Landmark Recognition

by

Sangeetha Verkot

Dr. Daniel Ayala Rodriguez, Advisor

Masters project submitted in partial fulfillment of the

requirements for the Master of Science in Data Science degree

in the College of Arts and Sciences of

Lewis University

Google Landmark Recognition Challenge

Sangeetha Verkot
Computer and Mathematical Sciences Department
Lewis University
Romeoville, USA
sangeethamverkot@lewisu.edu

***Abstract*—Convolutional Neural Networks have arguably been one of the most influential innovations in the field of Computer Vision. They play an integral part in Deep Learning and Image Processing by pushing the frontier of visual recognition technology. In this paper, I explore using the Convolutional Neural Networks architecture known as MobileNets to correctly identify landmarks in an image. MobileNets, developed by researchers at Google for TensorFlow, is a family of mobile-first computer vision models. MobileNets is small, low-latency, low-power models parameterized to meet the resource constraints of handheld devices such as mobile phones or tablets yet powerful enough that they can be used for classification, detection, embeddings and segmentation. MobileNets is trained to identify 1000 different classes from the ImageNet dataset, and I will be using a technique known as transfer learning to retrain the model to recognize landmarks from Google's 'Landmark Recognition Challenge' dataset.**

Keywords—image recognition, convolutional neural networks, computer vision, MobileNets, inception, transfer learning, TensorFlow

I. INTRODUCTION

Modern day cell phones have substantial computational power and they are capable of taking high quality digital images. When we travel, we take hundreds of images of landmarks and store it on our mobile phones. While we are all familiar with popular attractions such as the Eiffel Tower or the Colosseum, a lesser known museum or a temple would be hard to recognize in your photo gallery. Thus, it is useful to have a tool that can recognize, with high accuracy, landmarks at world-scale that can run on the resource constraint environment such as a mobile phone or tablet. In this paper, I will explore using a technique known as transfer learning to retrain a model known as MobileNets to recognize landmarks in an image [6].

The biggest obstacle in Landmark Recognition was the availability of a large annotated dataset. Kaggle teamed up with Google to provide the largest annotated dataset to date for the Landmark Recognition Challenge [2]. Due to restrictions on distributing the actual files, the dataset contains a URL for each image. The training set was constructed by clustering photos with respect to their geolocation and visual similarity using an algorithm like the one proposed by Yan-Tao Zheng, Ming Zhao et al in their paper "Tour the World: Building a Web-Scale Landmark Recognition Engine" [1]. Matches between training images were established using local feature matching. The image below shows the first few rows of the dataset.

	id	url	landmark_id
0	cacf8152e2d2ae60	http://static.panoramio.com/photos/original/70...	4676
1	0a58358a2afd3e4e	http://lh6.ggpht.com/-igpT6wu0mIA/ROV8HnUuABI/...	6651
2	6b2bb500b6a38aa0	http://lh6.ggpht.com/-vKr5G5MEusk/SR6r6SJi6mI/...	11284
3	b399f09dee9c3c67	https://lh3.googleusercontent.com/-LOW2cjAqubA...	8429
4	19ace29d77a5be66	https://lh5.googleusercontent.com/-tnmSXwQcWL8...	6231

Figure 1 Snapshot of the Training Dataset

It contains a unique ID for each image, a link to the actual location of the landmark image, and a landmark_id which is essentially the label that the model will be trained to recognize for this project.

A python script was used to download the compressed version of the images and then stored on my computer. Another script was written to organize the images into folders based on their label. The training dataset contains 14,951 different landmarks and a total of 1,225,029 images. The test dataset has 117703 images.

An exploratory analysis of the dataset shows that folders 6051 and 9633 contain over 50,000 images each, but most the folders contain 10,000 or fewer images. Also, the dataset contains no missing data.

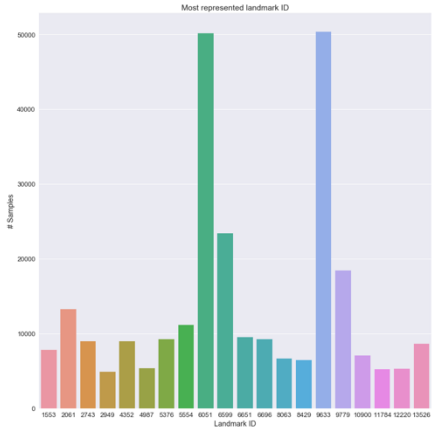


Figure 2 Most Represented Landmarks

A quick look at the folder '9633' shows that it contains images of St. Peter's Basilica from various of angles.

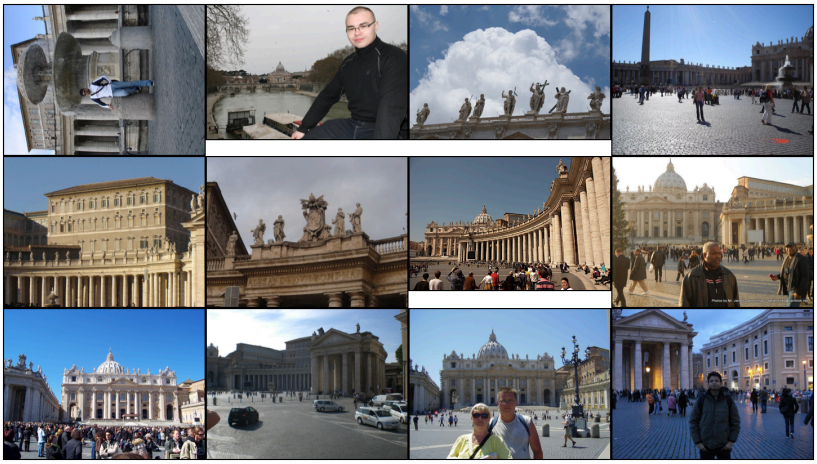


Figure 3 Images from folder 9633

Further analysis shows that:

- 25% of the classes contain less than 6 images in it.
- 50% of the classes contain less than 14 images.
- 75% of the classes contain less than 46 images.

In order to successfully train a model to recognize landmarks, ideally we need to have access to tens of thousands of images for each category, but obviously, that is not the case for this dataset. For this project, I will explore how well MobileNets could perform with limited training data to classify images in the landmark dataset.

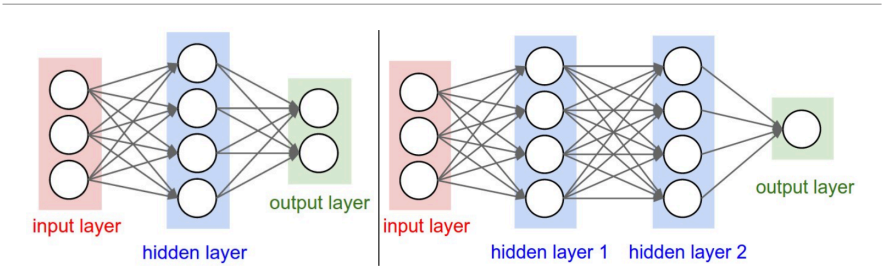
II. Prior Work

Computer vision is an interdisciplinary field that aims to give machines the capability to analyze, understand and extract useful information from digital images or videos. Tasks such as image recognition, image classification, object detection, etc. fall under the broad category of computer vision. Neural Networks are a subfield of computer vision and are vaguely inspired by the biological neural networks that constitute animal brains. They are ideal for image recognition as they can find complex relationships between features and the label.

Dr. Robert Hecht-Nielsen, the inventor of one of the first neurocomputers, defined a neural network as:

"a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

A neural network is a highly-structured graph, organized into one or more hidden layers. Each hidden layer consists of one or more neurons. Features are fed to the network via the input layer, which communicates to one or more 'hidden layers' where the actual processing is done through a system of weighted connections. The hidden layers then link to an output layer where the output is predicted. The image below shows two fully connected neural networks (which means that the neurons in one layer take inputs from every neuron in the previous layer) with one or more hidden layers.



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs.
Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Figure 4 A Basic Neural Network, Image Source http://cs231n.github.io/neural-networks-1/#nn [22]

A. A Brief History of Artificial Neural Networks

The idea of creating intelligent machines started in the 1940's with Alan Turing in his seminal paper entitled 'Computing Machinery and Intelligence,' where he proposed several measures to evaluate a machine's intelligence, which came to be known as the "Turing Test." Earlier works in machine learning were largely based on mimicking the way human brains are thought to work. In 1943, Walter Pitts and Warren McCulloch wrote a paper titled "A Logical Calculus of Ideas Immanent in Nervous Activity" [26]. Their model was vastly oversimplified for a biological brain, but it succeeded at showing a proof of principle.

The first real precursor to modern neural networks was the perceptron, an electronic device which was constructed in accordance with biological principles and showed an ability to learn. It was developed by Frank Rosenblatt in 1957 and is built around a nonlinear neuron, namely, the McCulloch–Pitts model of a neuron [14]. The success of Rosenblatt's perceptron was noticed by Marvin Minsky, who, along with Seymour Papert, wrote a book entitled "Perceptrons - An Introduction to Computational Geometry," in which they proved that the perceptron is incapable of learning a simple XOR function, thus killing the future of perceptron and ushering in an era known as the first 'AI winter' [18].

The event that renewed the interest in neural networks and learning was in 1974 when Paul Werbos proposed a backpropagation algorithm that effectively solved the exclusive-or problem, and, more generally, accelerated the training of multi-layer networks [27]. Backpropagation distributed the error term back up through the layers by modifying the weights at each node.

The next major development in the study of neural networks came when Geoffrey Hinton along with David Rumelhart and Ronald Williams published a paper entitled "Learning representations by back-propagating errors" [3]. In this paper, they showed that neural networks with many hidden layers could be effectively trained by a relatively simple procedure. This would allow neural networks to get around the weakness of the perceptron because the additional layers endowed the network with the ability to learn non-linear functions.

The following image shows the timeline of the history of neural networks.

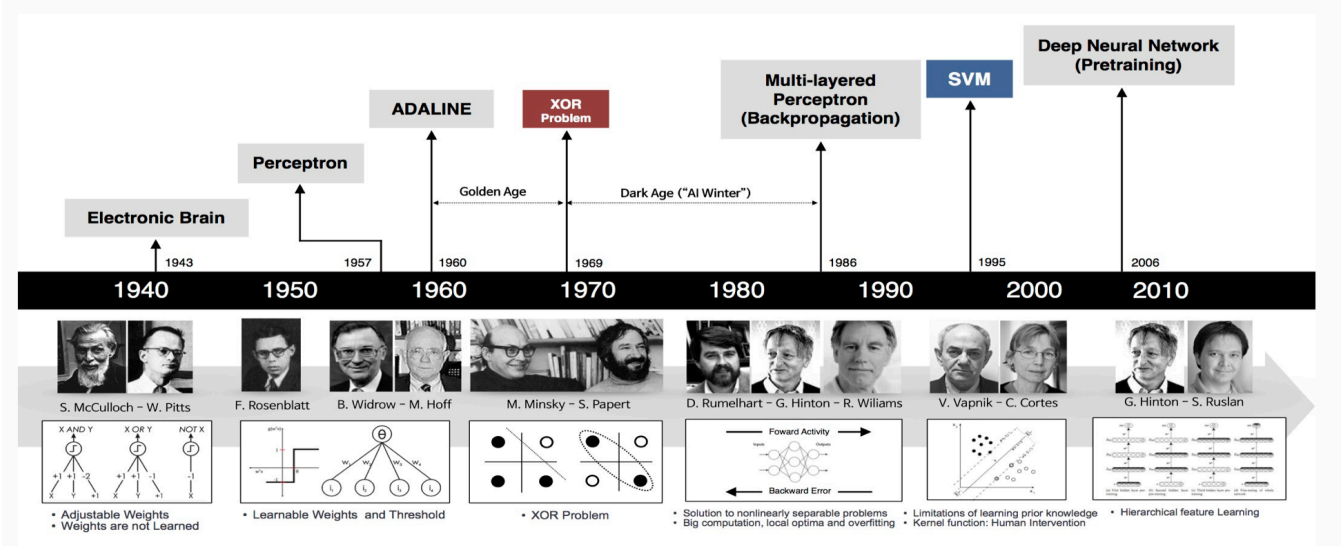


Figure 5 Major Milestones in the History of Neural Networks.
Image Source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html[23]

B. Convolutional Neural Networks

In the machine learning context, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks. The term convolution just means that the same calculations are performed at each location in the image. The main difference between a CNN and a conventional Neural Network (ANN) is that in an ANN, each neuron in the input layer is connected to all the neurons in the hidden layer whereas in a CNN, each neuron is connected to only a local region of the input volume.

1) Brief history of CNN

The idea of a convolutional neural network was first proposed by Fukushima in his paper "Neocognitron" published in 1980 [19]. The Neocognitron was inspired by the discoveries of Hubel and Wiesel in 1964 about the visual cortex of mammals. Their study concluded that vision impairments that occurred at birth affected the vision later in life, as the cells that were responsible for processing visual information redistributed to favor the unimpaired eye. Fukushima proposed various supervised and unsupervised learning algorithms (except for backpropagation, the core algorithm behind how neural networks learn) to train Neocognitron and implements all fundamental ideas behind convolutional neural networks. Fukushima applied the Neocognitron to recognize hand-written digits.

A convolutional-like network trained by backpropagation was proposed in 1990 by Kevin J. Lang et al, in their paper on the study of network architectures for performing spoken letter recognition [20]. It can be considered a convolutional network without pooling.

However, successfully applying backpropagation learning to a large, real-world application is attributed to Yann LeCun in his paper on recognizing handwritten digits using backpropagation [21]. The main point of this paper is to show that large backpropagation networks can be applied to real image-recognition problems without a large, complex preprocessing stage requiring detailed engineering. Unlike most previous work on the subject, the learning network is directly fed with images, rather than feature vectors, thus demonstrating the ability of backpropagation networks to deal with large amounts of low level information.

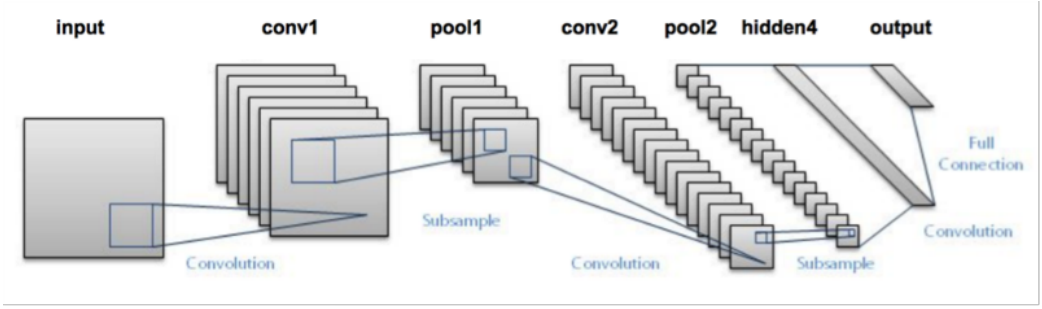


Figure 6 The CNN Model Proposed by LeCun et al in 1998 for handwritten digit recognition.

The popularity of CNNs increased in 2012 when a group of engineers from Google won the Large Scale Visual Recognition Challenge (LSVRC) using a CNN model labeled 'AlexNet' [4]. This paved the way for CNNs to become the mainstay algorithm of deep learning and one of the best models available today to perform image recognition and classification. CNNs, which are explained in more detail later in this section, can perform remarkably well even on small datasets without the need for any custom engineering. Additionally, Google's Inception [5] has accomplished significant advancements in the field of computer vision in the past few years, despite taking time and resources to perform well.

2) CNN Architecutre

A simple CNN can be described as a sequence of layers, with every layer of the CNN transforming one volume of activations to another through a differentiable function.

The three main layers of a CNN are:

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

These layers are stacked to form a full CNN architecture.

The layers are explained in detail below.

a) Convolutional Layer

The convolutional layer is the core component of a CNN and performs most of the complex computations. This layer performs the feature extraction, such as detecting the edges from an input image, for example. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data known as filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have a size of 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, to represent the RGB color channels). During the forward pass, the filter is 'convolved' across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. This action produces a two-dimensional 'feature map' that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. This process is repeated across the entire input and the feature maps are stacked along the depth dimension to produce the output volume. An additional operation called ReLU (which stands for Rectified Linear Unit) is performed after every convolutional operation to introduce non-linearity.

b) Pooling Layer

The function of the pooling layer (also known as subsampling or downsampling) is to progressively reduce the dimensionality of each feature map to reduce the amount of parameters and computation in the network, and thus to also control overfitting. The most common approach used in pooling is max pooling.

c) Fully Connected Layer

The input to this layer is the output of the layer that is preceding it, whether it is a convolutional layer, ReLU, or a pooling layer. It outputs an N dimensional vector where N is the number of classes the model could choose from. Basically, a FC layer looks at what high level features most strongly correlate to a specific class and has specific weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.

The image below shows the different layers of CNN in action. The initial volume stores the raw image pixels and the last volume stores the class scores. Each volume of activations along the processing path is shown as a column. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one.

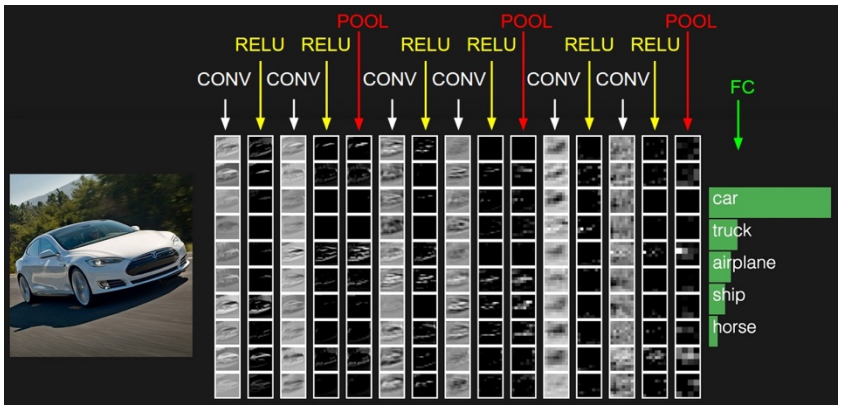


Figure 7 The Activations of a Sample CNN [22]

III. PROBLEM/PROJECT DESCRIPTION

Convolutional Neural Networks are the ideal model for recognizing landmarks in an image. However, a large obstacle was the availability of a large labeled dataset. Kaggle teamed up with Google to provide the largest annotated dataset to date for the Landmark Recognition Challenge. The dataset consists of 14,951 landmarks and a total of 1,225,029 images and the challenge is to identify if there's a landmark in the image and to correctly recognize it.

Training a dataset of this magnitude from scratch would take days, if not weeks, and would require significant computational power. As I do not have access to graphics processing units (GPU) and rely on CPU to perform image recognition and time is of the essence, using a pre-trained model was the most practical solution. The machine learning term for repurposing a model that's trained to do one task to perform another task is 'transfer learning'.

For landmark recognition, I'm reusing the feature extraction capabilities from a powerful image classifier known as MobileNets[5]. MobileNets is a family of mobile-first computer vision models for TensorFlow. A brief explanation of the history of TensorFlow and MobileNets below.

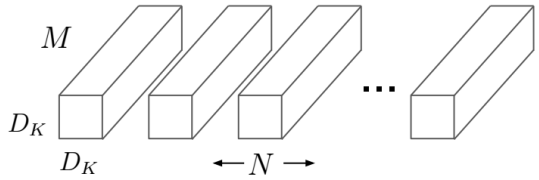
A. TensorFlow

TensorFlow is an open source library for numerical computations developed by the Google Brain Team [17]. It is developed primarily for Deep Neural Networks research and Machine Learning and became open source in November, 2015. TensorFlow uses data-flow graphs to perform numerical computations. The advantages of using a graph based model are that it's portable, transformable, and optimizable, and it supports distributed execution. TensorFlow's high-level APIs, in conjunction with computation graphs, enable a rich and flexible development environment and powerful production capabilities in the same framework.

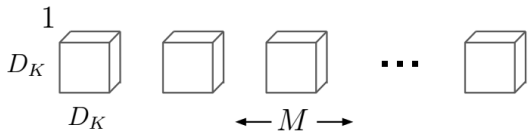
B. MobileNets

MobileNets was developed by Andrew G. Howard and Menglong Zhu, along with a group of researchers from Google. It uses depth-wise separable convolutions to build light weight deep neural networks. MobileNets is small, low-latency, low-power models that does not require a lot of computational power. The only drawback is that they are not as accurate as their bigger more resource intensive architecture such as Inception. They can be used for classification, detection, embedding, and segmentation, just like Inception. The main difference between the MobileNet architecture and a conventional CNN architecture is that instead of a single

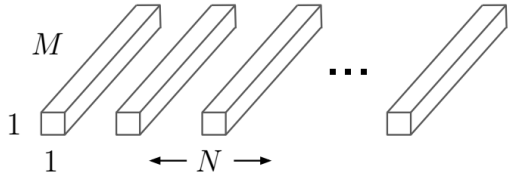
3x3 convolution layer followed by batch normalization and ReLU, MobileNets split the convolution into a 3x3 depthwise convolution and a 1x1 pointwise convolution as illustrated in the image below.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 8 MobileNets Architecture [5]

MobileNets is trained on a database called ImageNet that contains over 14 million URLs to images that have been hand-annotated to indicate what objects are pictured [12]. The basic principle behind transfer learning is that lower layers that have been trained to distinguish between some objects can be reused for many recognition tasks without any alteration. This can be achieved by deriving information about your input data from the penultimate layer of a trained model (known as bottlenecks in TensorFlow terminology), which encodes useful abstractions and then uses that as input to train your own much smaller neural net to predict your own classes. Because of the power of the learned abstractions, the additional training typically does not require large data sets.

IV. Details of the Methodology

Since I'm using transfer learning to retrain an existing architecture to recognize landmarks, the first step is to choose the right model. Google has provided two powerful models to choose from - Inception and MobileNets. Both models are trained on the ImageNet dataset and can differentiate between 1000 different classes, from a dishwasher to a Dalmatian. The kinds of information that make it possible for these models to differentiate among 1,000 can be used to recognize other objects such as landmarks by retraining the model. I decided to use MobileNets because I want the model to be small enough to run on a mobile device rather than a powerful computer, yet accurate enough to get good results, despite it not being as accurate as Inception.

Before training the entire dataset that contains over 1.2 million images with almost 15,000 labels, I decided to test the performance of MobileNets on two landmarks that contain the most number of images, St. Peter's Basilica and the Colosseum, each

of which had over 50,000 images in its dataset. The training took about three hours on a CPU and the training steps were repeated 500 times.

There were 200 images in the testing dataset, 100 images per class, and the final accuracy was an impressive 97%. The performance evaluation matrices are given below:

```
Precision [0.97959184 0.96078431]
Recall [0.96 0.98]
f1_score [0.96969697 0.97029703]
Confusion Matrix
[[96 4]
[2 98]]
```

Out of the 100 testing images under the label 6051 (for the Colosseum), 96 were correctly classified. Similarly, out of the 100 images under the label 9633 (for St. Peter's Basilica), 98 were correctly identified.

Training the entire dataset with over 1.2 million images would have taken significant amount of CPU time, so I decided to choose 1000 labels and retrain MobileNets to recognize landmarks. The smaller dataset contains over 150,000 images under 1000 labels. The results are explained in detail in a later section.

V. Implementation Description

The dataset provided by Kaggle contains URLs to the images, so the first step is to download the images. Downloading the images at their original size would take up significant amount of disk space, so Kaggle users have provided scripts [13] to download compressed version of the images to your hard drive.

Once the images are downloaded, they should be organized into labeled folders to be used with MobileNets. A Python script is used to open the CSV file containing the link to the images and organize them into folders based on the column 'landmark_id'. For example, all the images under the label '9633' will be stored in folder called 9633.

Now, onto training the model using MobileNets version 1.0. The script to retrain the model is given below. This script loads the pre-trained module and trains a new classifier on top for the landmark images that I have downloaded. Though the MobileNets model was not trained to recognize landmarks, with the magic of transfer learning, lower layers that have been trained to distinguish between some objects can be reused for many recognition tasks without any alteration.

```
IMAGE_SIZE=224
ARCHITECTURE="mobilenet_1.0_${IMAGE_SIZE}"
python -m scripts.retrain \
  --bottleneck_dir=images/bottlenecks \
  --how_many_training_steps=1000 \
  --model_dir=images/models/ \
  --summaries_dir=images/training_summaries/"${ARCHITECTURE}" \
  --output_graph=images/retrained_graph.pb \
  --output_labels=images/retrained_labels.txt \
  --architecture="${ARCHITECTURE}" \
  --image_dir=images/train
```

The arguments in the script are explained below.

- The IMAGE_SIZE variable refers to the input image resolution. The possible values are 128,160,192, or 224px. Feeding in a higher resolution image takes more processing time, but results in better classification accuracy as expected.

- bottleneck-dir - In TensorFlow, a bottleneck refers to the layer just below that final output layer that perform the classification. TensorFlow Hub calls this an "image feature vector". This penultimate layer has been trained to output a set of values that's good enough for the classifier to use to distinguish between all the classes it's been asked to recognize. That means it must be a meaningful and compact summary of the images, since it has to contain enough information for the classifier to make a good choice in a very small set of values. Calculating each bottleneck takes a substantial amount of time as every image is reused multiple times during training and calculating. These bottlenecks are cached in a /bottlenecks directory so when you run the retrain script again, these bottlenecks will be reused.

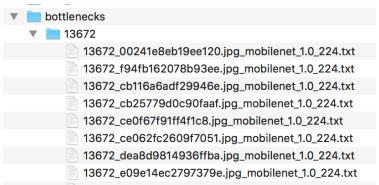


Figure 9 Sample Bottleneck Directory

- how_many_training_steps – The default value of 4000 iterations would yield higher accuracy, but it will also take longer to train. I chose the value 1000.
- model_dir – The directory that contains model, MobileNets_1.0 in this case.
- summaries_dir - Option that controls the name in TensorBoard.
- output_graph – The directory to store the graph file that will be used for classification.
- output_labels – Labels directory for classification purposes.
- architecture - the relative size of the model as a fraction of the largest MobileNet: 1.0, 0.75, 0.50, or 0.25. I chose the current version, 1.0.
- image_dir – the directory that contains the training data.

Once the bottlenecks are complete, the actual training of the top layer of the network begins. A series of step outputs are displayed, each one showing training accuracy, validation accuracy, and the cross entropy.

- The training accuracy shows what percent of the images used in the current training batch were labeled with the correct class.
- The validation accuracy is the precision on a randomly-selected group of images from a different set. The key difference is that the training accuracy is based on images that the network has been able to learn from so the network can over-fit to the noise in the training data. A true measure of the performance of the network is to measure its performance on a data set not contained in the training data--this is measured by the validation accuracy. If the training accuracy is high but the validation accuracy remains low, that means the network is overfitting and memorizing specific features in the training images that aren't helpful more generally.
- Cross entropy is a loss function which gives a glimpse into how well the learning process is progressing. The training's objective is to make the loss as small as possible, so if the loss shows a downward trend, that indicates that the learning is working.

The script ran for 1,000 training steps. Each step chooses ten images at random from the training set, finds their bottlenecks from the cache, and feeds them into the final layer to get predictions. Those predictions are then compared against the actual labels to update the final layer's weights through the back-propagation process. As the process continues, the reported accuracy should improve, and after all the steps are done, a final test accuracy evaluation is run on a set of images kept separate from the training

and validation pictures. This test evaluation is the best estimate of how the trained model will perform on the classification task. The landmark model shows a training accuracy of 89% and a validation accuracy that ranges from 80% to 66% in its final steps. This number is based on the percent of the images in the test set that are given the correct label after the model is fully trained.

```
INFO:tensorflow:2018-05-01 13:33:12.355840: Step 950: Train accuracy = 89.0%
INFO:tensorflow:2018-05-01 13:33:12.355994: Step 950: Cross entropy = 0.660593
INFO:tensorflow:2018-05-01 13:33:12.408165: Step 950: Validation accuracy = 80.0% (N=100)
INFO:tensorflow:2018-05-01 13:33:13.359937: Step 960: Train accuracy = 88.0%
INFO:tensorflow:2018-05-01 13:33:13.360099: Step 960: Cross entropy = 0.788265
INFO:tensorflow:2018-05-01 13:33:13.414909: Step 960: Validation accuracy = 66.0% (N=100)
INFO:tensorflow:2018-05-01 13:33:14.365969: Step 970: Train accuracy = 90.0%
INFO:tensorflow:2018-05-01 13:33:14.366134: Step 970: Cross entropy = 0.639148
INFO:tensorflow:2018-05-01 13:33:14.460967: Step 970: Validation accuracy = 73.0% (N=100)
INFO:tensorflow:2018-05-01 13:33:15.440402: Step 980: Train accuracy = 89.0%
INFO:tensorflow:2018-05-01 13:33:15.440559: Step 980: Cross entropy = 0.677505
INFO:tensorflow:2018-05-01 13:33:15.495935: Step 980: Validation accuracy = 73.0% (N=100)
INFO:tensorflow:2018-05-01 13:33:16.486468: Step 990: Train accuracy = 86.0%
INFO:tensorflow:2018-05-01 13:33:16.486632: Step 990: Cross entropy = 0.841019
INFO:tensorflow:2018-05-01 13:33:16.547500: Step 990: Validation accuracy = 66.0% (N=100)
INFO:tensorflow:2018-05-01 13:33:17.362456: Step 999: Train accuracy = 89.0%
INFO:tensorflow:2018-05-01 13:33:17.362598: Step 999: Cross entropy = 0.598122
INFO:tensorflow:2018-05-01 13:33:17.418497: Step 999: Validation accuracy = 68.0% (N=100)
INFO:tensorflow:Final test accuracy = 69.9% (N=28145)
```

VI. Results

The following script run the Python file label_images.py to predict the labels of testing data. During training, TensorFlow creates a graph that contains the model and stores it in a file. This graph will be used to predict the labels of the test class.

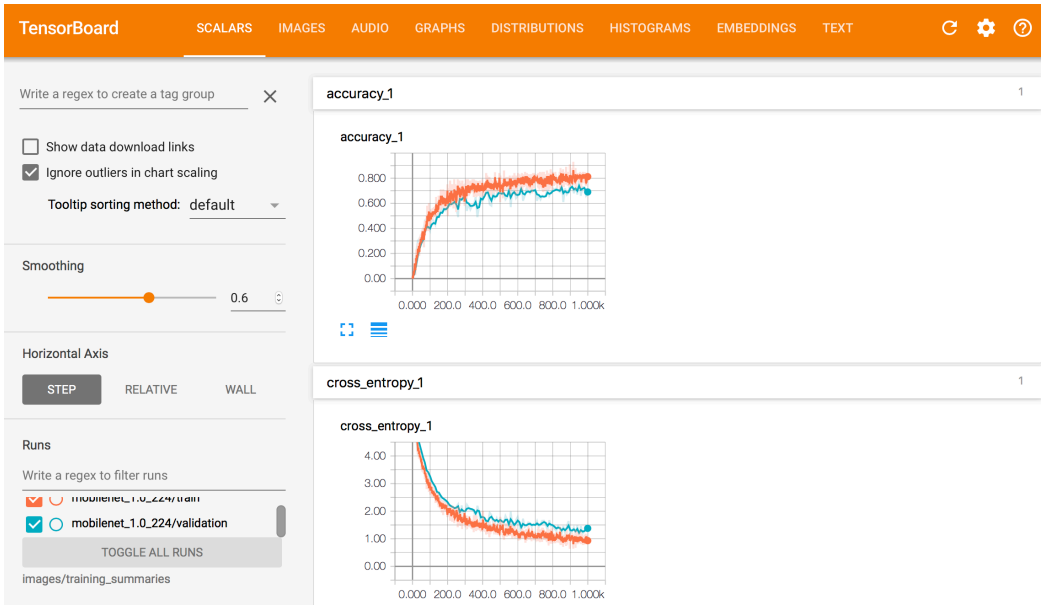
```
python -m scripts.label_images \
    --graph=images/retrained_graph.pb  \
    --image=/images/test
```

The table below shows the classification results of 30 images of label ‘9633’. Out of 30 images, 21 images were correctly classified as ‘9633’, resulting in an accuracy of 70%.

	Image_id	Prediction	Confidence
1	000a3c7c32743ef1.jpg	13489	0.23662952
2	00a1a7251d02f5ef.jpg	11371	0.21159577
3	00a1a8b9419c90d0.jpg	9633	0.8956684
4	0a00d750adf76ed3.jpg	9633	0.31832758
5	0a0a02f59903a130.jpg	13568	0.20402959
6	0a0c49bb9ca248af.jpg	9633	0.45058367
7	0a0d55394ab11c4d.jpg	9633	0.60614485
8	0a0d7504e7a7c90a.jpg	9633	0.5555334
9	0a0db547fc836761.jpg	9633	0.46376842
10	0a0ee61e7c1f248c.jpg	9633	0.17322397
11	0a1a23ff26440ec4.jpg	9633	0.1214397
12	0a1adf722d7abeaf.jpg	9633	0.34875196
13	0a1afb164a753ac7.jpg	9633	0.44159034
14	0a1ddbb53ea6e2ba.jpg	12370	0.12759455
15	0a1e0ec30108632e.jpg	9633	0.38657582
16	0a1e6cb166e5840e.jpg	9633	0.4894646
17	0a1ed37e5c2c3fef.jpg	10685	0.13708065
18	0a1ed65ef9f9f4db.jpg	11038	0.21042281
19	0a1ee0062f7f3c56.jpg	11781	0.07618652
20	0a2b38f48d3eb4f3.jpg	9633	0.58663696

21	0a3b01892e402390.jpg	9633	0.2916586
22	0a3bf3ca83b2b5c2.jpg	13653	0.17970371
23	0a3c0ed11e201a4b.jpg	9633	0.2040012
24	0a3eb2c5e666c27e.jpg	9633	0.53955156
25	0a3f8cb4b8a76e0a.jpg	9633	0.45683643
26	0a4a8a83a9dfc836.jpg	9633	0.11119142
27	0a4d80542f103bdf.jpg	9633	0.42200446
28	0a5a2f8476f74f57.jpg	9633	0.5686393
29	0a5afe0ad8bb4ad1.jpg	13471	0.37268758
30	015d3e5fa1dc316c.jpg	9633	0.34549206

TensorFlow comes with a suite of visualization tools called TensorBoard to help understand, debug and optimize TensorFlow programs. TensorBoard can be used to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it. When TensorBoard is fully configured, it looks like the image below.



The image shows how the accuracy and cross-entropy change over each traning step. It's helpful to have a tool to visualize the traning process, so we can see by looking at the graph the model is learning because the cross-entropy gradually decreases during each step of the traning phase. (1000 steps in this example).

The TensorFlow graphs are really complex, but the tool lets you dive deep into each component of the graph plot quantitative metrics about the execution of your graph, and show additional data like the images that pass through it. The following image shows the graph generate by TensorFlow that shows all the layers in the MobileNet architecture and how they are connected to oneanother.

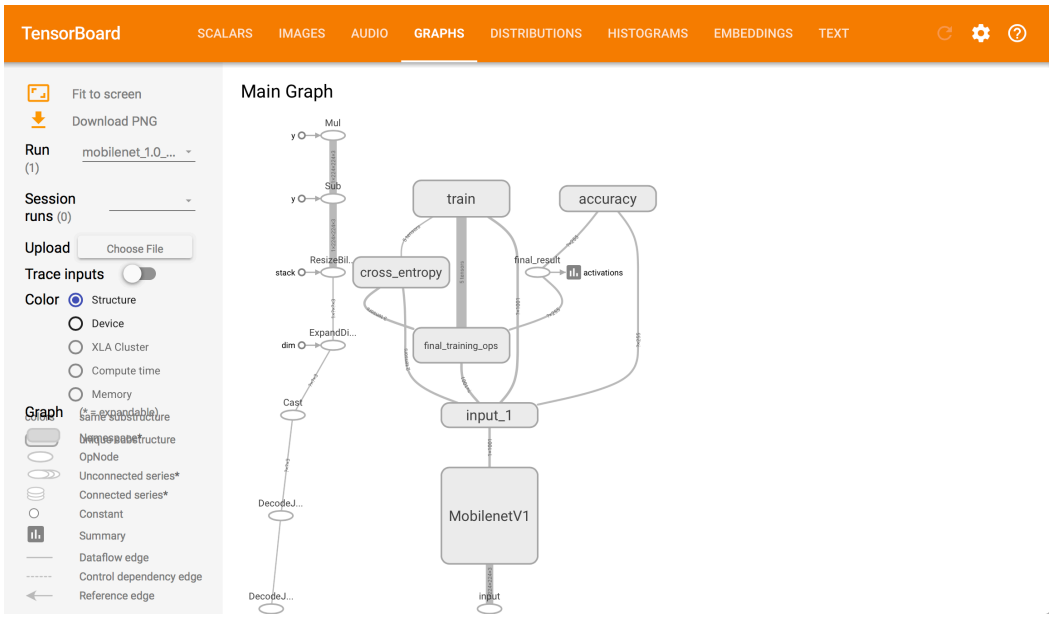


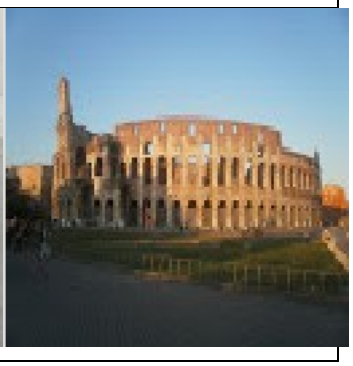
Figure 10 TensorFlow Graph

VII. Discussion of the Results

The MobileNets model performed extremely well with an accuracy of over 97% on the smaller dataset with 100,000 images and two labels. Due to limited disk space, I had to download the low resolution, compressed images which are about 10 KB in size. The results would have been much better if the images were of their original size.

The accuracy was significantly lower when I retrained the model to recognize 1000 landmarks. The training took 7 hours and the model had an accuracy of 70%.

The table below shows the images from five folders that were either correctly or incorrectly classified.

Correctly classified	**Incorrectly classified**
 	 

The images in each row belong to the same label, but the two pictures on the left are correctly identified as its corresponding label whereas the two images on the right as falsely identified as to belonging to a different category. The possible reasons for poor performance are:

- The images are of low resolution.
- There wasn't adequate training data.

- The testing images were obscured by another object such as a human or a tree or the image was taken from a different angle that it wasn't trained on.

VIII. Conclusion

MobileNets is small yet powerful tool for image classification yielding very good accuracy. They are architected from the ground up to be resource-friendly and run quickly and efficiently on a mobile device. A few years ago, training a dataset with more than 150,000 images under 1000 different labels would not have been possible on a laptop with just a CPU. Even with powerful computers, it would have taken a significant amount of time to train such a large dataset. But with the power of transfer learning and the availability of open-source models that are trained using powerful GPUs, I was able to train a classifier to recognize landmarks.

References

[1] H. Noh, A. Araujo, et al, "Large-Scale Image Retrieval with Attentive Deep Local Features" , 19 Dec 2016

[2] "Google Landmark Recognition Challenge", [Online] Available: https://www.kaggle.com/c/landmark-recognition-challenge/data

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation",11 Nov 2013

[4] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Dec 3, 2012

[5] C. Szegedy, V. Vanhoucke et al, "Rethinking the Inception Architecture for Computer Vision", 11 Dec 2015

[6] A.G. Howard, M. Zhu et al," MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 17 Apr 2017

[7] Mart´ın Abadi, Ashish Agarwal et al, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", 9 Nov 2015

[8] F. Chollet, "Building powerful image classification models using very little data", [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html [Accessed: 23- March- 2018].

[9] "How to Retrain an Image Classifier for New Categories", [Online]. Available: https://www.tensorflow.org/tutorials/image_retraining[Accessed: 25- March- 2018].

[10] A. Howard and M. Zhu, "MobileNets: Open-Source Models for Efficient On-Device Vision", [Online]. Available: https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html [Accessed: 01- April- 2018]

[11] "TensorFlow for Poets", [Online]. Available: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets, [Accessed: 28- March- 2018].

[12] "Image Net", [Online]. Available: http://www.image-net.org/, [Accessed: 20- March- 2018].

[13] "Fast Resized Image Download (Python 3)", Available: https://www.kaggle.com/lyakaap/fast-resized-image-download-python-3

[14] "ImageNets", https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

[15] "Rosenblatt's Perceptron "[Online] Available: https://www.pearsonhighered.com/assets/samplechapter/0/1/3/1/0131471392.pdf

[16] "TensorFlow-Slim image classification model library", [Online] Available: https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models.

[17] "Image Recognition", [Online] Available: https://www.tensorflow.org/tutorials/image_recognition

[18] Marvin Minsky, Seymour A. Papert, "Perceptrons - An Introduction to Computational Geometry", January 1969

[19] Kunihiko Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", 1980

[20] Kevin J. Lang, Alex Waibel, Geoffrey E. Hinton, "A Time-Delay Neural Network Architecture for Isolated Word Recognition", 1990

[21] Y. LeCun et al, "Handwritten Digit Recognition with a Back-Propagation Network", 1990

[22] "Convolutional Neural Networks for Visual Recognition", [Online] Available: http://cs231n.github.io/neural-networks-1/

[23] Andrew L. Beam, "Deep Learning 101 - Part 1: History and Background", [Online], Available: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

[24] Jon Shlen, "Train your own image classifier with Inception in TensorFlow" [Online] Available: https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

[25] Christian Szegedy, Wei Liu et al, "Going Deeper with Convolutions", Sep 2014

[26] W.S. McCulloch and W.Pitts, , "A Logical Calculus of Ideas Immanent in Nervous Activity", 1943

[27] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", 1974