

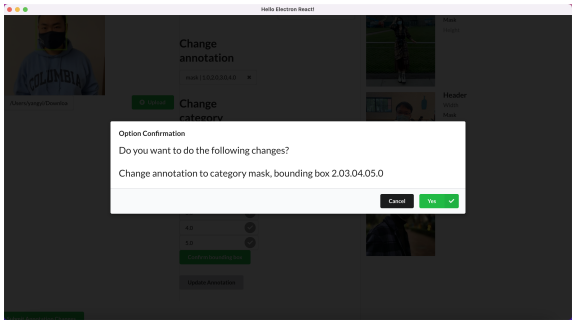
In order to better implement the business logic, we encapsulated the core logic in the Service module and called it through the Controller module, which also included our deep learning model. The Entity module encapsulates the various models that we interact with the database, and the Dao module handles the various parts of the interaction with the database. This way, the deployment of the project is more flexible, and it is convenient for us to insert more functions.



(a) Add an annotation



(b) Delete an annotation



(c) Update an annotation

Figure 10. Change annotations of the image

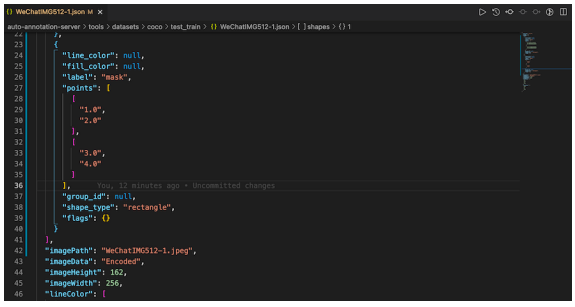


Figure 11. Annotation changes stored in back-end

5.2. Deep learning model

Our deep learning model is based on Faster R-CNN and trained using mask images we collected from Internet. We use approximately 400 images as a training dataset to obtain a initial model as shown in Fig. 12.

category	#instances
mask	543

Figure 12. Overview of training dataset

During our training process, we encountered several problems. At first, we employed a ResNet-18 to extract features, but found that ResNet-18 is too simple to make a good fit. Also, the random initialization of our network sometimes causes model divergence. We added a warm-up training at the beginning of the training process and set learning rate to be 0.0002 throughout all training. Some other parameters we use in training are shown as in Fig. 13.

```
MODEL:
  WEIGHTS: ""
  MASK_ON: False
  RESNETS:
    DEPTH: 18
    RES2_OUT_CHANNELS: 64
  ROI_HEADS:
    NUM_CLASSES: 2
    SCORE_THRESH_TEST: 0.85
    NMS_THRESH_TEST: 0.25
SOLVER:
  IMS_PER_BATCH: 8
  BASE_LR: 0.0002
  STEPS: (40000, 42000)
  MAX_ITER: 45000
INPUT:
  MAX_SIZE_TRAIN: 256
  MIN_SIZE_TRAIN: (0, 256)
  MIN_SIZE_TRAIN_SAMPLING: "range"
  MAX_SIZE_TEST: 1280
  MIN_SIZE_TEST: 0
  RANDOM_FLIP: "none"
```

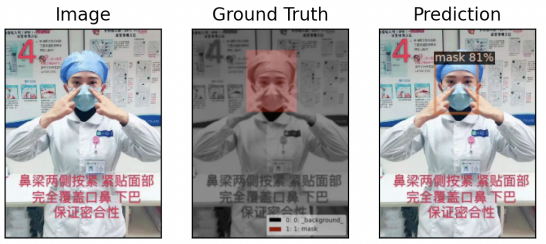
Figure 13. Overview of training parameters

6. Experiments

We conducted our deep-learning experiments on Google Cloud Platform. We use CUDA and a NVIDIA Tesla T4 GPU to speed up our training process. Fig. 14 shows how our model performs on our test dataset. We use a Faster R-CNN model with ResNet-34 as backbone and use mini-batch gradient descend as optimization. To avoid the prob-

lem of divergence, a warm-up training and a learning rate of 0.0002 are adopted. There are a total of 20000 epochs during training.

(a) Image



(b) Image

(c) Image

(d) Image

Figure 14. Test Image, Ground Truth and Prediction

AP	AP50	AP75	APs	APm	APl
32.494	64.792	26.677	20.755	36.286	42.568

Table 1. Average Precision

Table. 1 shows the average precision of our model. Average precision is the most direct criteria to evaluate how well our model performs. Also, it takes approximately 0.5 seconds to process each image on a MacBook Pro.

References

[1] Changelog. Introducing hooks. https://reactjs.org/docs/hooks-intro.html, 2021. 2

[2] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2147–2154, 2014. 1

[3] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[4] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018. 3, 4

[5] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5356–5364, 2019. 1

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1

[8] Rohan Reddy. Cross-platform applications with electron. https://www.section.io/engineering-education/cross-platform-applications-electron/, 2020. 2

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015. 1, 3

[10] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008. 1, 4

[11] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 1

[12] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014. 1

[13] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. 2013. 1

[14] Whereismango. React (javascript library). https://en.wikipedia.org/wiki/React_(JavaScript_library), 2021. [Online; accessed 22-July-2021]. 2