Machine Learning Engineer Nanodegree

Architectural Picture Classification

Wan Li
May 25, 2018

I. Definition

Project Overview

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 would roughly classify 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.

Judging the 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.

Based on this project, later I want to develop a project to help generate pictures of buildings based on input, view, objects and architectural style.

Relevant Research

There are some existing researches about classifying architectural styles and scenes. Many of them are about classifying traditional styles and scenes:
Classification of Architectural Heritage Images Using Deep Learning Techniques
Classifying U.S. Houses by Architectural Style Using Convolutional Neural Networks
Automatic Architectural Style Recognition
Training an Architectural Classifier
Research on Classification of Architectural Style Image Based on Convolutional Neural Network

Problem Statement

Given a picture of a building, I want the algorithm to tell the view of the picture, the obvious objects like people and cars and the most possible style of the building. The view will be among a few classes I defined and the style will be among 12 styles that I choose for the project. The obvious objects will give a name of the object and its coordinates.

This is an image recognition and classification problem. These problems should be solved by machine learning algorithms such as SVM, CNN, etc.

This report will focus on the architecture style classification.

Metrics

I would use accuracy rate and a confusion matrix as evaluation metrics to evaluate this machine learning classification performance. Accuracy rate is the rate of correct predictions. Confusion matrix will give detailed information about the classification predictions.

Another metric for the classification problem is log loss, also called cross-entropy loss. Cross-entropy loss takes into account the uncertainly of predictions and is used in training the neural network in this project. But it is difficult for people to get an intuitive understanding of a model's performance from the log loss value which doesn't have an upper bound. So accuracy and confusion matrix is better for demonstrating the performance of the model.

For this project, I will simply use the YOLO system as it is without training or evaluating the model statistically because the dataset is not labeled in terms of objects. I will test it with a few test cases.

II. Analysis

Data Exploration

I would mainly use two data sets for this project: arc_scene and arc_style. 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 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.

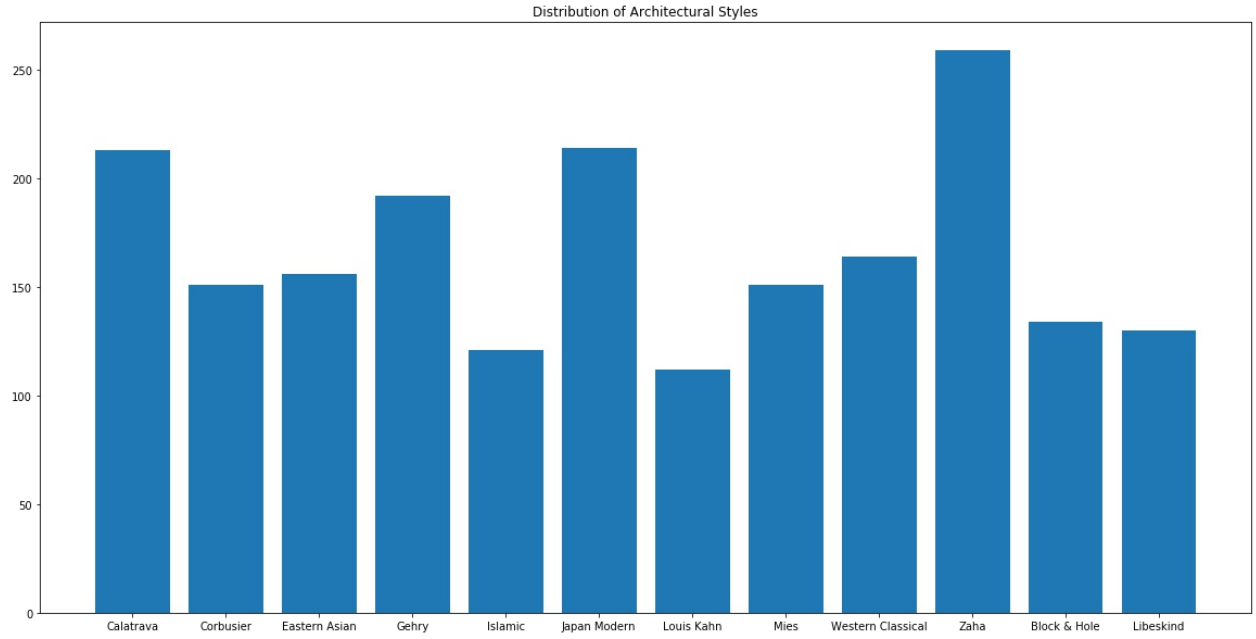
All the pictures have been manually collected from Google Images, Pinterest, Baidu Images and other websites and labeled by myself. These labeled pictures will be used as input for machine learning for classification.

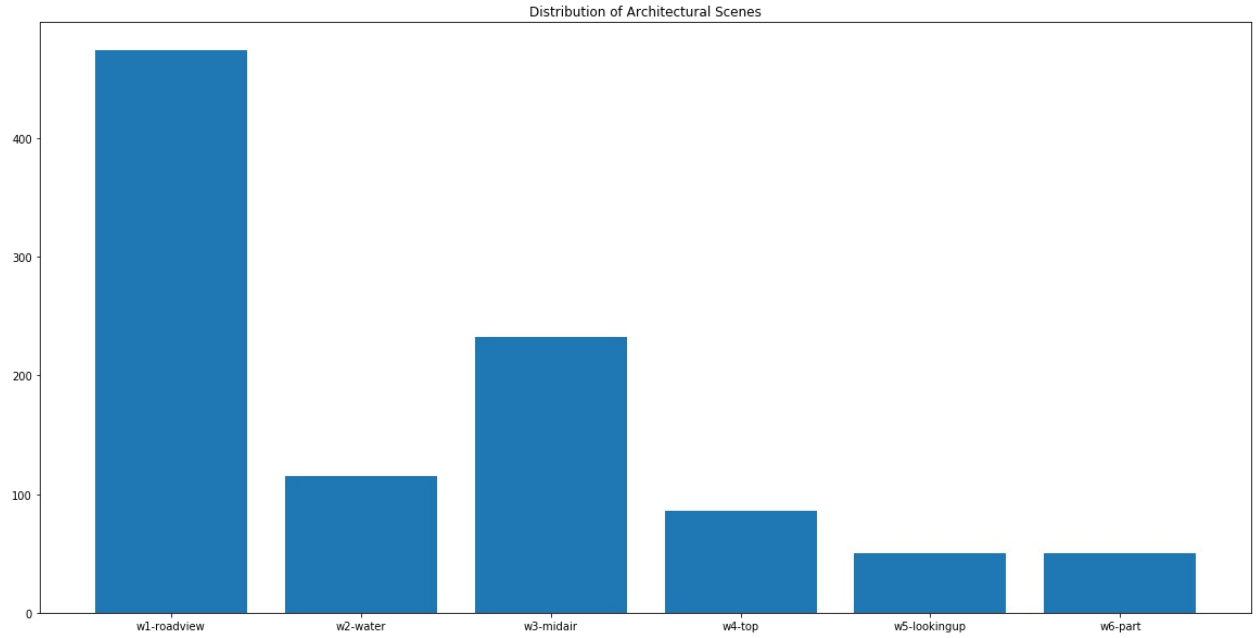
The images have varying sizes bigger than 300*300, and will be resized as 300*300 when loaded.

Exploratory Visualization

12 Architecture Styles in the Project:

This visualization shows all the architecture styles used in this project. Each picture is a typical example of its style. Each style has its visual features. But some styles share some common features, which the machine learning model may have difficulty distinguishing. For example, the works of Zaha and Calatrava both feature curves in their work, but the curve in Calatrava's works looks more structural. There are also many variations in the same style.



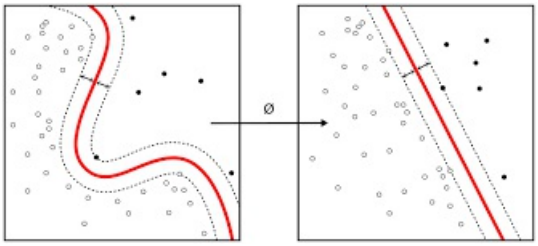


Algorithms and Techniques

SVMs (Support Vector Machines) and CNNs (Convolutional Neural Networks) are often used for learning images. Identifying the scene of image may be relatively simple. I will try SVM. To recognize objects like people and cars, I will use YOLO system to detect the objects in the picture. To predict the architectural style, I will use CNNs with transfer learning.

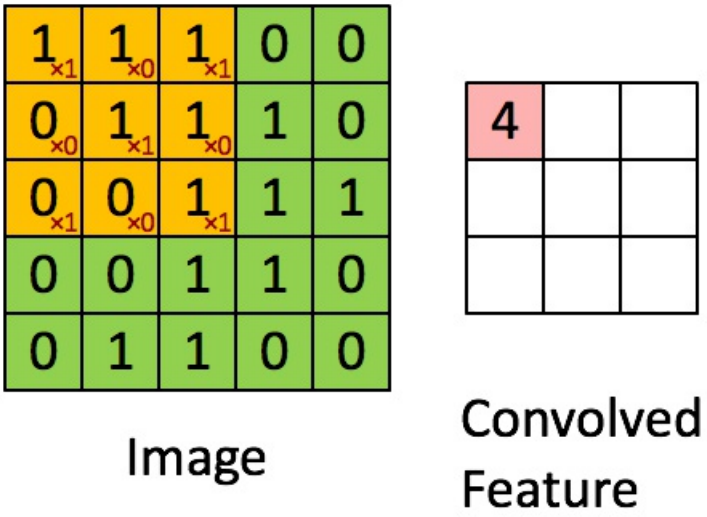
Support Vector Machines

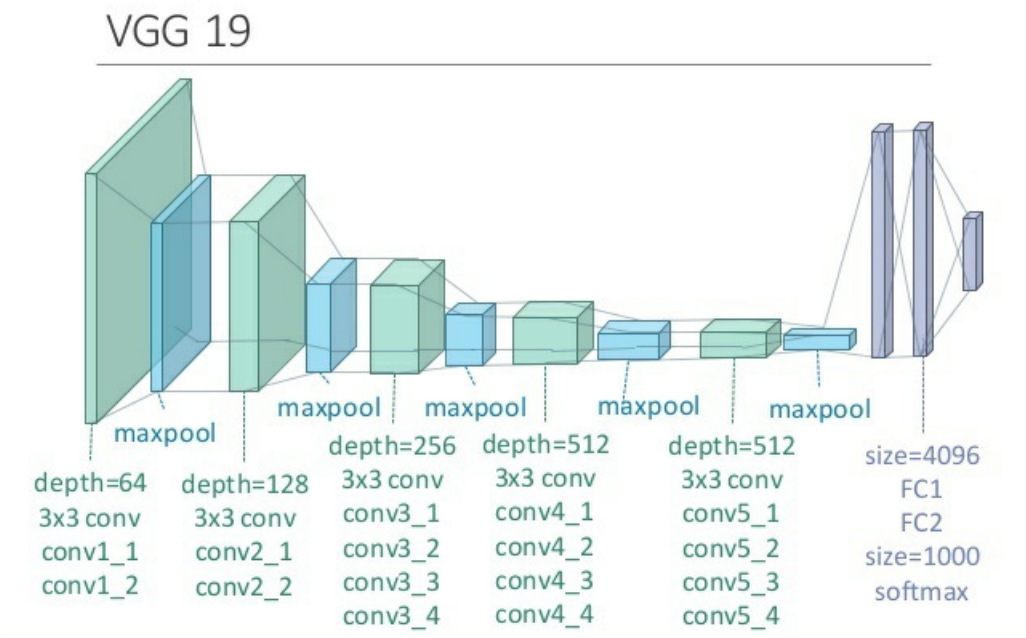
Support vector machine can construct a set of optimal hyperplanes in high dimensional space that separate the categories, which can be used for image classification.
These hyperplanes have the largest distance to the nearest training points of any class, which is called margin.
When the data are not separable linearly, SVMs can efficiently perform non-linear classification using kernel trick, implicitly mapping their inputs into higher-dimensional feature space.



Convolutional Neural Networks

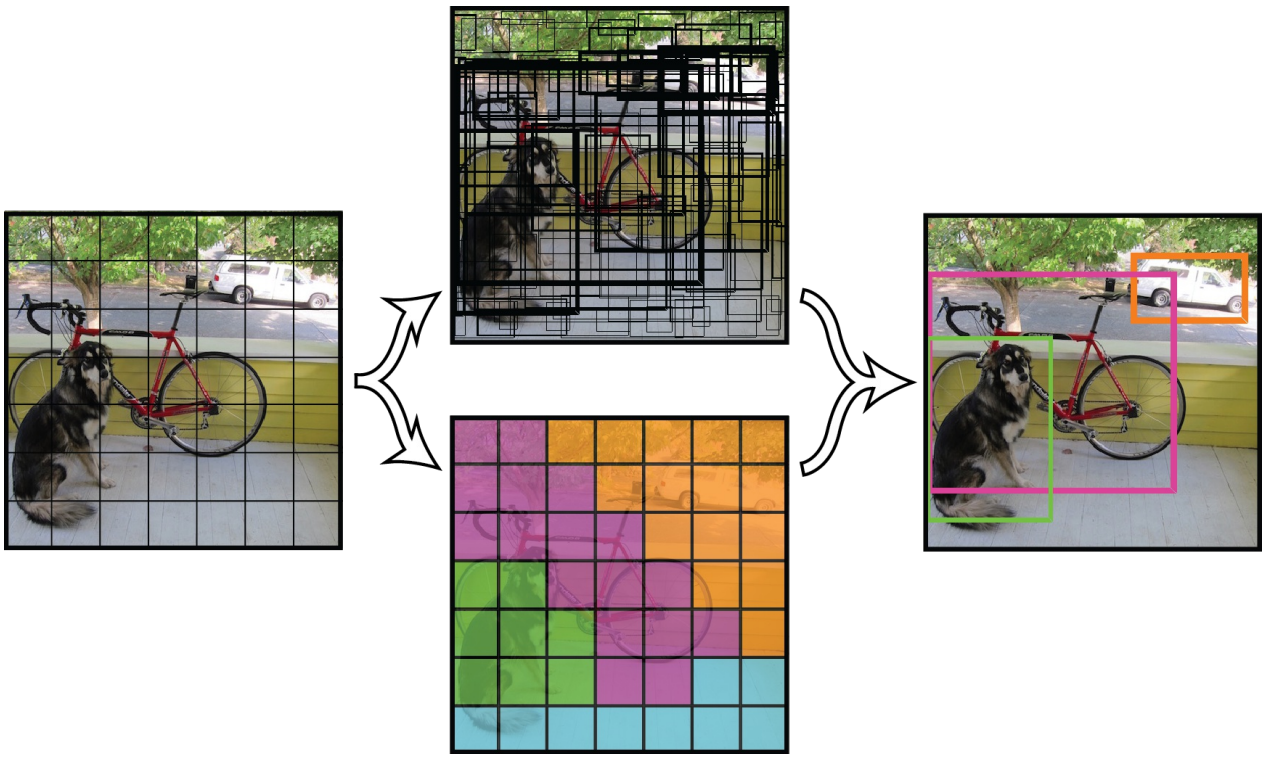
CNNs are a kind of Neural Networks. They are made up of layers of neurons that have learned weights and biases and can pass signal to neurons of next layer. CNNs have image data as input. While in regular Neural Networks one neuron is connected to all the neurons of adjacent layers, so-called fully-connected layer, the layers in CNNs are not fully connected. The neurons of a convolutional layer in CNNs are arranged in 3 dimensions. A neuron in a convolutional layer will only connect to a small region of the layer before it by a filter. Each filter recognizes some feature in the previous layer. Another main type of layer if pooling layer, which will reduce the number of parameters and amount of computation. Finally there will be some fully connected layers like the regular Neural networks. The final fully connected layer will compute the class scores to classify the images. There are some pre-trained CNNs for image classification, such as VGG and ResNet models.





YOLO System

YOLO is a state-of-art, real time object detection system. YOLO applies a single neural netw ork to the full image. This netw ork divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are w eighted by the predicted probabilities.



Benchmark

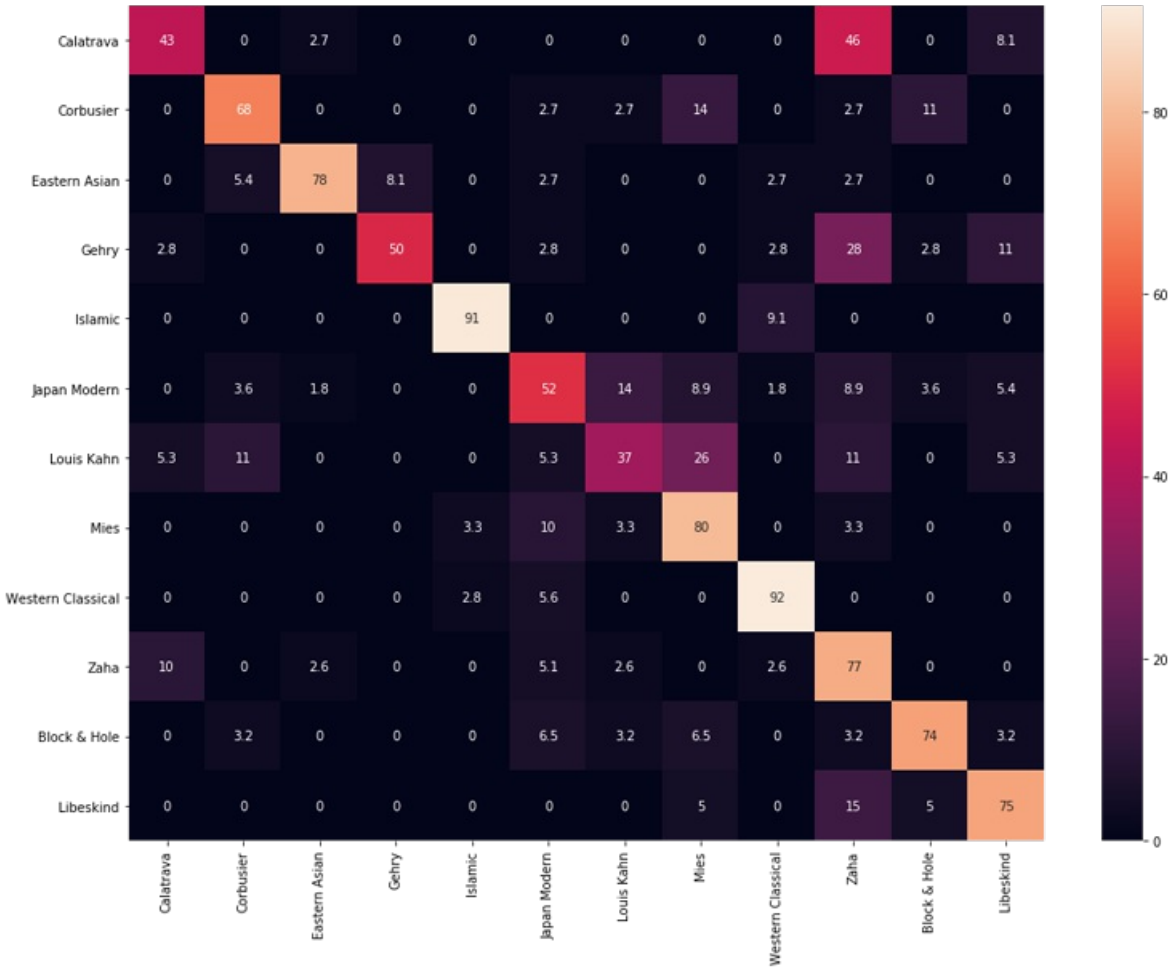
The benchmark model is a simple convolutional neural netw ork built by myself from scratch w ithout transfer learning.
I used augmented arc_style train data to train the model 50 epochs w ith batch size of 8.

```
_________________________________________________________________
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
_________________________________________________________________
```

The result is:

Test accuracy: 56.75%

Confusion matrix:

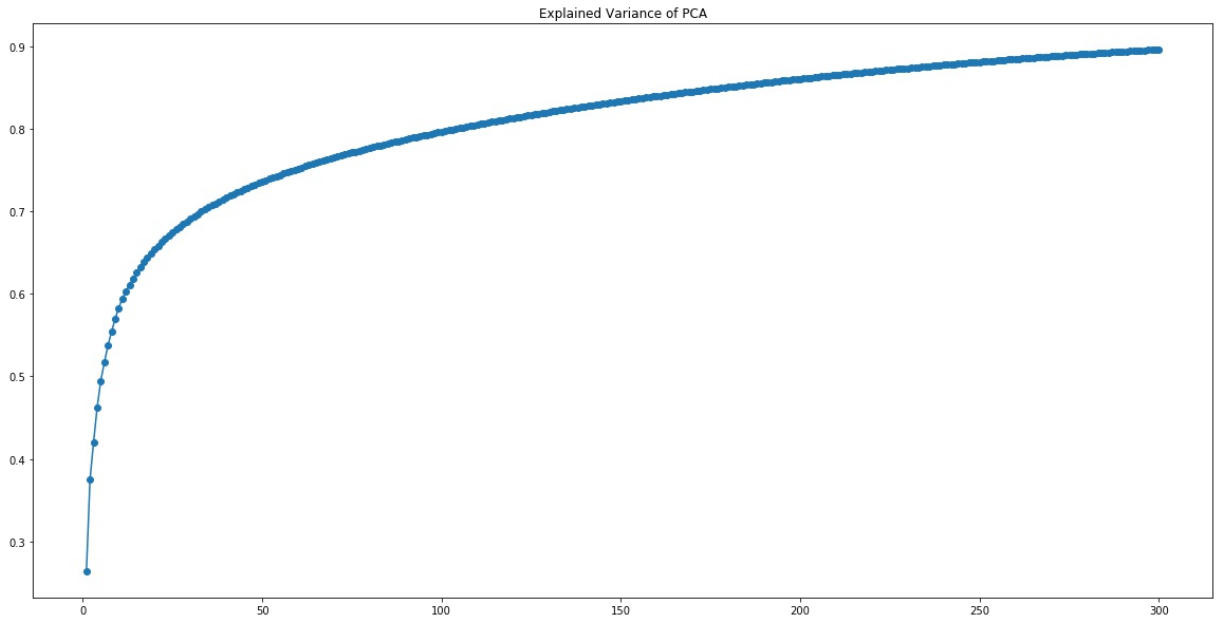


From the confusion matrix, we can see some styles are more distinguishable than others. Some styles have quite subtle features that may be hard to tell, like Louis Kahn's works.

III. Methodology

Data Preprocessing

- Load the images and resize them as 300*300, the original images vary and are bigger than 300*300.
- Turn images files into 4D tensor.
- Split the data into test, train and validation sets. 20% of the dataset makes the test dataset, which will be used at the end to show the performance of the model. In the remaining data, 85% is used for direct training and 15% is used for validating the model in the process of training.
- Create an `ImageDataGenerator` for data augmentation for the benchmark model. I used width shift (0.2), height shift (0.2) and horizontal flip.
- For the SVM algorithm to classify scenes, I implemented PCA to principal components. Based on the plot of explained variance of principal components, the explained variance didn't improve much after 50 components. I will pick below 50 principal components.



Implementation

Architecture style classification

- Import the pre-trained VGG-19 model (not including the top fully connected layers) and use it as a fixed feature extractor to get bottleneck features (9, 9, 512).
- Build a CNN model with one Flatten layer that takes the bottleneck features as inputs, two fully connected layers and 2 dropout layers in-between.
- Compile the model. Parameters: loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy']
- Train the model with bottleneck features 60 epochs with batch size of 8 and save the best model weights.
- Load the best weights and test the model with test data.
  The initial model:

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
Dropout: 0.3
Epochs=60
Batch_size=8

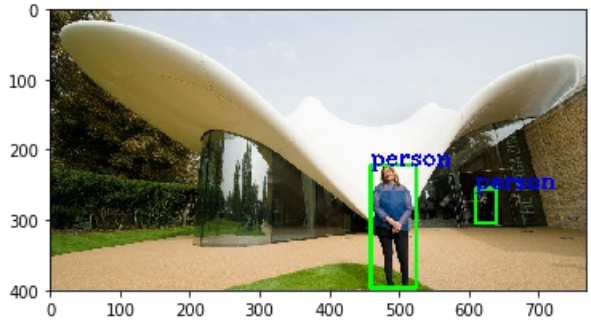
The result:
Test accuracy: 67.25%
** Extracting the bottleneck features may take some time. You can save the bottleneck features as `.npy` files for future use.*

Architecture scene Classification

- Tried a few numbers of principal components under 50 and finally use 40 principal components as input, which has a better result than others.
- Use grid search to train a SVM classifier with `rbf` kernel to classify the scenes and find the best values for `C` and `gamma`.
- The final precision is 0.72.

Objects recognition

- Install darkflow and import the YOLO model `yolo.cfg`.
- load the model's weights `yolov2.weights` and set the threshold to 0.3.
- Write an algorithm with OpenCV2 to draw the frame and label of all objects detected by YOLO model. The image below is an example.



To use YOLO system for python, follow guide on YouTube.
When using OpenCV2, convert the color of image: `img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`

Write the algorithm for the Project

- Write an algorithm to combine the above 3 functions to analyze a picture.

Refinement

The refinement will focus on the classification of architecture styles.
The initial solution has an accuracy of 67.25%.
For the initial solution, the training/validation loss in the process of training shows that the model tends to overfit. I tried the following techniques to improve the model.

- Change the Flatten layer to an GlobalAveragePooling2D layer to reduce the number of nodes to reduce the complexity of model.
- I tried different pre-trained models like VGG16 and ResNet50 to obtain bottleneck features, but the result is not as good as VGG19.

I used gridSearch to find the best value for following parameters:

- `batch_size` :[8,16,32]
- `epochs` :[50,70]
- `optimizer` : ['adam', 'rmsprop']
- `dropout` : [0.2,0.3,0.4]

Code:

```
def build_classifier(optimizer, dropout):
    classifier=Sequential()
    classifier.add(GlobalAveragePooling2D(input_shape=[9,9,512]))
    classifier.add(Dropout(dropout))
    classifier.add(Dense(256, activation='relu'))
    classifier.add(Dropout(dropout))
    classifier.add(Dense(12, activation='softmax'))

    classifier.compile(optimizer = optimizer, loss ='categorical_crossentropy', metrics = ['accuracy'])
    return classifier

classifier=KerasClassifier(build_fn=build_classifier)
parameters = {'batch_size':[8,16,32], 'epochs':[50,70],
              'optimizer': ['adam', 'rmsprop'], 'dropout': [0.2,0.3,0.4]}
grid_search=GridSearchCV(estimator= classifier, param_grid = parameters, scoring='accuracy', cv=6)
```

Complication:
When using GridSearchCV for the deep learning model, it will apply K-fold cross validation. So I combine the train data and validation data as the input data and it will apply K-fold split itself. The input labels should be one dimensional, so we can't use one-hot encoded labels. I applied `y_train=np.argmax(y_train, axis=1)` *to the labels.*

IV. Results

Model Evaluation and Validation

From the gridSearch, the best parameters are:
`{'batch_size': 16, 'dropout': 0.2, 'epochs': 70, 'optimizer': 'adam'}`
Best accuracy: 74.45%

The final model architecture:

Layer (type)	Output Shape	Param #
global_average_pooling2d_251	(None, 512)	0
dropout_503 (Dropout)	(None, 512)	0
dense_503 (Dense)	(None, 256)	131328
dropout_504 (Dropout)	(None, 256)	0
dense_504 (Dense)	(None, 12)	3084

Input: bottleneck features from VGG19 model
Total params: 134,412
Trainable params: 134,412
Non-trainable params: 0
Dropout: 0.2
Epochs: 70
Batch size:16
optimizer: 'adam'

I test the model with unseen test data set of 400 images. The final result is:
Test accuracy: 79.50%

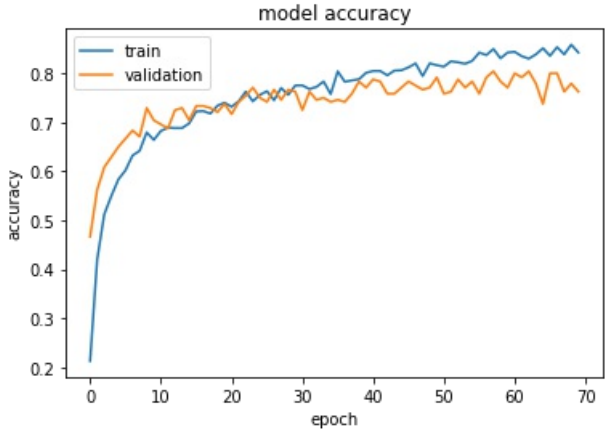
KFold validation

I used KFold(k=6) cross-validation to validate the the model in two different `random_state`s.
The results:
random-state=7
accuracy: 75.74%
accuracy: 76.67%
accuracy: 75.19%
accuracy: 70.45%
accuracy: 77.57%
accuracy: 74.43%
75.01% (+/- 2.27%)
random-state=123
accuracy: 76.84%
accuracy: 73.33%
accuracy: 73.31%
accuracy: 70.08%
accuracy: 74.14%
accuracy: 77.10%
74.13% (+/- 2.38%)

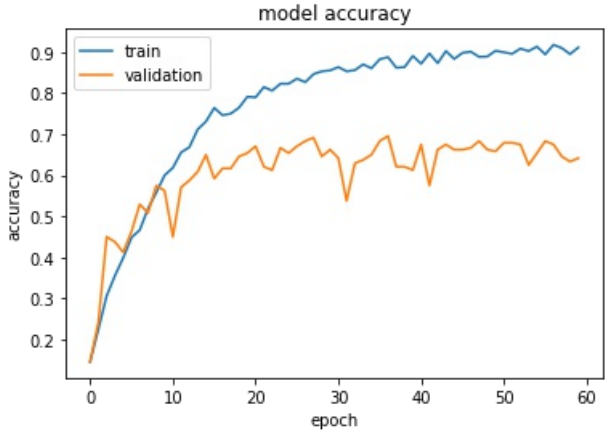
From the results, we can see the model is quite robust and stable since the difference between different folds and states are not big.

Learning Curve Plot

I trained and validated the model with the original train and validation data.
The learning Curve of Final Model:

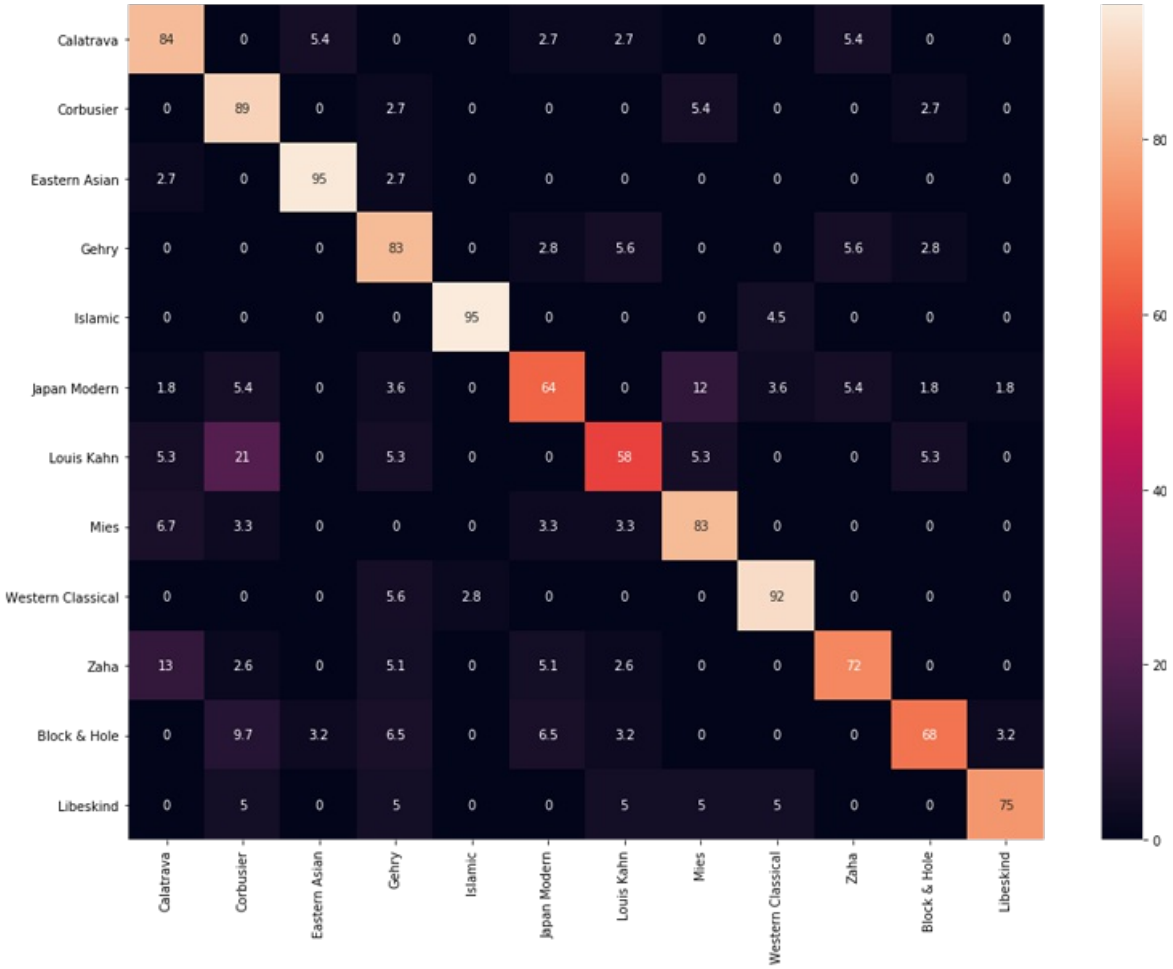


The learning Curve of Initial Model:



From the learning curve of the final model, the model may overfit a little, but not much. It has improved a lot from the initial model.

Confusion Matrix



- The final model trained using the pre-trained VGG19 model as a fixed feature extractor has a test accuracy rate of 79.50%.
- From the confusion matrix, we can see that the model can recognize traditional styles pretty well.
- The algorithm has difficulty recognize styles that are subtle like Louis Kahn, which can also be similar to other styles.
- The styles that are often mistaken for each other by the algorithm are often similar styles, like Corbusier and Kahn.

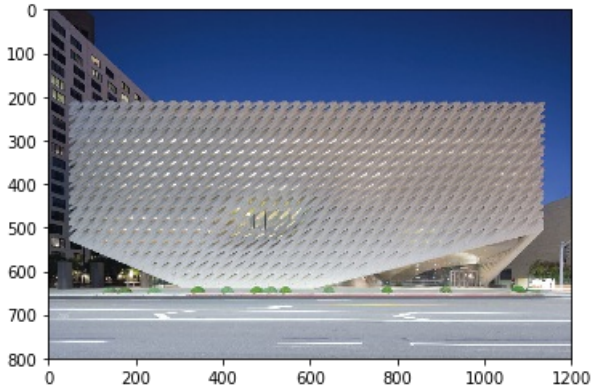
Justification

- The final test accuracy rate 79.50% is much better than the benchmark which is 56.75%.
- As architecture styles can be ambiguous and there are 12 styles to choose from, the final result is quite good. Sometimes when it classify a picture wrong, it can still give a result that is kind of close to its style.

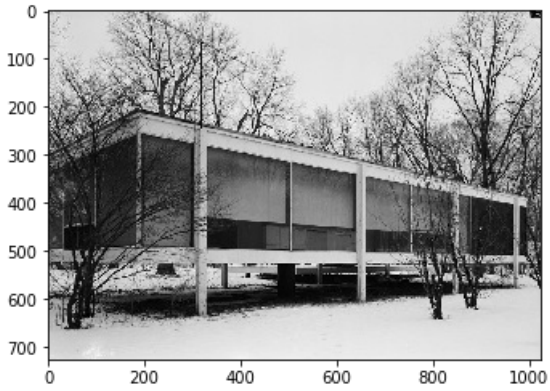
V. Conclusion

Free-Form Visualization

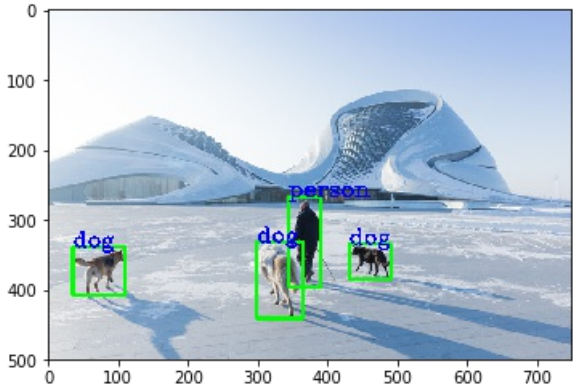
I applied the algorithm to some pictures.
The result:



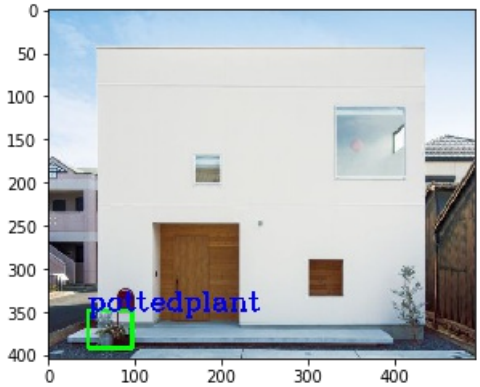
Scene: roadview
Style: Calatrava



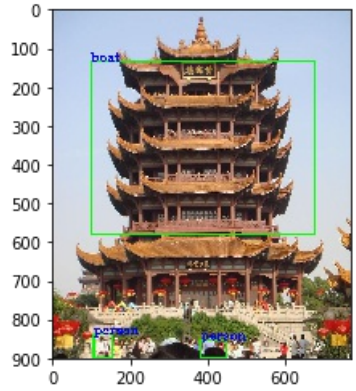
Scene: roadview
Style: Mies



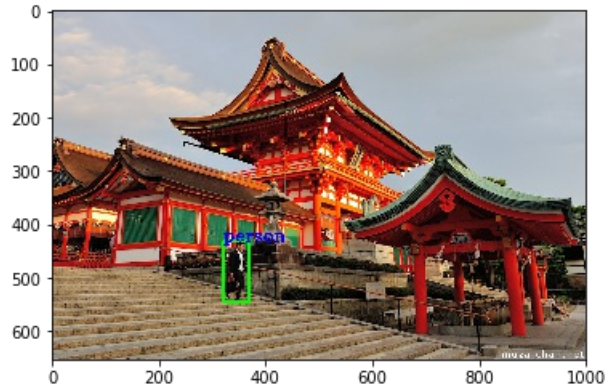
Scene: roadview
Style: Zaha



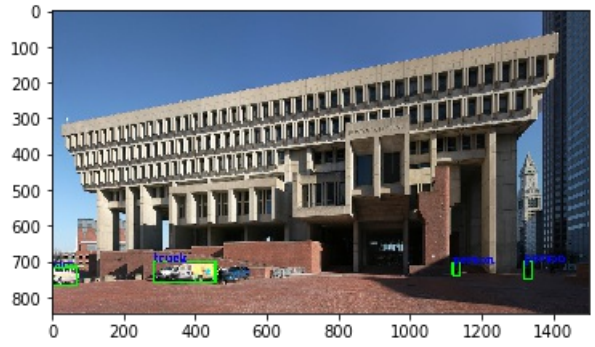
Scene: roadview
Style: Block & Hole



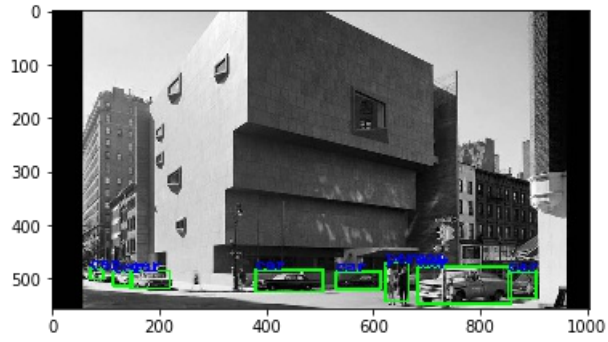
Scene: roadview
Style: Eastern Asian



Scene: roadview
Style: Eastern Asian

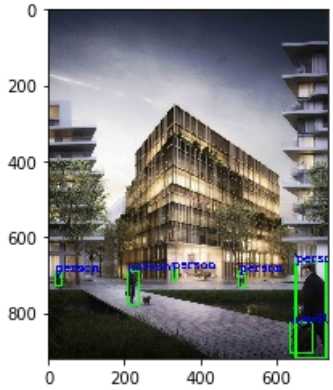


Scene: roadview
Style: Corbusier



Scene: roadview
Style: Corbusier

Scene: roadview
Style: Western Classical



Scene: roadview
Style: Japan Modern

Scene: roadview
Style: Western Classical

** Some classifications are wrong and some parts of some buildings are mistakenly recognized as objects.*
** Some pictures are not exactly one of these styles but algorithm will give a style that is close.*

Reflection

The process for the project can be summarized as the following steps:

Step 1: Import and preprocess the two datasets Step 2: Use SVM to classify scenes Step 3: Write an algorithm with YOLO system to find the objects in the picture Step 4: Create a CNN from scratch to classify architectural styles Step 5: Use transfer learning to create a CNN to classify architectural styles Step 6: Write an algorithm to analyze a picture of a building Step 7: Test the algorithm

- Creating a new dataset is quite difficult and time-consuming and can influence the outcome of project a lot. Partly because the `arc_scene` dataset is not very developed, I didn't spend too much efforts on learning scenes.
- Step 4 and step 5, machine learning of architecture styles, are the most critical steps for the project. I tried many different configurations to get the final test accuracy of 79%. I am quite satisfied with the outcome. Some buildings may be hard to judge even for people if the traits are not obvious.
- Since styles of modern architecture is not well defined, I classified some of them according to my knowledge. The result is quite interesting. From the

confusion matrix, we can analyze the similarity between different styles as well as how distinguishable an architect's works are.

Improvement

- The datasets are collected by myself in limited time and the classification of modern architecture doesn't have clear criterion. The dataset of arc_style has about 2000 files, which may not be big enough for deep learning. To improve the
project, the dataset should be bigger with more rigorous classification standards.
- As it is not easy to classify building pictures, I researched about unsupervised deep learning which I hope it can cluster the pictures by itself. I haven't tried unsupervised deep learning yet.
- In my project, I first built a CNN and trained it with augmented data as benchmark. Then I used transfer learning with bottleneck features extracted from pre-trained VGG19 model. If I use the final solution as the new benchmark, I will try fine-tuning to further improve the result. I will import the convolutional base of VGG19 and load the weights, add fully-connected model on the top, freeze the layers of the VGG19 model up to the last convolutional block, and train the model the augmented data.