Classification of Buildings using Google Street-View

Francisco Lozano, Abhi Kamboj, Cary Chai
EPFL, Lausanne, Switzerland

***Abstract*—There are currently a very diverse range of building materials and construction styles used in cities throughout the world. Being able to track these materials and methods in buildings is important as it determines the procedures for rehabilitation and repairs. In order to determine the make of a building, recent data about the buildings is needed which is not always readily available. Identifying each building, if not automatized, would require an enormous amount of manpower. We hope to automatize this task, providing a framework capable of fetching images of buildings from Google Street-View and classifying them.**

I. Introduction

For the purpose of collecting data of the buildings of Switzerland, together with the *Laboratoire dínformatique et mécanique appliquées à la construction*, we were tasked with capturing images of buildings in Zurich and saving their locations and characteristics. Data was collected using the Google Street-View API and several methods were attempted to manipulate and transform the data sufficiently for the neural network. Then using classification models, certain characteristics were extracted from the images. The building classification was broken down into 3 steps, and three corresponding neural net models. The first task was detecting the buildings in an image. The second step was distinguishing openings (windows, doors, etc.) on a building to calculate the opening to façade ratio, a highly important characteristic in determining the building type. The third step was classifying the buildings based on the materials they were made of. We have chosen for these tasks:

- Faster R-CNN ResNet-50 FPN [1] to extract the buildings from Google Street-View.
- DeepLabV3 ResNet101 [2] to differentiate openings on the facade.
- ResNet-50 [3] to classify the materials on the facade.

Since this project consisted of characterizing buildings with specific characteristics there was no pretrained model or prexisting labeled dataset. Therefore, tools for continuously sampling images from Google Street-view and an image labeling application was created. Labeling images is an arduous task that had to be done by hand, but many labeled samples are necessary to train deep learning models. For this reason, we have studied the effect on the efficacy of the models as the dataset they are trained on increased.

II. Data Collection

A large portion of this project was data collection and refinement for the machine learning models. The first image generator simply pulled random images of buildings in Zurich from the Google Street-view API. This method was quick however it gave images that would make it more difficult for the neural net. Buildings appeared from different angles and distortions. In addition, since one goal was to capture the opening to facade ratio, it was undesirable to capture only part of the building since different parts of the building might have disproportional opening to facade ratios, and just the part that is captured is not representative of the entire building.

To combat this problem, we focused on how to get the image of the complete building. The Google Street-view API allows one to query an image by location, heading, field of view, and a few other parameters. The API was queried at the same location with various headings such that the camera was panned around a fixed point, creating a panorama. The image was then divided to see each side of the road (180 degrees) individually. As shown in figure 1, this resulting image had some distortion and although it covered the entire building, it might have misled the neural network. This also suffers from the original problem; it captures many different angles of the buildings which could mess up the opening to facade ratio. When looking at a building from an angle, the windows in the front, closer to the observer, are bigger and would count for a larger percentage of the building. This panorama also took about 50 images to complete, which is time consuming and computationally costly. We succeeded on decreasing the amount of images required to only 3 consecutive images. The result, shown in figure , has much more distortion then before.

Figure 1. The panoramic output has a much larger field of view and allows much more information to be captured, however in 2d the façade of the building is distorted and might affect the opening to façade ratio. Additionally, this higher quality required 50 separate images from the Google Street-view API.

Figure 2. This is a quick stitch that takes significantly less images (only 3 images) and the image is facing the buildings directly, however, it has some distortion.

The second and more effective solution to generating sufficient images was to use a preexisting database of coordinates of buildings in Zurich, provided by the *Laboratoire dínformatique et mécanique appliquées à la construction*. This database provided the coordinates of the center of each building. Using a snap-to-road function which inputs any coordinate pair and returns the coordinates to the nearest road, Google Street view API was used to get the image of the building at that coordinate. Now the image could be guaranteed to be facing the building straight on, by calculating the desired heading of the camera to be perpendicular to the road and the building. This direction can be calculated by taking the difference from building coordinates to the corresponding snapped road coordinates and using basic trigonometry to calculate the angle relative to North that the building was at, since North corresponded to zero degrees heading in the Google Street-view API queries [3].

Figure 3. In the regular view, less of each building is captured however the image is taken perpendicular to the facade, which would in theory provide a more accurate facade to opening ratio.

Although most of the images are captured perpendicular to the building there are still some issues. Images from the Google API taken from the sidewalk do not have a fixed heading of 0 corresponding to North as do those taken from the road, therefore those images are still not facing directly towards the building after the heading calculation. Also, the database of buildings contained more buildings than the Google Roads API, so there were many times where different buildings would snap to the same nearest road or calling the Google Street view API would return the same image. The data generator was later adjusted so that repeated pictures were ignored.

III. Object Detection

For the task of object detection we considered implementing Faster R-CNN. Faster R-CNN shows no worse accuracy, and runs quite faster than other state-of-the-art models used for object detection [4].

To obtain the necessary samples to train the model, we implemented a multi-platform application capable of browsing through a directory and allowing you to locate where the buildings are located using a simple and intuitive interface.

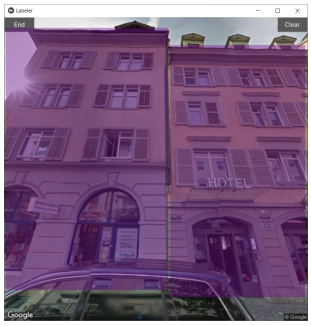
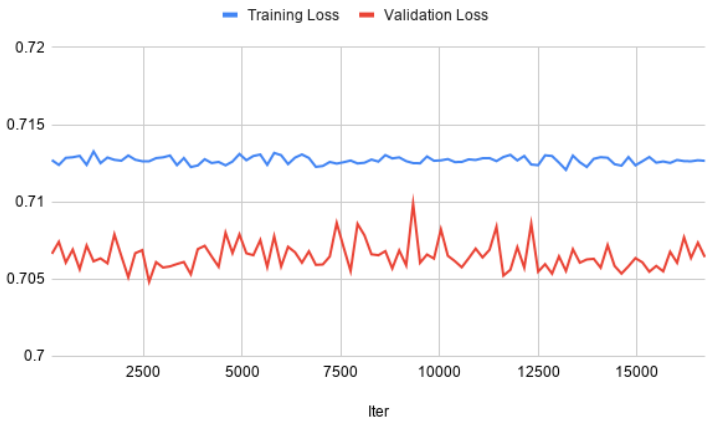


Figure 4. This labeling application was created specifically for this project. This image shows the object detection labelers which allows a user to click and drag open a bounding box around the buildings.

This application stores the coordinates and the label of each bounding box created and stores them with a reference to the image in a preexisting csv file that will be used to train our model.

Due to the time intensive nature of our data collection for this project and our research into state of the art object detection models, we opted for the Faster R-CNN ResNet-50 FPN by Torchvision. We trained it on iteratively larger image sets (500, 1000, and 1500) that we created using our custom tool from Figure 4. However, we were unable to get the accuracy to increase using this model and loss remained constant. We believe that the model implemented by Torchvision requires fine-tuning to be able to reach a sufficiently precise model.

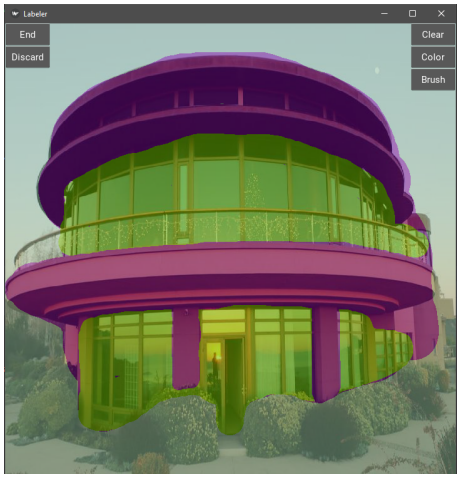


IV. Image Segmentation

For image segmentation, we created a tool to easily label each image with pixel-wise classes. For this task we selected three classes:

- 0 - Not building
- 1 - Opening
- 2 - Facade

To label the images we set out to create a painting tool to quickly set the class of every pixel within an image. This application browsed through a directory and created a numpy array containing the pixel-wise labels.



Even with the newly created tools to label images, labeling one image with precision is very time consuming, ranging from 2 minutes to 5 minutes for a single image. For that, we approached the training with the intention of minimizing the amount of samples as much as possible.

We started with a conservative 30 samples, and we began to train the model without data augmentation to see how quickly it overfitted. When the model started showing signs of overfitting, we increased the dataset size to 100 and started measuring the accuracy of the model. After that, we waited to see any signs of overfitting. We quickly saw a drop in accuracy. We added Gaussian noise and S&P noise, while addin random crop transformations. We quickly saw the accuracy rise, reaching a maximum of 80% accuracy. After the model started to overfit, we worked to add more precise samples to finetune the model. The change from rough and imprecise borders to more precise samples created a soon visible effect on the model. Decreasing the overall accuracy and a sudden increase in training loss, while maintaining the validation loss almost constant [5].

Seeing the model classify the pixels with 80% accuracy, we have concluded that not many labeled images are required to train an accurate model, but the model should be trained from the beginning using very precise samples, because the model quickly learns to incorrectly predict borders between classes and fine-tuning the model once it has already learned high accuracy is counter-productive.

V. Image Classification

To classify the material the building is made of we required simple photos of wall textures. Because of that, to avoid unnecessary manpower expended on labeling data, we implemented a downloader that assigns a label index for every string query and attempts to download as many images as requested from Google Images using the Google Images API.

Observing the different types of buildings we were observing, we decided on selecting as possible classes:

- Stone cladding
- Metal siding
- Bricks
- Render

After that we utilised the image fetcher to download around 450 images of each type. As model, we tried both ResNet 50 and ResNet 101 provided by the Torchvision library, both with Cross Entropy loss and an Adam optimizer. Because of the dramatic change in size between both models and our limited GPU memory, the ResNet 101 was trained with half the batch size. And even though it required much longer to train and it is more complex, it still provided worse results than the ResNet 50. 80% [6] accuracy opposed to 60% accuracy [7] in the same amount of iterations.

VI. Summary

The Torchvision Faster R-CNN was unable to achieve the desired results with a small batch size. Regardless, we are confident in the a Faster R-CNN's ability of achieving decent results with more fine-tuning and research. Both the image segmentation model and the image classifier show promising results even with limited computing resources. We can say that with sufficient precise data samples and computing resources, the framework would achieve its desired goal.

Acknowledgements

We would like to acknowledge the host lab for our project *Laboratoire dínformatique et mécanique appliquées à la construction*, which provided many resources including the building database of Zurich and a meeting space. In addition, we would like to personally thank our project mentor Alireza Khodaverdian for the advice and guidance throughout the course of this project. Finally, we would like to acknowledge the ML CS-433 Course professors and staff for giving us the freedom and support in conducting this project.

References

[1] R. G. J. S. Shaoqing Ren, Kaiming He, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Cornell University*, 2015.
[2] F. S. H. A. Liang-Chieh Chen, George Papandreou, "Rethinking atrous convolution for semantic image segmentation," *Cornell University*, 2017.
[3] S. R. J. S. Kaiming He, Xiangyu Zhang, "Deep residual learning for image recognition," *Cornell University*, 2015.
[4] "Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd and yolo)," *mc.ai*, 2018.

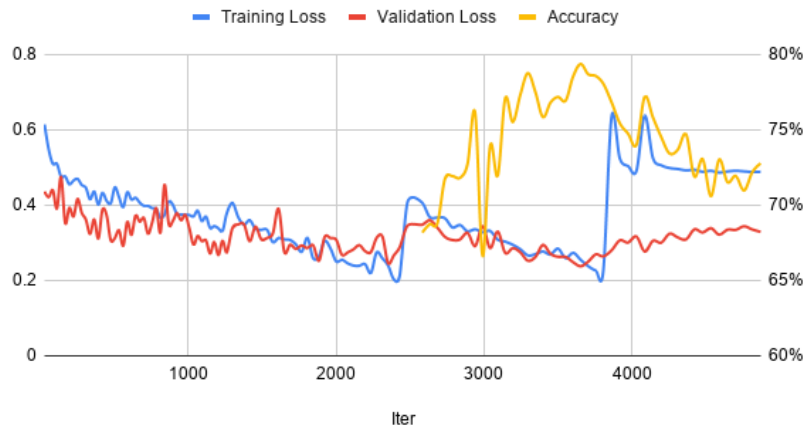


Figure 5. Image segmentation progress

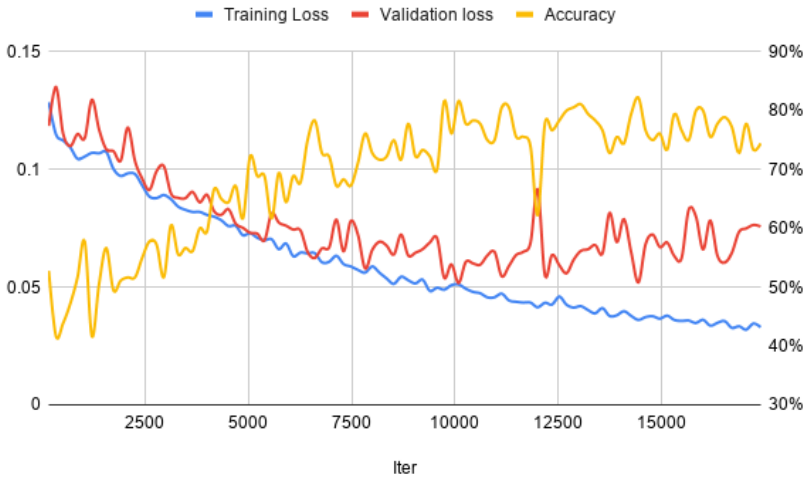


Figure 6. ResNet 50 Progress (Batch size 15)

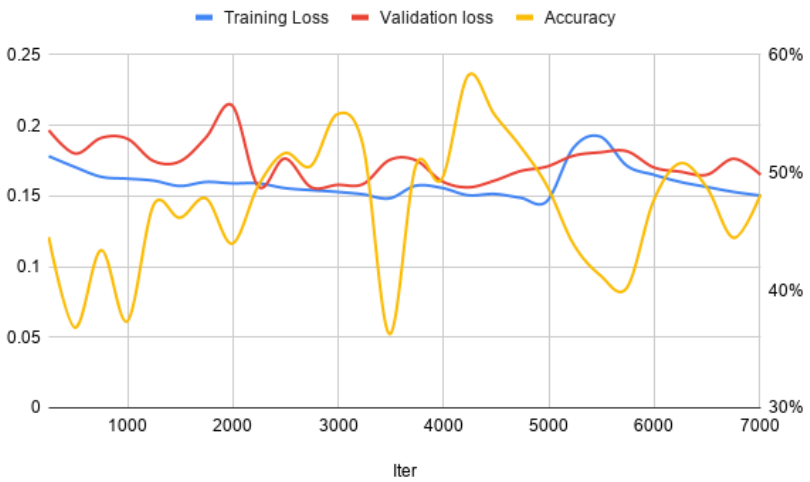


Figure 7. ResNet 101 Progress (Batch size 7)