Google Landmark Recognition Challenge

Hiten Nirmal
University of Georgia
Department of Computer Science
hn97292@uga.edu

Ankit Vaghela
University of Georgia
Department of Computer Science
ankitkumar.vaghela@uga.edu

Abstract

Image classification technology has shown remarkable improvement over the past few years, however- a great obstacle to landmark recognition research is the lack of large annotated datasets. This technology can predict landmark labels directly from image pixels to help people better understand and organize their photo collections. The project challenges to build models that recognize the correct landmark in a dataset of test images. We try to implement various deep learning models which can be used to train and predict a dataset of test images. The Kaggle challenge provides access to annotated data which consists of various links to google images along with their respective labeled classes. [3]

1 Introduction

In order to continue advancing the state of the art in computer vision, many researchers are now putting more focus on fine-grained and instance-level recognition problems – instead of recognizing general entities such as buildings, mountains and (of course) cats, many are designing machine learning algorithms that are capable of identifying the Eiffel Tower, Mount Fuji or Persian cats. However, a significant obstacle for research in this area has been the lack of large annotated datasets. With the vast amount of landmark images on the Internet, the time has come to think about landmarks globally, namely to build a landmark prediction engine, on the scale of the entire earth.[1]

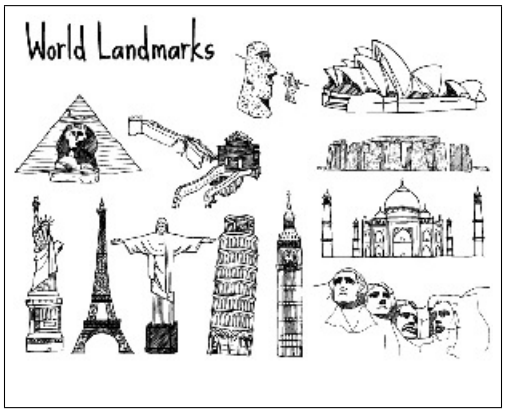


Figure 1: Somple popular and easily recognizable landmarks

Simple and most popular tourist landmarks are easily distinguishable since they are part of people tours which are further noticeable via their culture and history. But, that's not the case with **not so famous landmarks**. Such a large-scale landmark recognition engine is tremendously useful for many vision and multimedia applications. First, by capturing the visual characteristics of landmarks,

Submitted to CSCI 8360 Data Science Practicum, University of Georgia on 27th April 2018. Do not distribute.

the engine can provide clean landmark images for building virtual tourism of a large number of landmarks. Second, by recognizing landmarks, the engine can facilitate both content understanding and geo-location detection of images and videos. Third, by geographically organizing landmarks, the engine can facilitate an intuitive geographic exploration and navigation of landmarks in a local area, so as to provide tour guide recommendation and visualization.

To build such an earth-scale landmark recognition engine, the following issues, however, must be tackled:

- there is no readily available list of landmarks in the world;
- even if there were such a list, it is still challenging to collect true landmark images; and
- efficiency is a nontrivial challenge for such a large-scale system.[1]

Here, we try to tackle and deal with the first problem of increasing the amount of various different annotated landmarks.

1.1 Overall framework

A basic simple approach is to run a large scale model that can efficiently classify various landmarks. However, the size of the dataset is huge, which is further associated with the variety of unique classes. Thus we try to develop a model that filters out unwanted data like a face or some random objects like a dog or a car. This can be achieved by using some basic object detection models like YOLO or Tensorflow Object Detection API. For both the object detection algorithm there is a pre-trained model which is able to distinguish between some basic objects like a person, animal, cup etc. We could not integrate object detection model because of limited time constraint. After preprocessing the above data, we pass that into some basic convolutional network or some advance Deep Residual Networks. Once the model is trained efficiently we fit that in our test image dataset to retrieve the image landmark class along with confidence score.

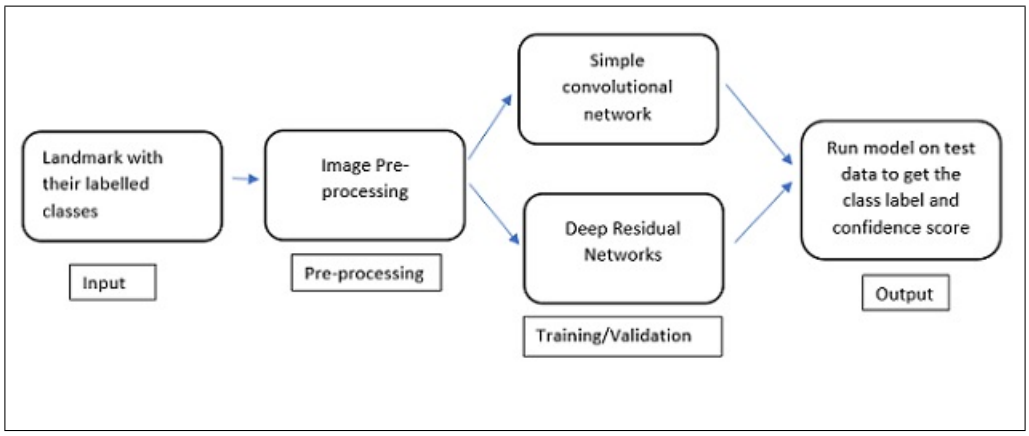


Figure 2: Architecture

2 Understanding the Data

The dataset for this Kaggle competition is available on the following website:

`https://www.kaggle.com/c/landmark-recognition-challenge/data`

The list of train images can be found in `train.csv` file which includes a large number of labeled landmark images along with their web-links. In training dataset, each image represents exactly one landmark.

Similarly, test images are found in `test.csv`. Test image may have no landmark, one landmark or sometimes more than one landmark prediction. Each image has a unique id (a hash) and each landmark has a unique id (an integer).[3]

File descriptions:

- train.csv - the training image set
- test.csv - the test set containing the test images for which we may predict landmarks
- sample submission.csv - a sample submission file in the correct format

There are overall approximately 1.2 million train images with 15,000 unique classes, whereas 0.1 million testing images for labeling and classification.

Count	Landmark_id	Count	Landmark_id
9633	50337	**2061**	13271
6051	50148	**5554**	11147
6599	23415	**6651**	9508
9779	18471	**6696**	9222

Figure 3: Some Popular landmark along with their count

After some data analysis, we found that top 1,000 popular classes(out of 15,0000) contributed to more than 70 percent of the image data. Also, there were some classes with just one image, which is too less to be considered as a training data.

Thus, one approach was to train the landmark based on most popular classes count and then later (if time permits before the competition deadline) we can train less famous class and combine the weights into one model.

One can also use the Python Script(available on the above-mentioned website,[3]) to download the image dataset into your local storage or VM instances.

3 Pre-processing

As discussed earlier, we used Tensorflow Object Detection API. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. It is based and trained on COCO API dataset. Also, Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be compiled. Although there are a limited number of objects and classes available in this pre-trained model, it that suffices our needs to detect a person and some other common objects.

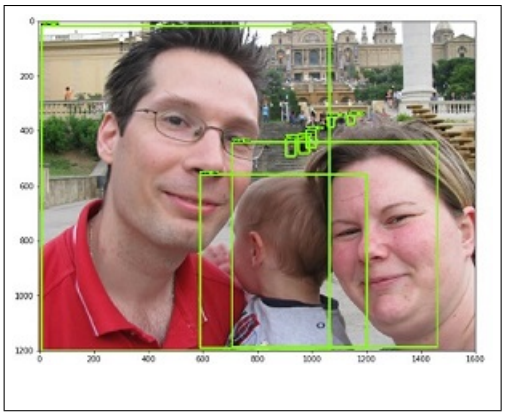
There have been such cases where the whole image in train data consists of some random people and there is not much information about the background landmark. We try to remove such redundant information from our training dataset thereby removing the redundant information. Also, face detection algorithm like `viola jones face detection` were considered, but failed to effectively implement on such a large dataset in a given amount of time.

3.1 Creating Numpy array

The whole pipeline in our project involves a tremendous amount of images (1.2 million). Thus resulting landmark prediction engine also incorporates a large number of landmarks and model images.

In order to simplify, we created a numpy array of images for each channel and another corresponding array to store the labels. This array was divided into 3 parts for simplistic reasons. The size of the image in the dataset varied from very small pixelated image to high-quality photographs. Thus the image resolution was **resized to 128*128** pixel per inch. All processing stuff was done on Google cloud- VM instance using Nvidia GPU. The Data is stored in VM-instance of the machine. Once, we are done with pre-processing of the data, the next step is to train that in a convolutional network model.

(a) Good Image with some knowledge of background landmark



(b) Bad Image with almost no details about the landmark

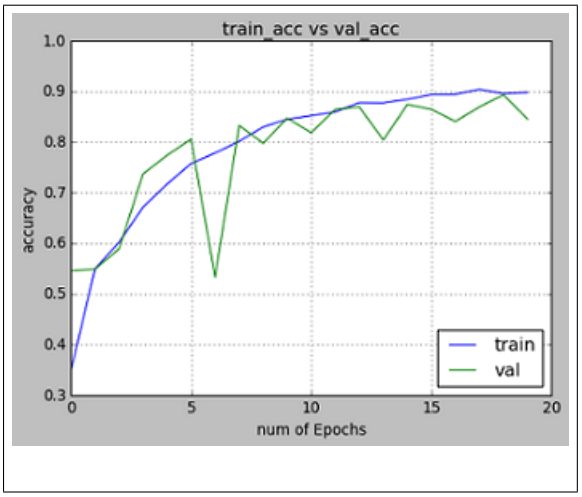
Figure 4: Object Detection

4 Training Model

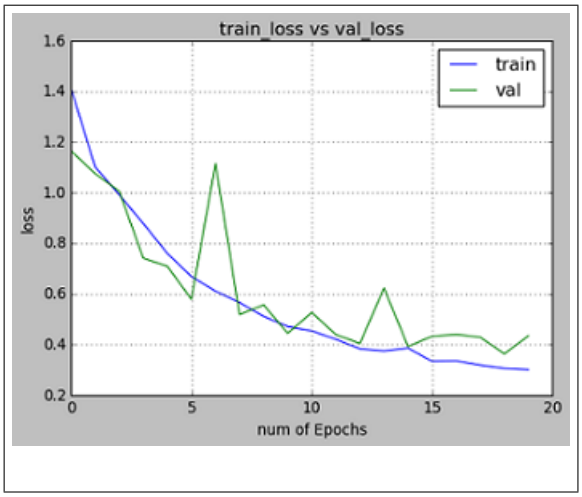
4.1 Simple Convolutional Network

While we were working on a small data, a simple convolutional network was used for classification purpose. This network was trained for some popular of classes and on a particularly targeted dataset. The network was 4 layer deep and was running on `epoch=20` . The `droupout value` was set to 0.5.

Train/Validation accuracy was around 0.85 and results seemed to be quite promising.



(a) Training Vs Validation Accuracy



(b) Training Vs Validation Loss

Figure 5: Graphical Comparison between Training and Validation

4.2 Residual Learning for Image Recognition

Deep convolutional neural networks have led to a series of breakthroughs for image classification. Many other visual recognition tasks have also greatly benefited from very deep models. So, over the years there is a trend to go more deeper, to solve more complex tasks and to also increase /improve the classification/recognition accuracy. But, as we go deeper; the training of neural network becomes

difficult and also the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems.[2]

ResNet50 is a 50 layer deep Residual Network.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 6: Resnet architecture- based on 50 layer sample (snippet from paper)[2]

For the network:

- inputshape: The input shape in the form (channels, brows, cols)
- num outputs: The number of outputs at final softmax layer
- block fn: The block function to use. This is either ‘basic block‘ or ‘bottleneck‘.
- The original paper used a basic block for layers < 50 repetitions: Number of repetitions of various block units. At each block unit, the number of filters is doubled and the input size is halved
- Both bottleneck and basic residual blocks are supported. To switch them, simply provide the block function here.

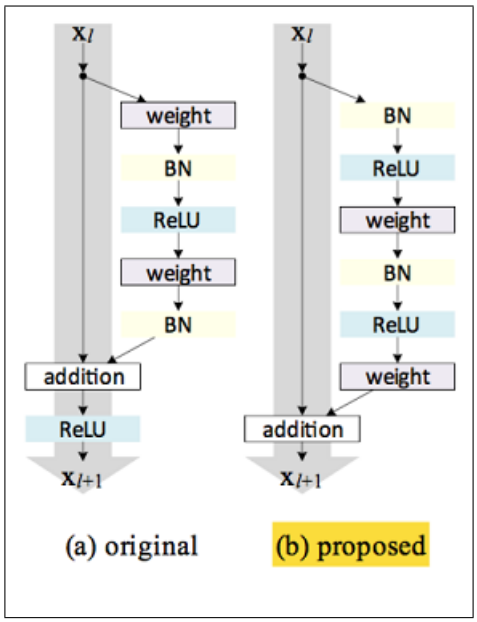


Figure 7: The residual blocks are based on the new improved scheme proposed in Identity Mappings in Deep Residual Networks as shown in figure b (snippet from paper)[2]

5 Obstacles Overcome

- **Large volume of training data:** Firstly, we had to use some efficient infrastructure to handle that data. GCP(Google cloud computing) came to the rescue. Two Nvidia Tegra K80 GPU were constantly running throughout one whole week.
- **Unavailability of images:** There were around 9,000 images from the training dataset which were not available to download from the web link. We just ignored that, however, there were around 2,500 images not available in test data set. We labeled them blank prediction, an idea we came up with after reviewing the Kaggle discussion board .
- **Pupular Landmark Classes:** In pre-processing we found out that nearly top 1000 labels(out of nearly 15,000 total labels), compromised 70 percent of the dataset. Thus we trained only limited number of classes, thereby ignoring the classes having very few images for our model to train on.

6 Post Processing

Once we are done with training the model, we just need to fit that model in our test numpy array and then classify the corresponding test images.

For each id in the test set, you can predict at most one landmark and its corresponding confidence score. Some query images may contain no landmarks. We may decide not to predict any result for a given query, by submitting an empty prediction. The submission file should contain a header and have the following format (larger scores denote more confident matches):

| id,landmarks

000088da12d664db,8815 0.03

0001623c6d808702,5523 0.85

0001bbb682d45002,5328 0.5 |

Final Submission is evaluated on kaggle. One can submit the results on the following website:

`https://www.kaggle.com/c/landmark-recognition-challenge/submit`

Submissions are evaluated using Global Average Precision (GAP) at k, where k=1. This metric is also known as micro Average Precision (microAP). It works as follows:

For each query image, we will predict one landmark label and a corresponding confidence score. The evaluation treats each prediction as an individual data point in a long list of predictions (sorted in descending order by confidence scores), and computes the Average Precision based on this list.[3]

If a submission has N predictions (label/confidence pairs) sorted in descending order by their confidence scores, then the Global Average Precision is computed as:

$$GAP = \frac{1}{M} \sum_{i=1}^{N} P(i) rel(i)$$

where:

- N is the total number of predictions returned by the system, across all queries
- M is the total number of queries with at least one landmark from the training set visible in it (note that some queries may not depict landmarks)
- P(i) is the precision at rank i
- rel(i) denotes the relevance of prediciton i: it's 1 if the i-th prediction is correct, and 0 otherwise

6.1 Final Output/Analysis

Out of 1.2 million images, we trained top 0.6 million popular landmark image classes using Residual Learning.

Our Final Kaggle Rank is 134 out of 309 participants [as on 27th April 16.00] with a score of 0.003

7 Future Work

We plan to continue work on this project until the deadline of the competition. Some of the things we plan to implement are as follows:

- Integrate `object detection model` with better pre-processing techniques to detect sharp edges of the object rather than a block.
- Train model on more images and add the weights to current trained model.
- Try to implement various other models like VGG16, InceptionResNetV2.
- Lastly, ensemble and run two or more classification models.

8 Conclusion

Here, we build a large-scale landmark prediction engine, which organizes, models and recognizes the landmarks. Constructing such an engine is a multi-source and multi-modal data mining task. This project shows that we can build an acceptable classification engine with good efficiency, hope this would help researchers to get annoted data about the 'not so famous' and 'famous' landmark images; advance steps towards computer vision and image classification.

Acknowledgments

This work would not have been possible without the support of Dr. Shannon Quinn, [Assistant Professor, University of Georgia Departments of Computer Science and Cellular Biology] who worked actively to provide us with academic time and advice to pursue those goals.

References

[1] Yan-Tao Zheng, Ming Zhao, Yang Song, Hartwig Adam Ulrich Buddemeier, Alessandro Bissacc, Fernando Brucher Tat-Seng Chua, and Hartmut Neven(2010) Tour the World: building a web-scale landmark recognition engine. *NUS Graduate Sch. for Integrative Sciences and Engineering, National University of Singapore, Singapore*, *Google Inc. U.S.A*

[2]Kaiming He, Xiangyu Zhan,g Shaoqing Ren, Jian Sun(2015) Deep Residual Learning for Image Recognition *arXiv:1512.03385v1 Microsoft Research*

[3] https://www.kaggle.com/c/landmark-recognition-challenge/data

[4] https://www.kaggle.com/c/landmark-recognition-challenge

[5] https://github.com/raghakot/keras-resnet *raghakot/keras-resnet*

[6] https://github.com/tensorflow/models/tree/master/research/object_detection *Tensorflow object detection model*