December 2019 CHEN LIANG

CSC420 project report

1. Introduction

When I was travelling in Paris years ago, I was always marvelled by some magnificent yet elegant architectures along the streets. At that time, I was so annoyed because I could bearly tell which building they were due to my lack of background knowledge of Paris. When I learned deep learning from CSC420 this semester, I realized that I could solve the dilemma I encountered when I was a kid by merely applying a convolutional neural network to judge what architecture it is. To achieve this, my teammate and I read through the Facebook research on Detectron [1] and want to recreate our version of object detection based on Detectron. We expect our object detector could categorize each architecture to its corresponding class or label accurately and draw a ground truth box that tightly bounds the outer frame of the architecture. We also expect the detector to reveal how much "confident" such classification is correct by showing a prediction confidence score alongside each ground truth box. One of our main challenges would be there is no existing dataset online, so we have to create a dataset containing 14 different kinds of architectures in which every single image needs to be downloaded from the Internet manually. Our second main challenge is to establish a particular environment to make our object detector work. More specifically, we need to install all dependencies for Detectron, including Caffe2 [2], then set up the Detectron module. Setting up Inference might appear to be simple, but ensuring the environment is indeed the one we want would be super time consuming, and the whole process proves to be very gruelling. Our third main challenge could be drawing bounding boxes for every single image, which would take a lot of time and effort. Since Detectron could implement detection modules like Faster RCNN [3], MaskRCNN [4], it is possible to detect multiple objects in a single image with a relatively faster speed while keeping comparatively high accuracy. In this way, the most significant limitation would be the size of the dataset we created, where the maximum number of images for each class never exceeds 300 images. The size of our data set is drastically smaller than the size datasets commonly used, such as the COCO dataset [5] or VOC2007 dataset [6]. Due to the limitation of dataset size, a shoot of architecture from a different angle might not be recognized. Another limitation that needs to be addressed is if we apply filters on the input videos, our model might not recognize these landmarks as precisely as wanted.

2. Methods

Data Collecting: My teammate and I built a Paris dataset from scratch. This dataset contains14 classes and for each class, the number of images ranges from 100 to 300. Then we use labelImg [7] to label ground truth boxes for all 2503 images manually and save information for all ground-truth boxes as paris_data.txt, including their image ids, the starting width, height and starting width, starting height for each box. Then use generate_json_for_paris_dataset.py to convert the .txt file to a .json file, and save it ***paris_building_train.json*** under annotation folder. Our .json file functions as a dictionary, with images labelled by a unique index, and by checking its index, we could learn about their bounding box information.

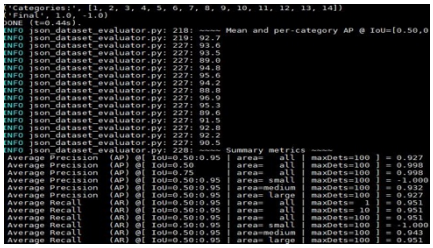
Architecture: We chose to use ResNet50 [8] as the backbone rather than VGG 16 [9] due to the reason that replacing VGG16 with ResNet could be spotted improvements as much as 28% [10]. Moreover, as ResNet50 could tackle the vanishing gradient problem, ResNet50 allows us to train relatively deep neural networks without its performance gets saturated or even starts degrading rapidly [11].

Model: We chose to use Faster RCNN based on FPN [12] to train our dataset. FPN is very competitive with state-of-the-art detectors because it generates multiple feature map layers with better quality information than the regular feature pyramid for object detection within a comparatively inference time. [13] The below graph illustrates Faster RCNN on FPN is an expert in extracting masks for images, and it performs better than models like G-RMI, Attractio-Net, etc.

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Source

Training Model: We created a YAML file named ***Paris_Detector_ResNet50_FPN_s1x-e2e.yaml*** under configs folder. In this YAML file, we configured hyperparameters for training, setting the maximum iterations as 90000, the number of ROIs per image as 512, images per GPU as 2, the number of top-scoring RPN proposals as



2000 [14]. Then we started the training process. The training was done on an NVIDIA GTX 1070 GPU, achieved >92% accuracy, and the weight was saved as ***model_final.pkl*** under model folder. Left is our training result, it might appear to be a little bit overfitted due to the number of iterations we set (90000 iterations).

Testing On Videos: After getting trained weights, we first test on the test dataset, and the testing accuracy for all classes is all over 88% (as the image on bottom left shows). After confirming the testing result on the test dataset is acceptable, we call ***video_infer_simple.py*** (based on infer.py and infer_simple.py written by Facebook Research team [1]) to obtain detection results for the input video. The result comes as 25 frames of detected images per second, and are all saved under demo_test_out folder. Then we wrote ***generate_video.py*** to convert frame images back to the resulting video. The testing result on videos functions way better than expected, as it can even correctly detect buildings if only parts of the buildings are shown.

Map Feature and Filters: We utilized the function of object detection by adding a map indicating the location of the architecture detected. In this way, if a tourist even does not know the name of the detected architecture, he could follow the map shown at the bottom right corner of the video. To achieve this, we changed the vis.py file under the detectron/utils folder. By using ***add_map_to_image*** function, we could add a map of Paris at the bottom right corner for all frames. Once labels or are detected in a frame, we use ***add_icon_to_map*** function to add the corresponding icons for detected buildings to the map according to their locations on the map.
Also, in vis.py file, we could add filter functions to achieve the effects of Instagram like filters on resulting images. Here we devised filters such as Gotham, Inkwell, Canny Edge, Hefe, Vintage etc. Also, we could move the filter functions out of the vis.py file and use it to filter the input video. In this way, we could check whether our detector could detect buildings as well-off as detecting videos without any filters.

Responsibilities:
Yifan Zhang: Yifan Zhang was mainly in charge of setting up the environment for training and testing processes, configuring the YAML file for hyperparameters, and devise the commands and folder structures to make training and testing processes function well. He also downloaded images for the Paris data set, drew bounding boxes with Chen and added the map feature to make map visualization possible. He did most of the work for writing and editing the PowerPoint for our presentation.
Chen Liang: Chen Liang was responsible for downloading images to Paris dataset, use IabelImg [7] to draw bounding boxes manually, and create ***paris_building_train.json*** file to store all bounding boxes information. He also did some researches online and chose the model for training and configured the YAML file with Yifan Zhang. He converted the resulting training images back to the video, and he also added the filters feature to the project. He also edited the video for the presentation.

Facebook Research Team: Because this project is based on the code provided by Facebook Research Team, they provided the code and ideals for the Detectron system. [1]

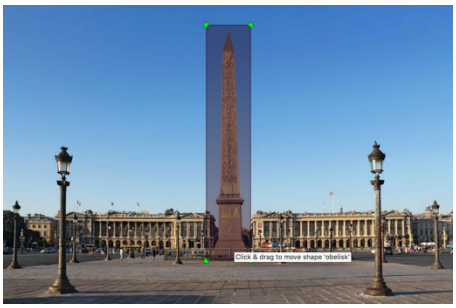
3 Results and Discussions

Overall, our Paris building detector yields fairly good results. In the 2-minute we uploaded on Chen's youtube channel, nearly all buildings could be correctly classified, and the ground truth boxes could be accurately detected. For specific architectures with relatively larger dataset sizes, such as Eiffel Tower, Notre Dame, and Sacré-Cœur, the detector is almost 100 percent about the detection, and almost no frame missed detecting architectures like these. It could even effectively detect a building when only part of the building is displayed, as the below graph illustrates.

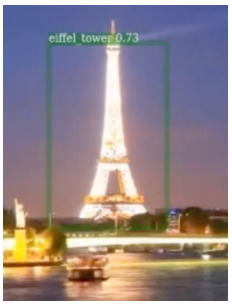
Moreover, the detector can also detect multiple objects within one frame, and functioned as we wanted, like the graph left shows.

Unfortunately, our Paris building detector still has its limitations:
(1) Some architectures like Palais du Luxembourg and the obelisk on la Place de la Concord are never successfully detected (over 70% confident) in the video. For Palais du Luxembourg, we conclude the reason leading to such a result mainly due to small dataset size. There are only 100 images for Palais du Luxembourg both for training and testing since it is hard to find

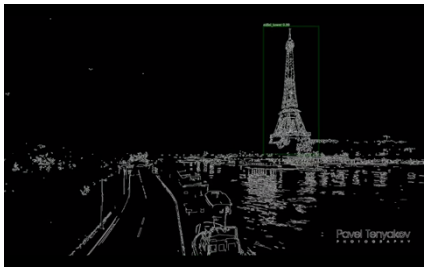
images for this place on Google image search, resulting from the fact that Palais du Luxembourg is relatively unpopular compared to touristy places like Effiel Tower. Most of the images in this class we found online only contains the frontal view of this building, making the number of photos containing the side view even less. Furthermore, in our 2-minute video, all frames containing Palais du Luxembourg, the main Palais is blocked by a statue, only making it even harder to detect. To solve this problem, we need to collect a larger dataset for Palais du Luxembourg, where it contains photos of the main Palais from various shooting angles.



For the obelisk on la Place de la Concord, we conclude the reason for the detector failed to detect it as we made a mistake when plotting bounding boxes for the training dataset. Left is an example of how we typically drew bounding boxes for obelisks. After converting the image bounded in the bounding box to a 256 * 256 image (as left image shows), the shape of the image got significantly disfigured. And make things worse, in our system the maximum length-width ratio for bounding boxes is 1:4, meaning that even if we could successfully bound the obelisk with a 1:4 bounding box; due to the unique shape of obelisks, the obelisk could not occupy the most of the spaces within such bounding box, varying a lot from what the machine has been trained. To tackle this issue, we need to devise a larger length-width ratio and leave more space when creating bounding boxes for this class.



(2) Illumination has a significant impact on object detection.
The example below shows vividly demonstrates that illumination on structures might have a notable influence on detecting the object. When the lights lit on, the accuracy decreases. We could preprocess the input video to avoid buildings got overexposed. Besides, we can add more images with lights on into our dataset to address this issue.



(3) Applying self-written filters on input videos would make object detection a little bit harder, but the impact is much significant than expected. We clipped a short video (https://youtu.be/pQIt2QPstYc) to show that if we apply filters such as HEFE filter or Canny Edge filter, the preprocessed video will look way more artistic, but the test accuracy was compromised on a very slight scale. We wanted to spice things up by adding some more elegant filters such as converting images to Van Gogh's Starry Night style [15] at the first place; later we realized converting to Starry Night might make the image lose its features and leading detection way harder. If we dramatically increase the training dataset and adding more augmentations during the training process, artistically processed images might be accurately detected.

4 Challenges

We faced several challenges when working on this project.
The first great challenge was that we have to learn the Facebook Research group's Detectron project [1] from scratch and set up an environment exactly the same as the INSTALL.md file required. During the Inference setup, we encountered many problems like the current version of Caffee2 [2] did not contain OpenCV, etc. Many of these problems arose when we started the training process, so we had to adjust the environment until it was correct constantly.

As we chose to use FasterRCNN [3] on FPN [12] as our model, it also met the common issue all generalized RCNN methods share: Our model sometimes fails to detect small objects. To solve this problem, we could change our SSD [16], DSSD [17], or Feature-Fused SSD [18], as generalized SSD models could effectively detect small objects in a fast and accurate manner.

Another challenge was many frames of our input video contains buildings viewed from different angles. Some of the frames are even aerial shots. To cope with this issue, we added images of multiple facets of the same building, ensuring they could be viewed from as many viewing points as possible.

As mentioned above, to solve issues like failing to detect architectures with relatively long shapes, we could set more choices for bounding boxes when testing, and when manually draw bounding boxes for training purposes, we'd better ensure bounding boxes do not exceed the maximum length-width ratio we set.

We were inspired by a project which calculated the distance from the camera to object/marker [19][20] and tried to incorporate knowledge of camera model into our project to calculate the distance of buildings to the camera. However, to achieve this, we need to know the relative size of the building under a pre-fixed camera distance. In the future, we could achieve this by taking reference photos of target buildings; then, we could calculate the distance when given an image containing these buildings.

5 Conclusion:

We successfully finished the object detection part of our project. We also implemented useful features like showing the location of the detected object on a Paris map and wrote a couple of artistic filters to input/output videos. We failed to calculate the distance from buildings to the camera due to the lack of reference photo and reference distance. We will replace Faster RCNN with other models like SSD and check which model could achieve the best training result both in terms of training speed and testing accuracy. I will be travelling in Paris again in May 2020; I'll take more photos for each building from different viewpoints and under different conditions (night, rainy day, etc.) to make our dataset more complex. I will also take reference photos and finished the distance calculation part of this project.

6. Citations

[1] Facebook Research/Detectron [online]. Available: https://github.com/facebookresearch/Detectron. [Accessed November 2019]

[2] FacebookArchive/Caffe2 [online]. Available: https://github.com/facebookarchive/caffe2. [Accessed November 2019]

[3] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. [online]. Available: https://arxiv.org/abs/1506.01497v3. [Accessed November 2019]

[4] Mask R-CNN. Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. IEEE International Conference on Computer Vision (ICCV), 2017. Available: https://arxiv.org/abs/1703.06870. [Accessed November 2019]

[5] Coco Dateset. [online]. Available: http://cocodataset.org/#home, info@cocodataset.org. [Accessed October 2019]

[6] PASCAL visual object clases,[online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/. [Accessed November 2019]

[7] tzutalin/labelImg, [online]. Available: https://github.com/tzutalin/labelImg. [Accessed November 2019]

[8] Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Available: https://arxiv.org/pdf/1512.03385.pdf. [Accessed November 2019]

[9] Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman. Available: https://arxiv.org/abs/1409.1556. [Accessed November 2019]

[10] Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image Classification: From Microsoft to Facebook, Parakah Jay. Availale: https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624. [Accessed November 2019]

[11] Understanding and Coding a ResNet in Keras, Priya Dwivedi. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. [Accessed November 2019]

[12] Feature Pyramid Networks for Object Detection. Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. [online] Available: https://arxiv.org/abs/1612.03144. [Accessed November 2019]

[13] Understanding Feature Pyramid Networks for object detection (FPN), Jonathan Hui. [online]. Available: https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c [Accessed November 2019]

[14] Facebook Research/Detectron [online]. Available: https://github.com/facebookresearch/Detectron/tree/master/configs/getting_started. [Accessed November 2019]

[15] jcjohnson/neural-style[online]. Available: https://github.com/jcjohnson/neural-style [Accessed November 2019]

[16] SSD: Single Shot MultiBox Detector, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. [online]. Avalilable: https://arxiv.org/abs/1512.02325[Accessed November 2019]

[17] DSSD : Deconvolutional Single Shot Detector, Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, Alexander C. Berg. [online]. Available: https://arxiv.org/abs/1701.06659. [Accessed November 2019]

[18] Feature-Fused SSD: Fast Detection for Small Objects, Guimei Cao, Xuemei Xie, Wenzhe Yang, Quan Liao,Guangming Shi, Jinjian Wu. [online]. Available: https://arxiv.org/pdf/1709.05054.pdf. [Accessed November 2019]

[19] Find distance from camera to object/marker using Python and OpenCV, Adrian Rosebrock.[online]. Available: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/.[Accessed November 2019]

[20] pablovela5620/Hand-Detection-and-Distance-Estimation. [online]. Available: https://github.com/pablovela5620/Hand-Detection-and-Distance-Estimation. [Accessed November 2019]

7. Project Presentation Links

link to code: https://github.com/liangc40/csc420-project

link to presentation (Youtube): https://youtu.be/wJ3nLqt2oeQ

link to full result video: https://youtu.be/1SOQVGi7OgE

link to test result on filtered video: https://youtu.be/pQIt2QPstYc