# Project Report : Geo-Location Estimation using Convolutional Neural Networks

M.Pranith Reddy(IMT2015025)
M.Karthikeya Reddy(IMT2015026)
G.Sravya(IMT2015014)

December 2018

### Problem Statement

Given an image, predicting the location where the image was taken is the main goal of this project.

## Introduction

Predicting Geolocation of an image is an extremely challenging task since many images offer only a few cues about their location and these cues can often be ambiguous. In the context of computer vision or recognition there have been many traditional models developed for Geo-recognition task. The approaches were focused on various levels of geo-location estimation such as a variant of the Nearest neighbour algorithm using generalized Minimum Clique Graphs proposed by Zamir and Shah[1] for geo-localisation.

But there have been less experiments involving deep learning models such as CNN's for this recognition task. Hence this project employs a methodology that explores CNN architectures to classify images according to the geospatial attributes. Images often contain information such as landmarks, weather patterns, roads, and architectural details, which we hope that our CNN architecture learns and able to determine location. For this project, we tried exper-

imenting with two different approaches, coarse and fine grained classification. Coarse classification is where given an image, we tried to predict its city and fine grained is where we try to predict the exact sub-region in the city.

## Dataset Description

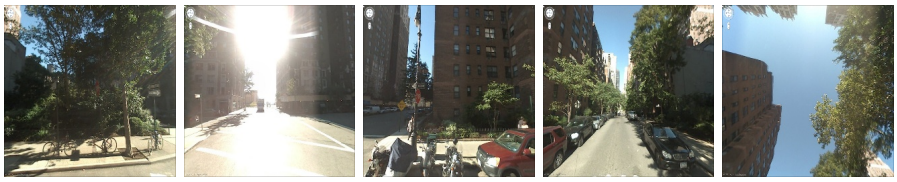
We tried our models on 4 cities- Sydney, London, Paris and Moscow. To obtain the street view images of these cities, we used Google street view API. We created a API from google maps and wrote a script to download the images from google street view maps.

URL = ("http://maps.googleapis.com/maps/api/streetview?sensor=falsesize =256x256fov=120key=" + API_KEY + "location=lat,lon)

Given a Longitude and Latitude, above url path returns an image with size 256 x 256. By using this, we created a random points of latitude and longitude in a city to obtain 3000 images per city as the API supports only 25,000 requests. Each image label will consist of its coordinates such as latitude and longitude.


**Figure 1:** Sample images from Google Street View Images Dataset

For fine grained classification, we used dataset of Google street view images obtained by Zamir and Shah[2]. The dataset itself contains a total of over 62,238 RGB images taken from 3 cities: Pittsburgh, New York City, and Orlando. For each city, a set of images is captured at placemarks located roughly 12m apart, allowing for more fine recognition than would be possible for datasets of less uniformly or less densely scattered data. Each placemark contains a total of 5 images, representing four panoramic side views and one upward view. Within this dataset, there are approximately 5500 placemarks from New York, 3000 from Pittsburgh, and 1500 from Orlando. Each image is labeled with GPS and compass coordinates, and all were taken in outdoor environments under similar physical conditions. So for our experimentation we used New York city data, which consists 25,818 images with size 1280 x 1025 pixels.



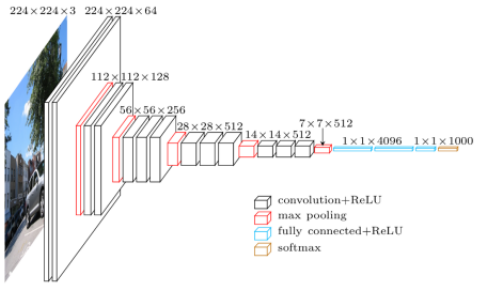
Sample input images for Fine-Grained classification

# Technical Approach

As we have divided the problem statement into two sub-problems, we present our approach to both classifications.

## Coarse Classification

- Given a test image, it determines the corresponding city in which the image was captured.
- As discussed in the dataset description, we obtained 3000 images randomly from each city and divided the training and testing into 10114, 1785 images respectively. All the images were split into train and test by stratification.
- We trained these samples on different CNN architectures such as GoogleNet (InceptionNet_v3), VGGNet(16), ResNet(50).
- **VGGNet** : This architecture has 13 convolution layers, 5 pooling layers and 3 dense layers. Each convolution layer used 3x3 filters, 2 pixel padding, and stride 1. The number of filters at each level were 64,128,256,512,512 respectively. The pooling layers are max-pool with stride 2 and 2x2 areas. The activation function used in all the layers is 'relu'. The last dense layer used softmax activation function.




VGG-16 Architecture

- **GoogleNet(InceptionNet_V3)**: The architecture we used is Inception v3 which has 3 convolution layers having filters of size 3x3, stride 2,1,1 respectively, followed by a pooling layer having stride 2 and 3x3 areas. It is followed by 3 convolution layers having filters of size 3x3 and stride 1,2,1 respectively. Followed by 3 Inception layers which is a combination of 3 convolution layers of filter size 3x3, 4 convolution layers of filter size 1x1 and a pooling layer. It is followed by 5 Inception layers which is a combination of 3 convolution layers of filter size nx1, 3 convolution layers of filter size 1xn, 4 convolution layer of filter size 1x1 and a pooling layer. Followed by 2 Inception layers which is a combination of 2 convolution layers of filter size 1x3, 2 convolution layers of filter size 1x3, one convolution layer of filter size 3x3, 4 convolution layers of filter size 1x1 and a pooling layer. These layers are followed by a pooling layer of 8x8 areas, a dense layer followed by a softmax layer.

- **ResNet50:** This architecture has a convolution layer having 64 filters of size 7x7,stride 2 and relu activation function. It is followed by a max pooling layer of stride 2 and 3x3 areas, followed by a 3 convolution blocks each having 3 convolution layers of 64,64,256 number of filter of size 1x1, 3x3, 1x1 respectively. Followed by 4 convolution blocks each having 3 convolution layers of 128,128,512 number of filters of size 1x1,3x3 and 1x1 respectively. It is followed by 6 convolution blocks each having 3 convolution layers of 256,256,1024 number of filters of size 1x1,3x3 and 1x1 respectively. It is followed by 3 convolution blocks each having 3 convolution layers of 512,512,2048 number of filters of size 1x1,3x3,1x1 respectively. These layers are followed by a average pooling layer,a dense layer and finally a softmax layer.

The optimizer used in all the above architectures is SGD with a learning rate of 0.0001, momentum of 0.9 and the loss function is categorical cross entropy.

## Fine-Grained Classification

- Given an image taken from a single city, it determines the exact, corresponding sub-region within the city.

- In the case of fine grained classification with in city, we choose New York because of dense distribution of data in the city. We split the dataset in to 23236 training images and 2582 testing images.

- For defining fine location classes with in a city, we applied two strategies:
  - tiling by latitude/longitude
  - clustering the coordinates

### Tiling

In case of tiling, we create a grid whose components are of equal spatial size. To create a grid, we initially created a csv file which consists of GPS coordinates of all the images. Each row in it corresponds to an image and each column specify latitude and longitude. For intializing a grid we have used 'mercantile' library, the mercantile module provides ul(xtile, ytile, zoom) and bounds(xtile, ytile, zoom) functions that respectively return the upper left corner and bounding longitudes and latitudes. For a given value of zoom, the mercantile initialises a grid over the world but as we are interested only on New York city, to obtain the tiles we used tile(lng, lat, zoom) which returns coordinates of tile. Based up on the above information, we create labels or classes for every image and each tile(class) may consists images based up on zoom value. However as the images are not uniformly distributed across the city, tiling often yields buckets that are unbalanced in their constituent data points.


Visualization of tiling over New York City

### Clustering

In case of clustering, we project the latitude and longitude of images into x, y coordinates by using basemap library from mpltoolkits. Then by using those coordinates, we perform a kmeans clustering which yields us clusters, where each cluster consists of data points.From the above mechanism each cluster can be considered as a class label for training, splitting has done based on stratification that is number of test images from all the clusters are equal.

- We trained CNN models for different granularities, so that we can observe at which granularity the optimal classification is being achieved. Over-

all, our results describe as the number of tiles increases the accuracy of predicting the sub-region decreases. It's because of decreasing number of data points per class and increase in classes, so it makes sense in terms of decreasing accuracy. Similar results were also observed in the case of clustering mechanism too.

# Training and Testing

- All the test and train images are resized to 256x256x3.
- We trained all the model using the following hyper-parameters as below :
  - Batch Size = 32
  - epochs = 100
- The optimizer used in all the models is sgd:
  - Learning rate = 0.0001
  - Momentum = 0.9
- The average training time for coarse classification - 6 min/epoch.
- The average training time for fine grained classification - 17 min/epoch

# Results

We evaluated our results based on accuracy metric.

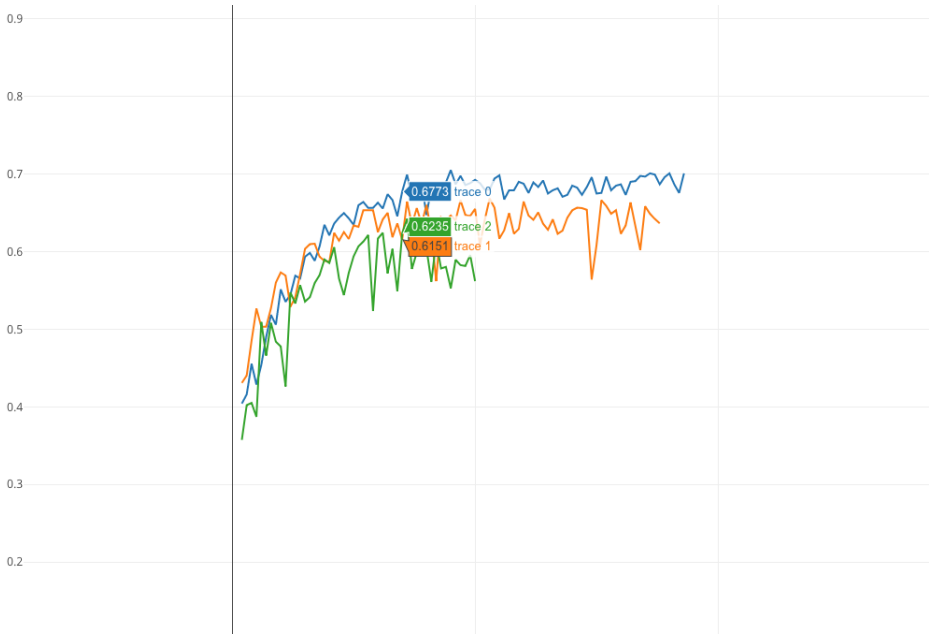
## Coarse Classification

For Coarse Classification we obtained maximum accuracy using VGGNet architecture.

| Model | Max-Accuracy |
|---|---|
| VGGNet-16 | 70.08 |
| GoogleNet | 66.61 |
| ResNet-50 | 64.2 |

**Table 1:** Accuracy Comparison between different architectures

Some images which were wrongly classified




Blue - VGGNet Orange -GoogleNet
Green - ResNet

Accuracy vs number of epochs (over 100 epochs)

As we can see from the table that VGG net performs well in classifying images. Though the validation accuracy is not great but we still think this is a good performance over all because the training accuracy in all the models was more than 97%. Also it should be noted that the images were extracted from random points in a city, if every street view image is captured in all directions and trained then the accuracy would be a lot high. We could also make models robust with including images that were taken from various times of the day and under different view points or angles. Because this way the model can learn to distinguish weather in different cities. This an important task in identifying geo location because as humans we ourselves extract contextual information from the environment in order to draw any conclusions about the location.

### Fine-Grained Classification

| Classes | 3 | 12 | 23 | 144 | 582 |
|---|---|---|---|---|---|
| Accuracy | 87 | 77.5 | 71.4 | 46 | 41 |

**Table 2:** Accuracy Comparison for different granularity

As we trained CNN models(VGG) for different granularity (number of tiles) from the table it is clear that as increase of the number of classes the ability of the CNN's to classify decreases. While comparing our validation accuracy with training accuracy the difference was not much for each granularity level and as the number of epochs increases the training accuracy increases but validation remains more or less the same. The error rate can also be due to the large similarities between adjacent tiles and the contrived boundaries enforced by tiling. Implementing a new strategy could help this by creating more finer boundaries. Also increasing the training data would also help improve performance.

## Conclusion and Future Work

In this project, we tried experimenting with different architectures and the results were good enough considering the amount of data that is used for training. Hence we can say that CNN's have the ability to classify images in both coarse and fine grained classifications. But as the test data source is also from Google street view data, hence the structure would be a lot similar, so in order to measure the generalization capability of models, it should be tested on more diverse data. If we consider granularity in cities range, we tried on 4 cities, this can be expanded to more number of cities but at the same time that would require larger dataset. Also by visualizing the first few convolutional layers, we can obtain the knowledge about what the model is able to learn such as architectural styles, landmarks, greenery of the environment. For calculating the exact location of a given image, we can recursively sub-divide the regions using CNN's. It means after classifying the image to a region by CNN model, the model can be applied again in the region to determine the sub-region. This can be done recursively until the lowest granularity level that may be a street. This kind of architecture would be sophisticated and takes more training time but we hope it would be worth experimenting.

# References

1. A.R.Zamir and M.Shah. Accurate Image Localization Based on Google Maps Street View. Proceedings of the 11th European conference on Computer vision, 2014

2. A. R. Zamir and M. Shah. Image Geo-localization based on Multiple Nearest Neighbour Feature Matching using Generalized Graphs. IEEE Transactions on Pattern Analysis and Ma- chine Intelligence (TPAMI), 2014

3. Centre for research in computer vision Google Street View Dataset [$http://crcv.ucf.edu/data/GMCP\_Geolocalization/$]

4. Geo-location Estimation with convolution neural networks by Amani and James hong, Stanford university

5. G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2007.

6. Mercentile [$https://mercantile.readthedocs.io/en/latest/$]

7. Fully convolutional geo location on street view data by Philipp Krähenbühl [$https://github.com/philkr/geoloc$]

8. PlaNet - Photo Geolocation with Convolutional Neural Networks by Tobias Weyand, Ilya Kostrikov, James Philbin [$https://arxiv.org/abs/1602.05314$]