Architectural Picture Classification

Wan Li

05/23/2018

This project is aimed to analyze a picture of a building in terms of the style of buildings and the composition of the photo. Though photos and renderings of buildings may have infinite possibilities, many of they can be classified into a limited number of popular types according to the composition, style of buildings, etc.

I roughly classified exterior pictures of (mid-rise to high rise) buildings into a few categories according to the scene, point of view, including street view, far away view, mid-air view, top view and look up view. There are usually other objects like people, cars, boats and animals in the picture, which I will also identify.

The judgement of a style of architecture is kind of subjective without very concrete standards now, and a person need to learn a lot about architectural history and current famous architects to tell the possible style of a building. I want to design an algorithm to identify the possible style of building in a photo or rendering. For this project, I just focus on about 10 common architectural styles.

Table of Contents

Step 1: Import and preprocess the two datasets

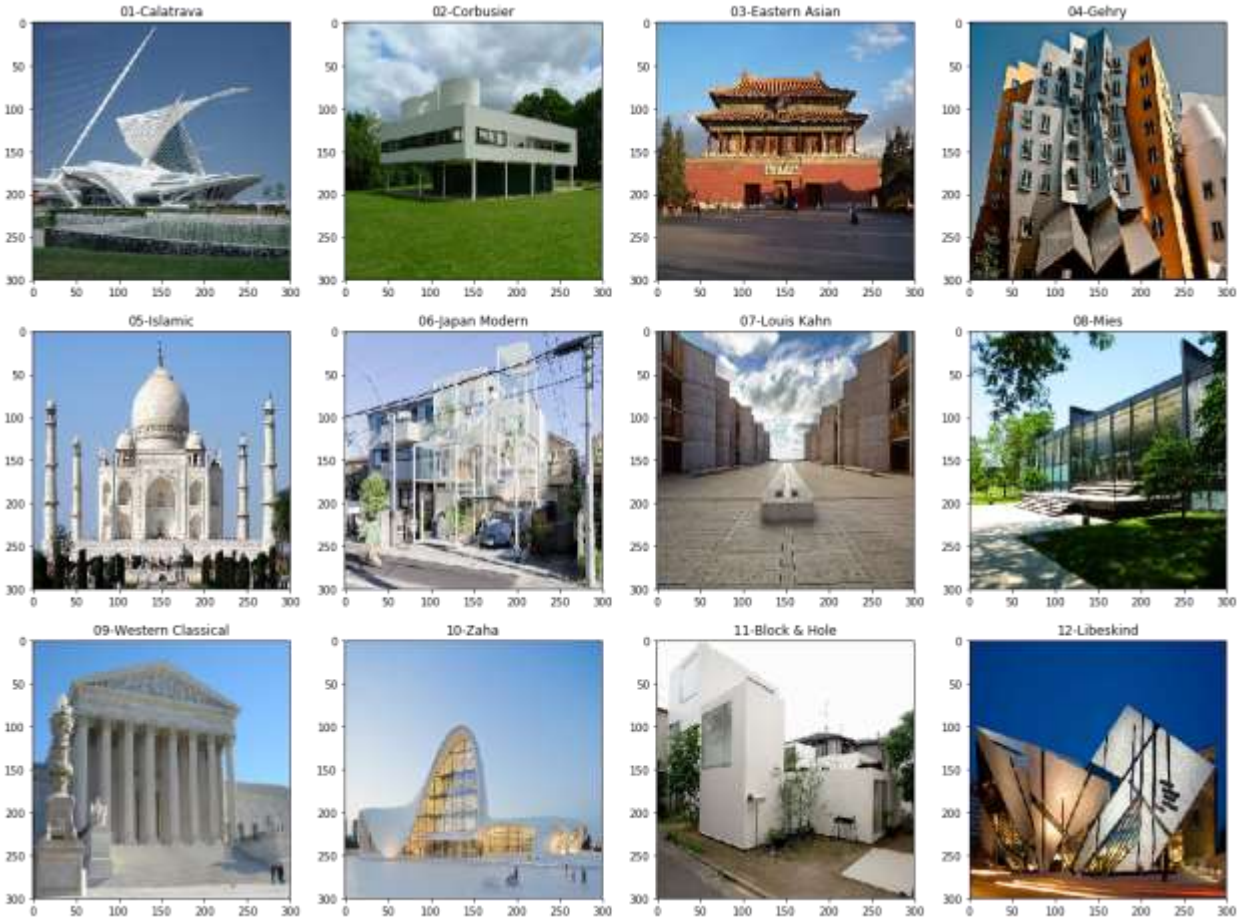
Import two datasets

The dataset *arc_scene* includes about 1000 pictures of buildings labeled according to its scene. The dataset *arc_style* includes 2000 pictures of buildings labeled according to its architectural style. I selected 12 styles that are far from inclusive, but they are very distinguishable and common in the architectural design industry. Each style has around 100 or 200 images.

Unlike traditional buildings, there is no clear criterion for classification of modern architecture. So I defined some styles based on knowledge of architecture. There are well known traditional architectural styles and the modern styles are mostly based on the work of a famous architecture with strong features.

```
There are 6 total scene categories.
There are 1007 total scene files.
There are 12 total style categories.
There are 1996 total style files.
```

```
12 styles:
```



Preprocess data

Turn images into 4D tensor, and split data into train, validation and test sets.

Augment data

Use ImageDataGenerator to augment arc_style train dataset.

Step 2: Use SVM to classify scenes

Use Principal Components Analysis to reduce the images to 40 principal components and use Support Vector Machine algorithm to classify the images.

The test result:

```
                              precision    recall  f1-score   support
```

```
 arc_scene\view1-roadview\       0.79      0.82      0.80       107
    arc_scene\view2-water\       0.62      0.56      0.59        18
   arc_scene\view3-midair\       0.62      0.63      0.62        46
      arc_scene\view4-top\       0.78      0.93      0.85        15
arc_scene\view5-lookingup\       0.71      0.42      0.53        12
     arc_scene\view6-part\       0.50      0.25      0.33         4

               avg / total       0.72      0.73      0.72       202
```

```
[[88  4 13  1  0  1]
 [ 4 10  3  0  1  0]
 [14  1 29  2  0  0]
 [ 0  0  0 14  1  0]
 [ 6  1  0  0  5  0]
 [ 0  0  2  1  0  1]]
```

Step 3: Write an algorithm with YOLO system to find the objects in the picture

YOLOv2 system is used for recognizing objects.

@article{redmon2016yolo9000, title={YOLO9000: Better, Faster, Stronger}, author={Redmon, Joseph and Farhadi, Ali}, journal={arXiv preprint arXiv:1612.08242}, year={2016} }

The model setting is:

'model': 'cfg/yolo.cfg',

'load': 'bin/yolov2.weights',

'threshold': 0.3

The objects and labels are shown in the image:

Step 4: Create a CNN from scratch to classify architectural styles

The model structure is:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 298, 298, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 149, 149, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 147, 147, 128)     36992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 73, 73, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 71, 71, 512)       590336
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12)                1548
=================================================================
Total params: 695,436
Trainable params: 695,436
Non-trainable params: 0
Ps: dropout=0.3
```

Use augmented arc_style train data to train the model 50 epochs with batch size of 8.

The result is:

```
Test accuracy: 56.7500%
Confusion matrix:
array([[24,  0,  2,  0,  0,  6,  0,  0,  1,  5,  0,  1],
       [ 1, 23,  1,  1,  0,  0,  1,  0,  0,  0,  2,  0],
       [ 0,  0, 35,  0,  0,  0,  0,  1,  0,  0,  0,  0],
       [ 6,  2,  7,  7,  3,  3,  0,  0,  0,  4,  4,  2],
       [ 1,  0,  1,  1,  9,  2,  0,  0,  0,  2,  2,  0],
       [ 2,  1,  4,  1,  0, 43,  0,  0,  1,  2,  6,  1],
       [ 1,  1,  6,  0,  0,  3,  3,  0,  4,  1,  0,  0],
       [ 4,  4,  0,  0,  0,  2,  0, 15,  1,  1,  1,  1],
       [ 0,  0,  2,  0,  3,  3,  0,  0, 21,  0,  0,  0],
```

```
       [ 8,  0,  3,  0,  0,  9,  0,  0,  2, 21,  2,  2],
       [ 0,  3,  1,  0,  0,  2,  0,  1,  0,  1, 17,  2],
       [ 4,  1,  1,  1,  2,  3,  1,  0,  1,  5,  0,  9]], dtype=int64)
```

From the confusion matrix, we can see some styles are more distinguishable than others. Some styles have quite subtle features that may be hard to tell, like Louis Kahn's works.

Step 5: Use transfer learning to create a CNN to classify architectural styles

Use the pre-trained VGG-19 model as a fixed feature extractor to obtain bottleneck features.

The VGG19 style recognition model architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 41472)             0
_________________________________________________________________
dropout_5 (Dropout)          (None, 41472)             0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               5308544
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 12)                1548
=================================================================
Total params: 5,310,092
Trainable params: 5,310,092
Non-trainable params: 0
```

Ps: dropout=0.3

Use the bottleneck features to train the model 60 epochs with batch size of 8.

Test the model with the test dataset. The result is:

```
Test accuracy: 71.50%
```

Confusion matrix:

```
array([[24,  0,  1,  3,  0,  1,  0,  0,  0,  9,  0,  1],
       [ 0, 23,  0,  0,  0,  1,  1,  1,  0,  1,  0,  2],
       [ 0,  0, 33,  3,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  2,  1, 33,  1,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  0, 18,  0,  0,  0,  0,  0,  0,  0],
       [ 2,  2,  3,  5,  0, 37,  3,  1,  0,  8,  0,  0],
       [ 2,  0,  0,  0,  0,  1, 10,  3,  0,  1,  2,  0],
       [ 0,  0,  0,  1,  1,  3,  0, 22,  0,  2,  0,  0],
       [ 0,  0,  3,  0,  3,  1,  0,  0, 22,  0,  0,  0],
       [ 2,  0,  4,  3,  0,  1,  0,  0,  1, 35,  0,  1],
```

```
[ 0,  0,  1,  5,  0,  5,  0,  0,  0,  0, 14,  2],
[ 2,  0,  1,  5,  0,  2,  0,  0,  0,  2,  1, 15]], dtype=int64)
```

- The trained using the pre-trained VGG19 model as a fixed feature extractor has a test accuracy rate of `71.50%`, which is quite good considering the styles can be ambiguous.
- From the confusion matrix, we can see that the classes that are often mixed are similar, like Calatrva and Zaha.

Step 6: Write an algorithm to analyze a picture of a building

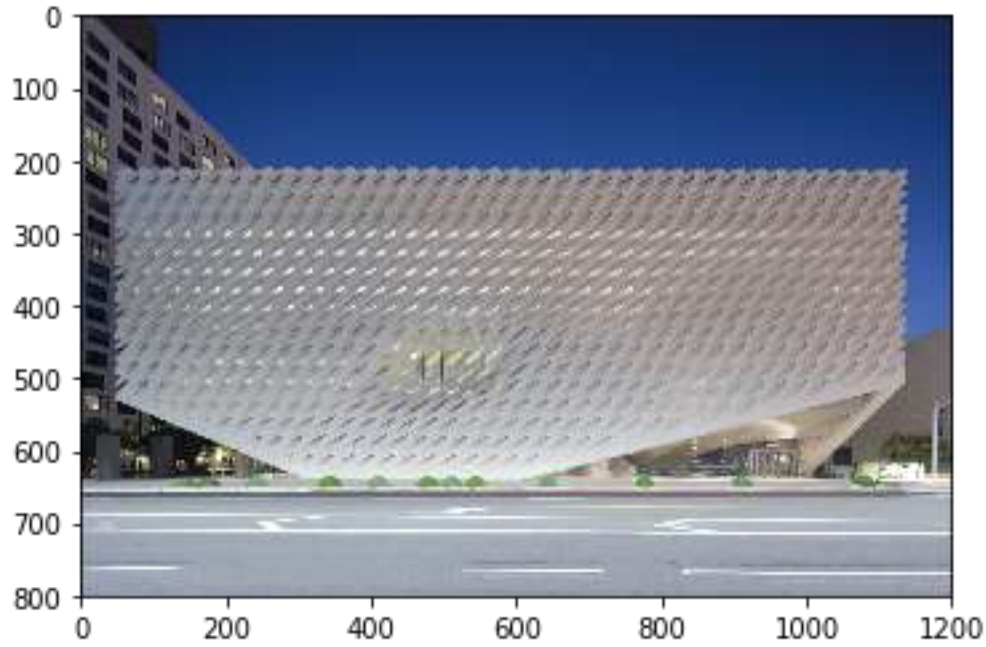
The algorithm contain three parts. First, it tell the view of the picture using the SVM algorithm from step 2. Then it recognize the objects and draw the frames and labels on the image with the YOLO system from step 3. Finally it tells the style of the building using the transfer learning model in step 5.

Step 7: Test the algorithm

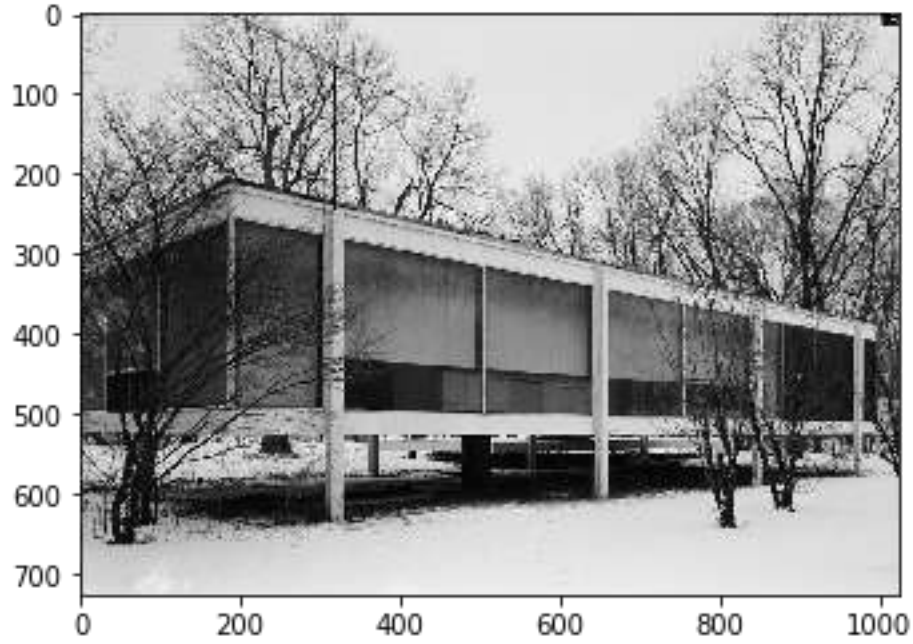
I applied the algorithm to some pictures.

The results:

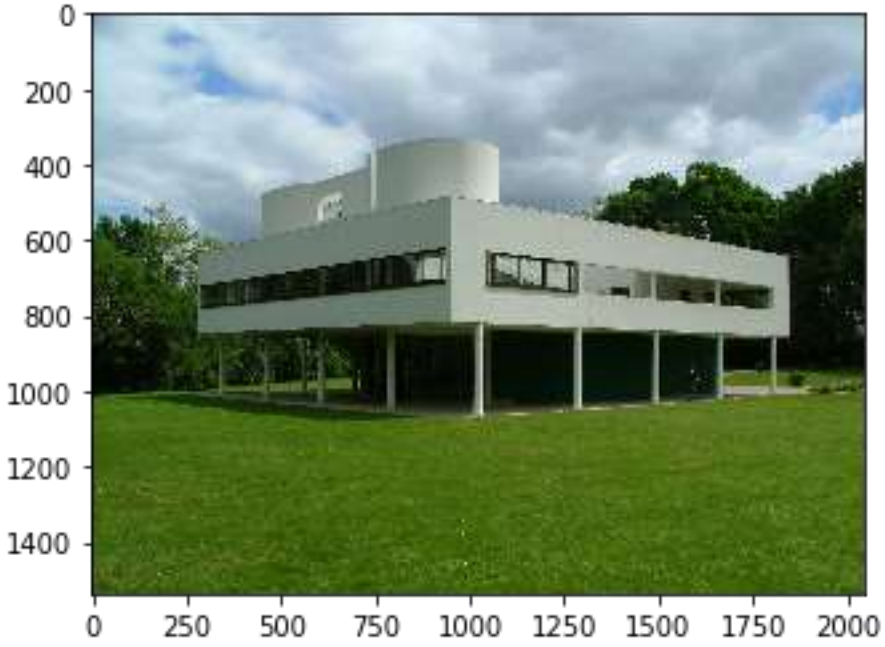

```
Scene:  roadview
Style:  Zaha
```



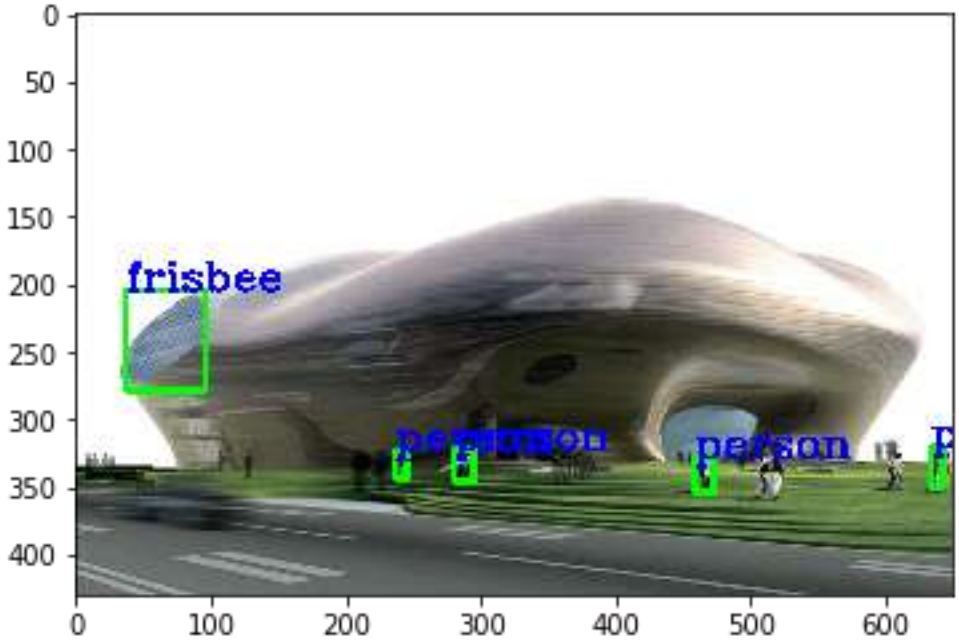
Scene: roadview
Style: Zaha



Scene: roadview
Style: Mies

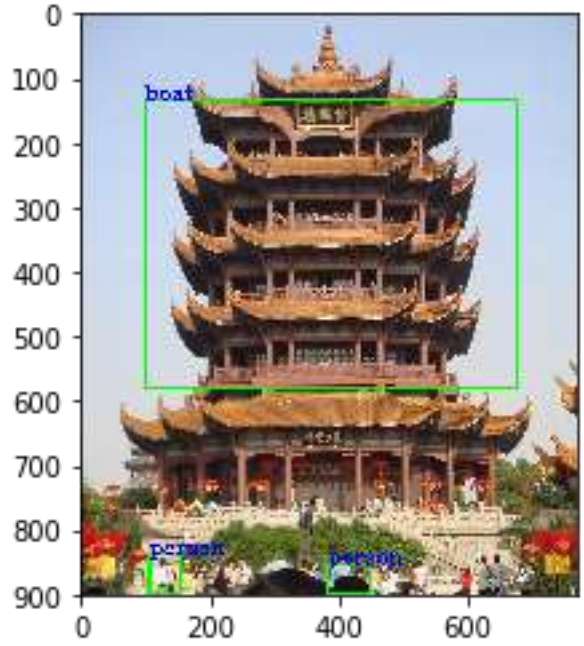


Scene: roadview
Style: Corbusier

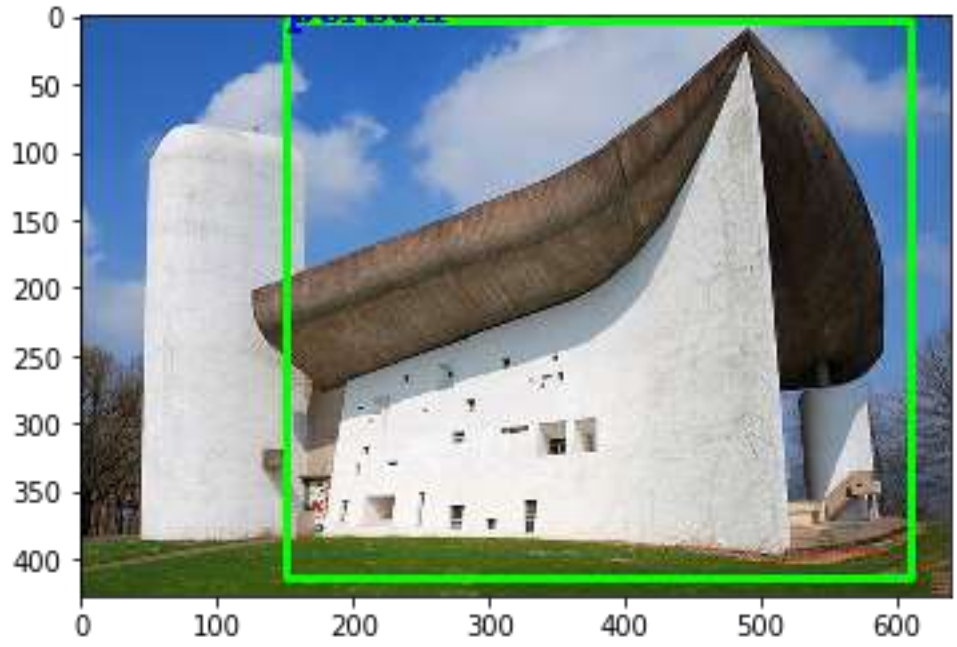


Scene: roadview
Style: Zaha

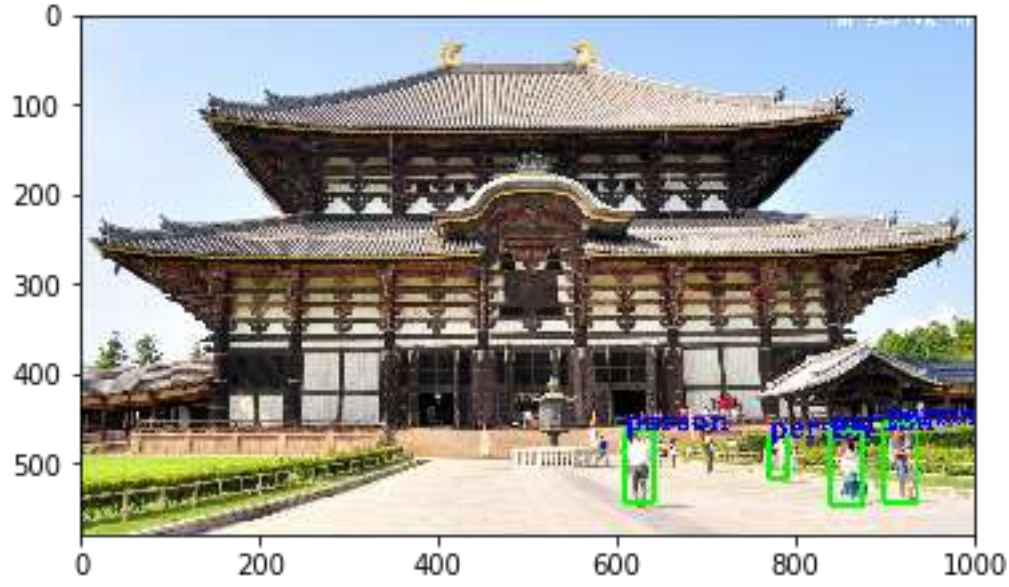
Scene: roadview
Style: Japan Modern



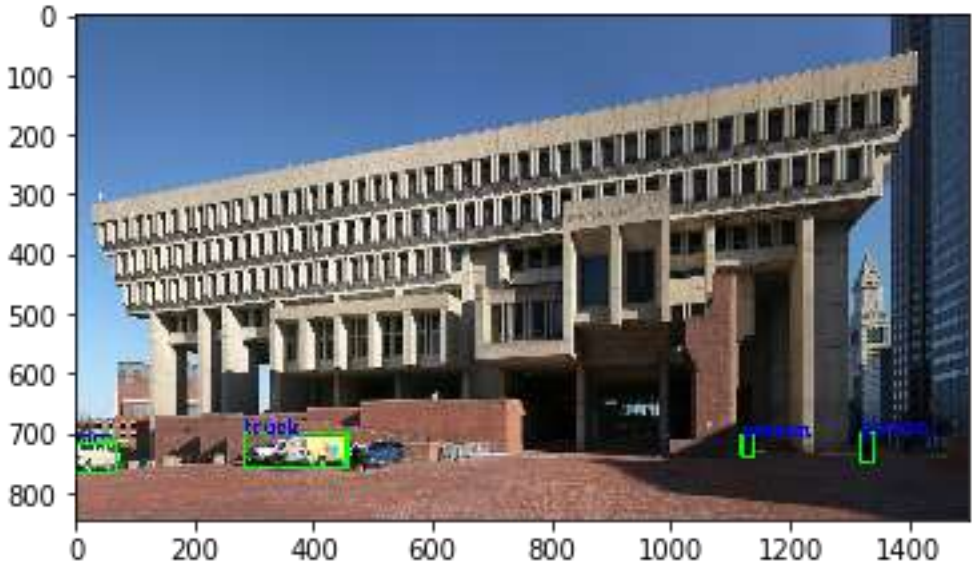
Scene: roadview
Style: Eastern Asian



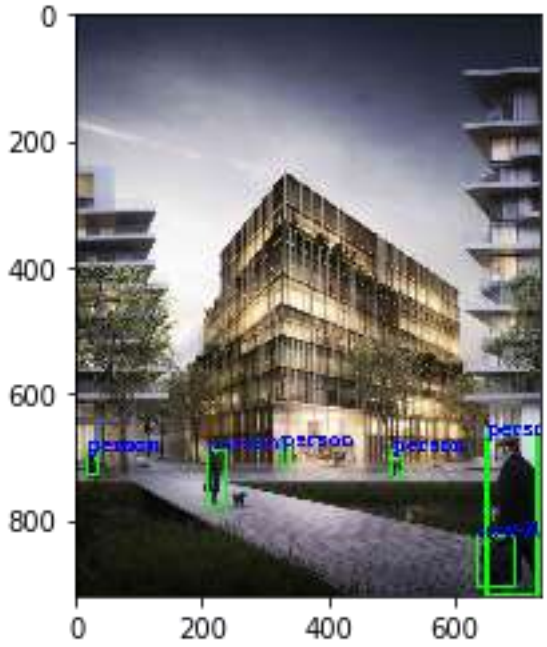
Scene: roadview
Style: Gehry



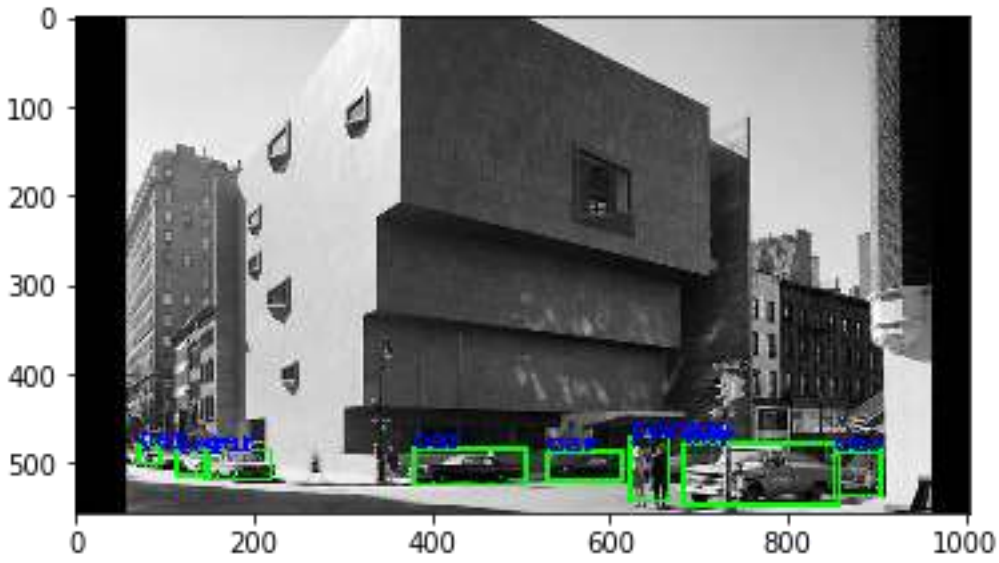
Scene: roadview
Style: Eastern Asian



Scene: roadview
Style: Corbusier



Scene: roadview
Style: Japan Modern



Scene: roadview
Style: Gehry

Scene: roadview
Style: Western Classical

- The algorithm can recognize traditional styles pretty well.
- The algorithm tends to recognize buildings as fancy styles like Zaha and Gehry, and has difficulty recognizing subtle styles like Japan Modern, Mies and Corbusier, which can be similar sometimes.
- Some parts of some buildings are mistakenly recognized as objects.