Machine Learning Engineer Nanodegree

Capstone Project

Sep 1, 2018
Amit Kapoor

I. Definition

Project Overview

Image classification has been explored extensively over the years and it has shown significant improvement over the past few years. ImageNet classification challenge has played a significant role for this improvement where error rates have been impressive substantially every year. Landmark recognition is being in focus these days to advance the state of the art in computer vision. It can predict landmark labels directly from image pixels, to help better understand and organize photos.

[2] Crandall et al’s paper on landmark classification described the improvement achieved using extra features like keywords however deep learning models looks more promising as compared to classification using K-means. Deep learning techniques appears can handle large set of data like provided in this Kaggle challenge. Similarly, [4] Xception model showed great results on a larger image classification dataset comprising 350 million images and 17,000 classes.

The biggest hurdle in landmark recognition research is being the lack of large annotated

datasets. In this competition, Google presents the largest worldwide dataset to make progress and build models that recognize the correct landmark.

Problem Statement

Problem statement is “to build model that recognize the correct landmark”.

The Kaggle competition (https://www.kaggle.com/c/landmark-recognition-challenge) is to classify about approx. 15000 landmark categories from images. Landmark recognition has been challenging in its own way. Landmark recognition can predict landmark labels directly from image pixels, to help better understand and organize photo collections. The main challenges included in this competition are

1. Larger number of categories (15k approx.)
2. Number of training examples per class may not be very large.
3. Recognize the correct landmark (if any) in a dataset of challenging test images.
4. Calculate Global Average Precision (GAP)

Metrics

Image retrieval (or in more generic word information retrieval) has traditionally been considered as a pure ranking problem. The assumption is that the system does not know the intent of the user and therefore that it should return all relevant information (or at least a large subset of them) in descending rank order and let the user choose the relevant ones.

For instance, if a user queries a database with a dog image, the intent of user is not known as whether user is interested in retrieving all sorts of dogs or only dogs of the same breed. However, there are applications where the intent of the user may be known. For example, one can be interested in retrieving images of the same object, of the same architectural landmark. In such a case, the ability to retrieve only relevant images could be of high value to the user. Therefore, we use micro average precision which measures the retrieval accuracy.

Micro Average Precision (microAP) computes the Average Precision for all queries simultaneously. It measures both ranking performance as well as the ability to set a common threshold across different images.

Since this is a Kaggle Competition, I already have an evaluation metric defined as Global Average Precision (GAP). This metric is also known as micro Average Precision (microAP), as per [1]. It works as follows:

For each query image, prediction will be 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.

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

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

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 prediction i: it's 1 if the i-th prediction is correct, and 0 otherwise

II. Analysis

Data Exploration

The dataset for this challenge is available in following links:
https://www.kaggle.com/c/landmark-recognition-challenge/data
https://www.kaggle.com/google/google-landmarks-dataset

The dataset contains 2 csv files with required information to make predictions.

train.csv - This file contains a large number of images labeled with their associated landmarks. These images will be used for training the models. The training set images depict exactly one landmark. The 3 columns in this file, contains the train image ID, its URL and its label respectively. Number of rows and columns in this file are 1225030 and 3 respectively.

id → image ID (a hash)
url → image URL

landmark_id → image label (an integer)

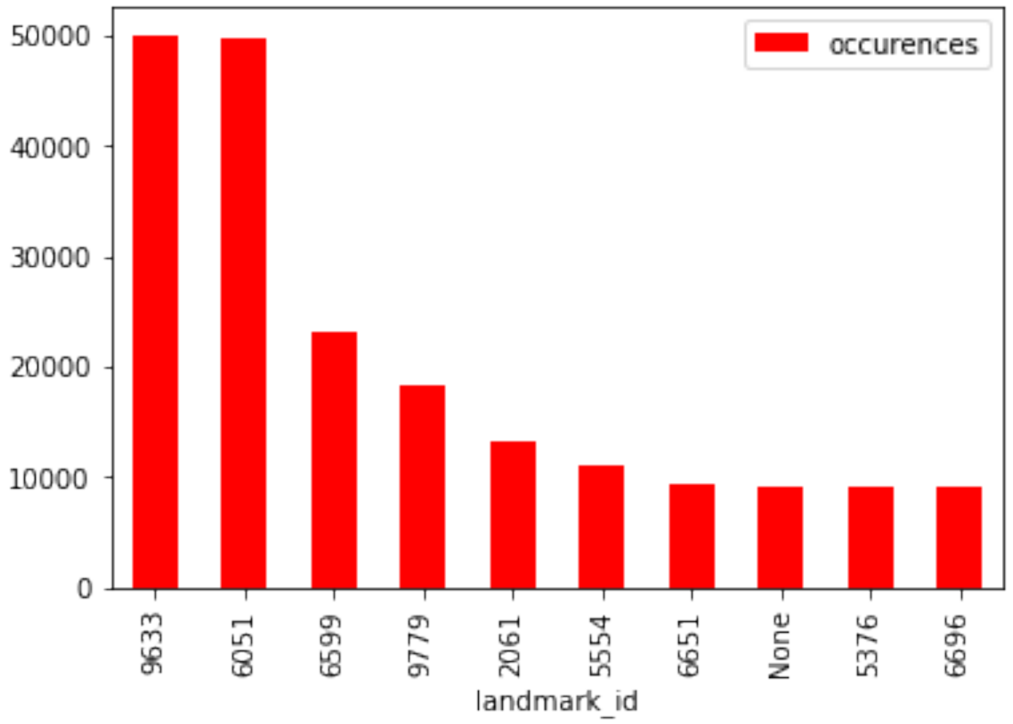
test.csv - the test set containing the test images to predict landmarks. Number of rows and columns in this file are 117704 and 2 respectively
id → image ID (a hash)
url → image URL

All images will be resized after download per the model being used for training. Size considered here is 256x256 which seems good enough to represent the images. These csv files contain the Image URLs only. All the images will be downloaded through Python script. The actual image data size is very huge so the link is being provided here. Since the dataset is huge and it requires lot of computation power, I will first start with relatively smaller dataset and test the model performance. It will then gradually increase it to larger dataset.

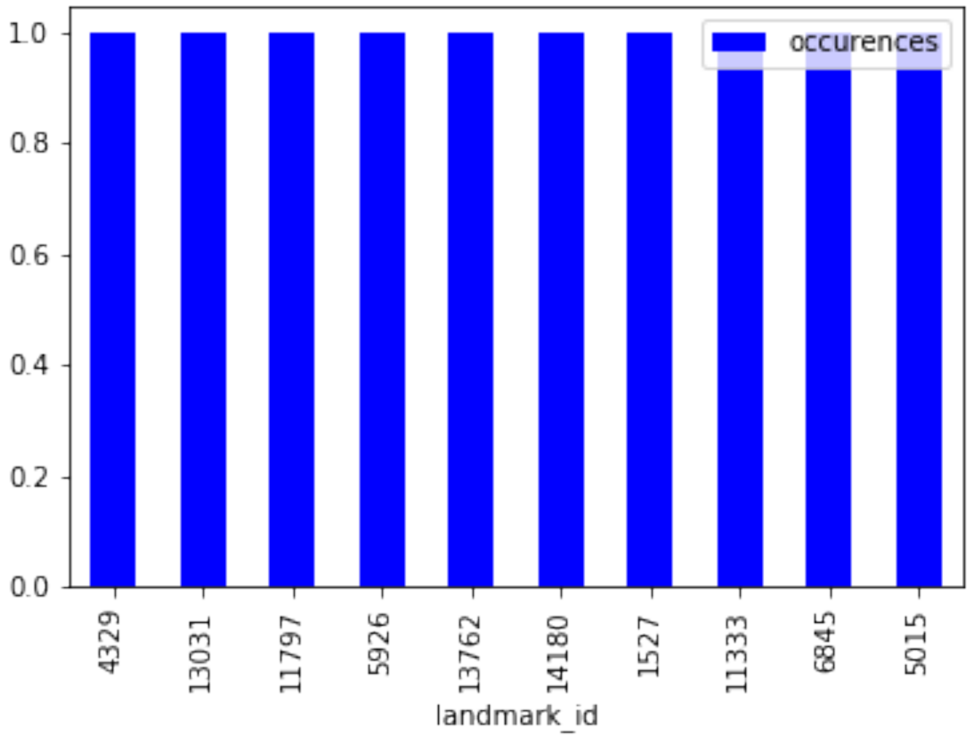
Exploratory Visualization

The challenge includes a large dataset of photos having famous landmarks and the objective was to classify an image to the correct landmark. The dataset indeed is very large containing over 14500 different classes (where each class corresponds to a landmark) and more then 1M images in entire training dataset. Some landmarks has only one or two photos in the training set and few test landmarks are not even part of training dataset. Most common occlusions include people. Training dataset is already labelled and can be used in Keras models which need properly labelled images.

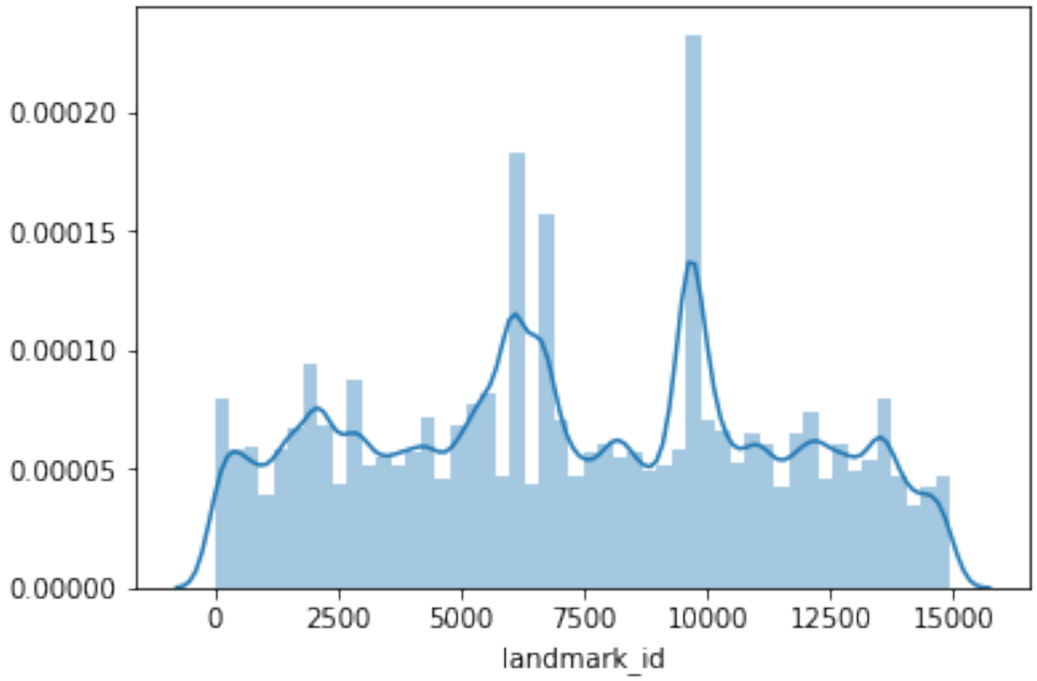
Below is the plot for 10 most frequent landmarks in the dataset. Most frequent landmark is 9633 and it appears 50010 times.

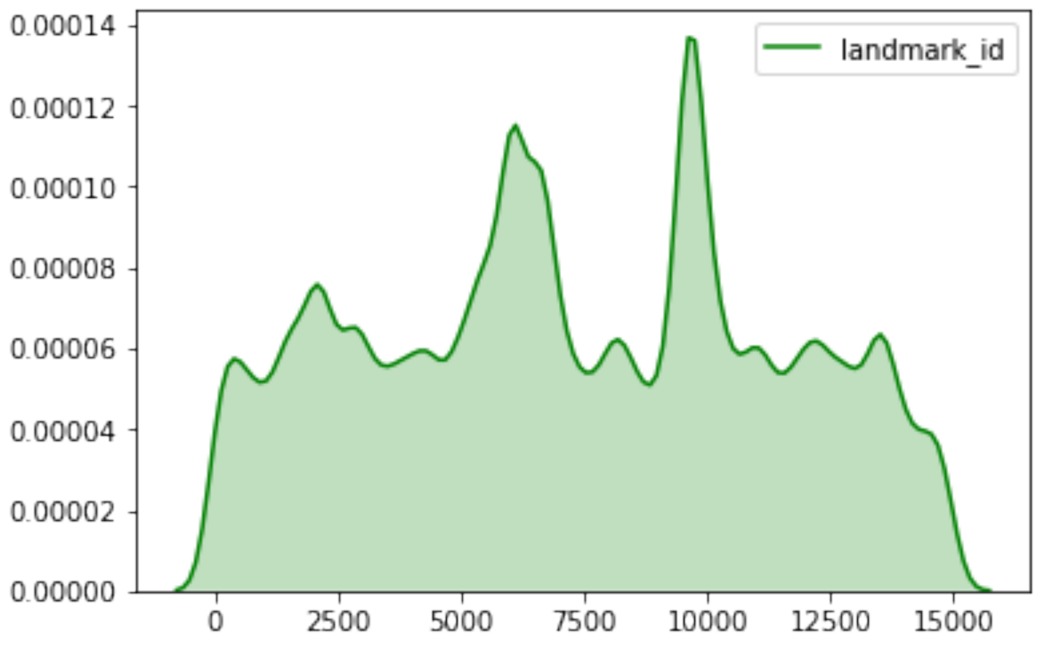


The next plot depicts the last 10 least occurred landmark ids. All of these have only 1 occurrence in the training dataset.

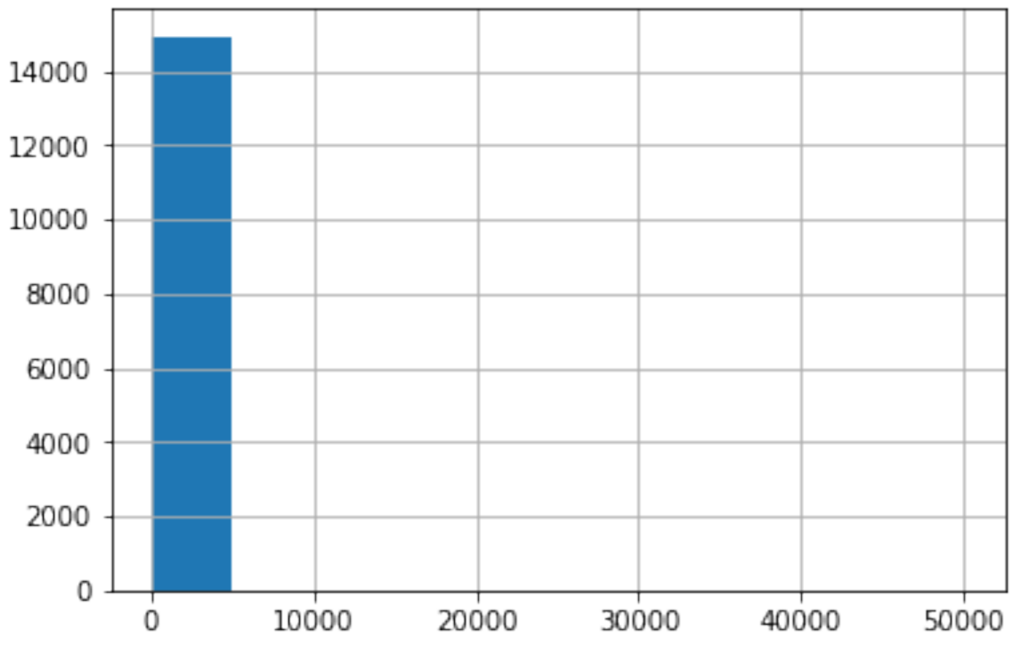


Next two following plots are distribution and density plots of the entire training dataset.





The next plot is histogram plot for unique categories of image dataset which is more than 14500.



Algorithms and Techniques

Deep learning techniques such as Deep Residual Networks, Xception and VGG models seem can handle over I million images of dataset. The success of deep convolutional networks in ImageNet classification encouraged me to use ImageNet weights that are already available in Keras. It is to initialize the weights in my trainable models.

Further I decided to use dropout as my regularization technique which drops nodes from layers with a given probability and prevents my network from overfitting.

Convolutional Neural Network is the state of the art algorithm for most image processing tasks, including classification. It requires large amount of training data compared to other approaches. This Kaggle competition does provide a very large image dataset to train the model.

Below CNN models are being considered:

- **VGG16**- It includes 16 layers and has 3x3 convolutional layers and max pooling layers combined to make deep convolutional network.

- **Xception**- The Xception model is an extension of Inception model with depthwise separable convolutions.

- **ResNet50**- ResNet50 is a 50 layers residual network which used skip connections to avoid vanishing gradients

The following parameters can be tuned for selected classifiers optimization:

- Number of epochs (training length)
- batch size (number of images to look at once)
- optimizer (rmsprop, adam etc)
- learning rate (could be dynamic as it is how fast to learn)

To perform the high computation, I have utilized AWS P2 and P3 instances which are GPU enabled and designed for deep leaning based large computations. I have utilized Keras with tensorflow as backend, keras-vis, opencv and scikit-image frameworks.

Benchmark

Benchmark model considered in this case is KNN (K nearest neighbors) approach. K nearest neighbors requires less computational overhead as compared to CNN (Convolution Neural Networks) and hence being considered here for baseline.

Along with that since this is a Kaggle challenge, the end goal will also be to have my model achieve comparative score against the ones listed in leader board of this challenge.

III. Methodology

Data Preprocessing

The preprocessing has been done using following steps:

1. First step is to setup an AWS AMI for deep learning with GPU support. I have used p2/p3 instances for this as it comes with pre-installed jupyter notebook, tensorflow and Keras.

2. I then installed other necessary packages like opencv, scikit-image and keras-vis to support various visualization and apply models.

3. Configured the GPU settings to be persistent. This command can take several minutes to run.

```
sudo nvidia-persistenced
```

4. Disabled the autoboost feature for all GPUs on the instance.

```
sudo nvidia-smi --auto-boost-default=0
```

5. Set all GPU clock speeds to their maximum frequency

6. All URLs of images are in train.csv. Below python script (image-downloader.py) was used to download all the images. I used this script to download training images with following format:

   {*class name*}**_IMG_**{*landmark id*}.jpg

   where class name and landmark id be replaced with actual values.

```
import sys, os, multiprocessing, urllib.request, csv
from PIL import Image
from io import StringIO, BytesIO

def ParseData(data_file):
  csvfile = open(data_file, 'r')
  csvreader = csv.reader(csvfile)
  #key_url_list = [line[:2] for line in csvreader]
  key_url_list = [line[:3] for line in csvreader]
  return key_url_list[1:]  # Chop off header

def DownloadImage(key_url):
  out_dir = sys.argv[2]
  (key, url, lid) = key_url
  name = "".join((key, "_img_", lid))
  #filename = os.path.join(out_dir, '%s.jpg' % key)
  filename = os.path.join(out_dir, '%s.jpg' % name)

  if os.path.exists(filename):
    print('Image %s already exists. Skipping download.' % filename)
    return

  try:
    response = urllib.request.urlopen(url)
    image_data = response.read()
  except:
    print('Warning: Could not download image %s from %s' % (key, url))
    return

  try:
    # pil_image = Image.open(StringIO(image_data))
    pil_image = Image.open(BytesIO(image_data))
  except Exception as error:
    print('Warning: Failed to parse image %s' % key)
    print(error)
    return

try:
    pil_image_rgb = pil_image.convert('RGB')
  except:
    print('Warning: Failed to convert image %s to RGB' % key)
    return

  try:
    pil_image_rgb.save(filename, format='JPEG', quality=90)
  except:
    print('Warning: Failed to save image %s' % filename)
    return
```

```
def Run():
  if len(sys.argv) != 3:
    print('Syntax: %s <data_file.csv> <output_dir/>' % sys.argv[0])
    sys.exit(0)
  (data_file, out_dir) = sys.argv[1:]

  if not os.path.exists(out_dir):
    os.mkdir(out_dir)

  key_url_list = ParseData(data_file)
  pool = multiprocessing.Pool(processes=50)
  pool.map(DownloadImage, key_url_list)


if __name__ == '__main__':
  Run()
```

7. Next, I executed below 3 shell scripts to setup train, validation and test directories. Here the images were divided into training and validation set.

xfer.sh

```
$MYDIR="/home/ubuntu/anaconda3/envs/tensorflow_p36/mlnd/images/train/"

DIRS=`ls -l $MYDIR | egrep '^d' | awk '{print $9}'`

# and now loop through the directories:
for DIR in $DIRS
do
echo  ${DIR}
mv *_${DIR}.jpg ./${DIR}
done
```

xfer_train_to_valid.sh

```
$MYDIR="/home/ubuntu/anaconda3/envs/tensorflow_p36/mlnd/images/train/"

DIRS=`ls -l $MYDIR | egrep '^d' | awk '{print $9}'`

# and now loop through the directories:
for DIR in $DIRS
do
echo  ${DIR}
cd ./${DIR}
mv `ls | head -100` /home/ubuntu/anaconda3/envs/tensorflow_p36/mlnd/images/validation/${DIR}/
cd ..
done
```

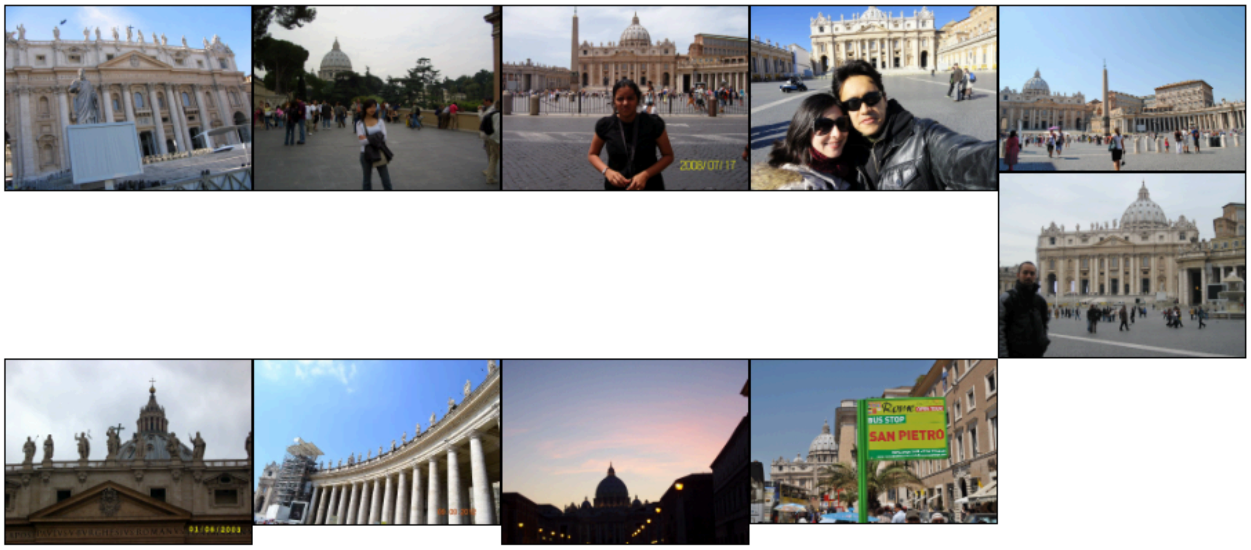
xfer_train_to_test.sh

```
$MYDIR="/home/ubuntu/anaconda3/envs/tensorflow_p36/mlnd/images/train/"

DIRS=`ls -l $MYDIR | egrep '^d' | awk '{print $9}'`

# and now loop through the directories:
for DIR in $DIRS
do
echo ${DIR}
cd ./${DIR}
mv `ls | head -50` /home/ubuntu/anaconda3/envs/tensorflow_p36/mlnd/images/test/predict/
cd ..
done
```

8. Here are few of the images set, that I analyzed during data preprocessing.



Implementation

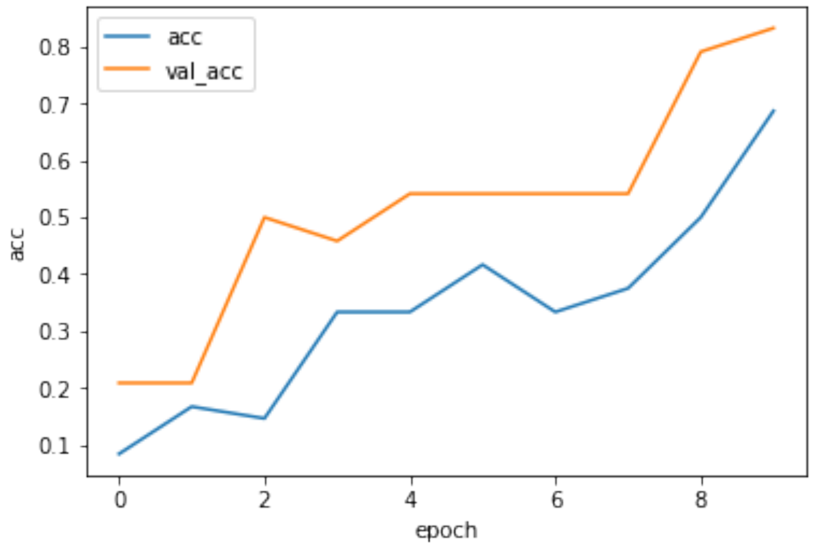
<u>**With smaller Dataset:**</u>

I started the implementation first with small dataset with 7 classes and approx. 100 images in total. Now I split the data in training, validation and testing datasets. I placed these images in separate folders using scripts to allow a flow from directories to data generators available in Keras.

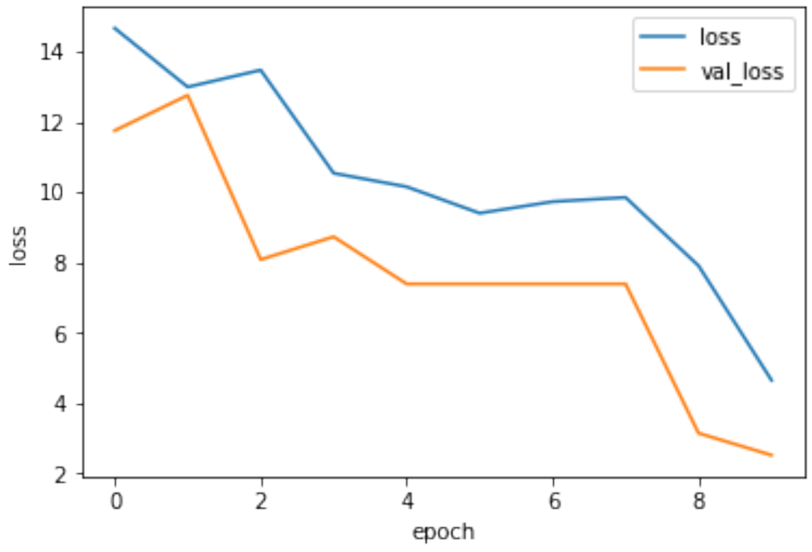
Image Augmentation: I utilized zoom, horizontal flips and rotations to account for noise in images.

VGG16: I started with training on VGG-16 model. It includes 16 layers and has 3x3 convolutional layers and max pooling layers combined to make deep convolutional network. I added dense and dropout layer to it. I initialized VGG16 weights on ImageNet dataset. Considering time, I only made the last block trainable and hence exploited the pre-trained ImageNet weights

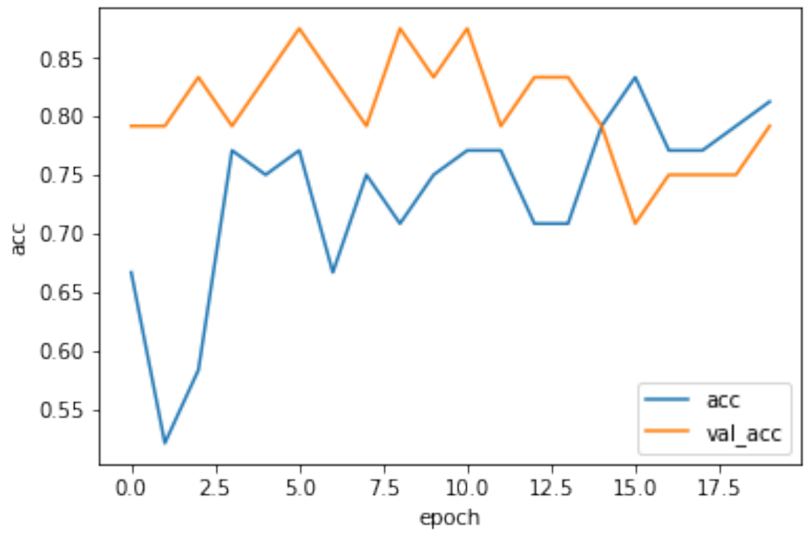
With 10 epochs, here is the plot for accuracy. Below we can clearly see that due to ImageNet weights initialization, accuracies increased quickly. This is the best overall model for accuracies.

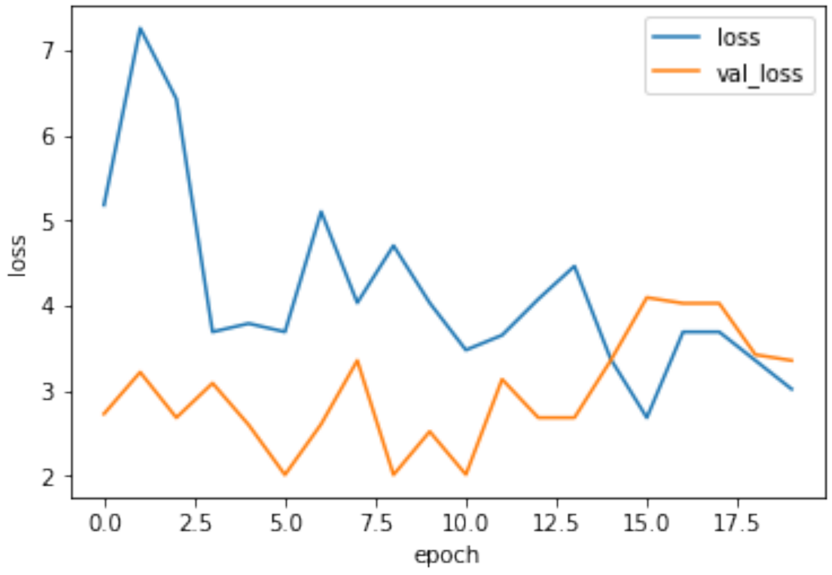


Below is the plot for loss. Well, the loss shows impressive trend in first attempt.

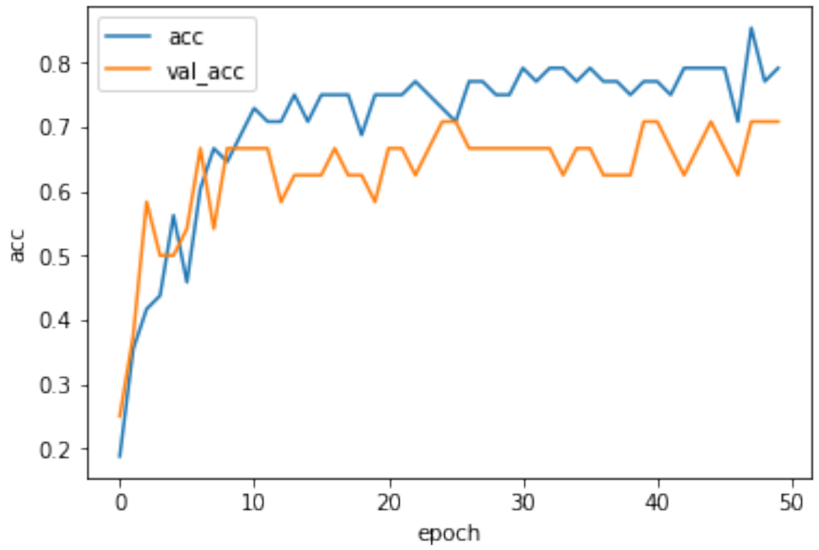


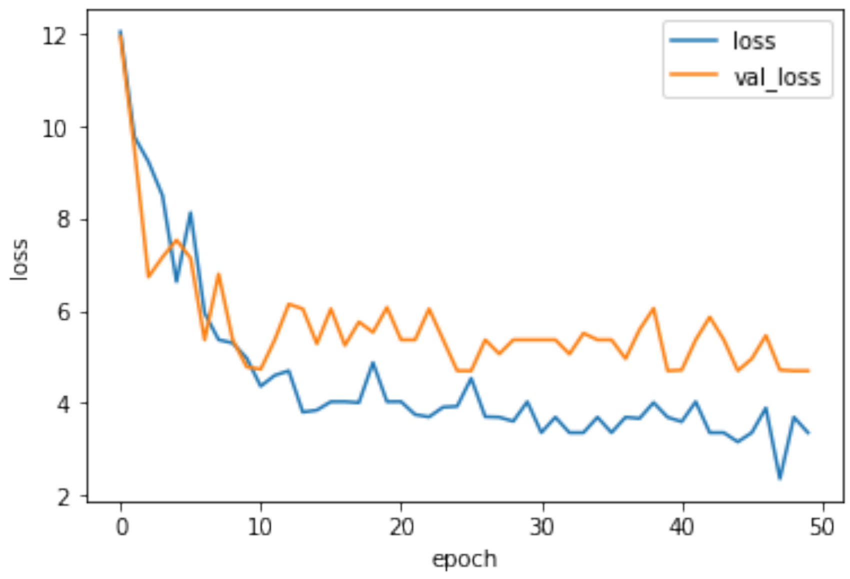
With 20 epochs, below is the trend for accuracy and loss.





With 50 epochs, below it started to converge. It means with increase in epochs, accuracy and loss goes very well with VGG16 model. Next, I will use Xception model with the same dataset.

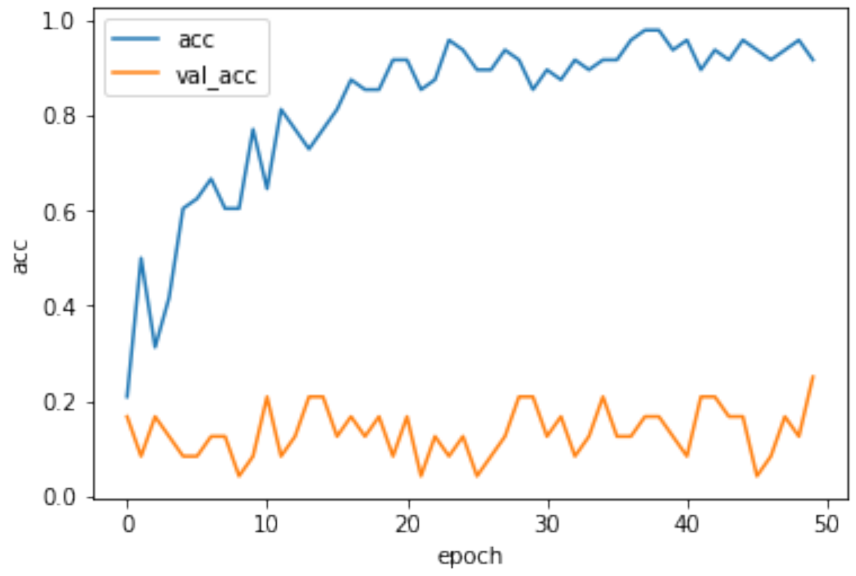


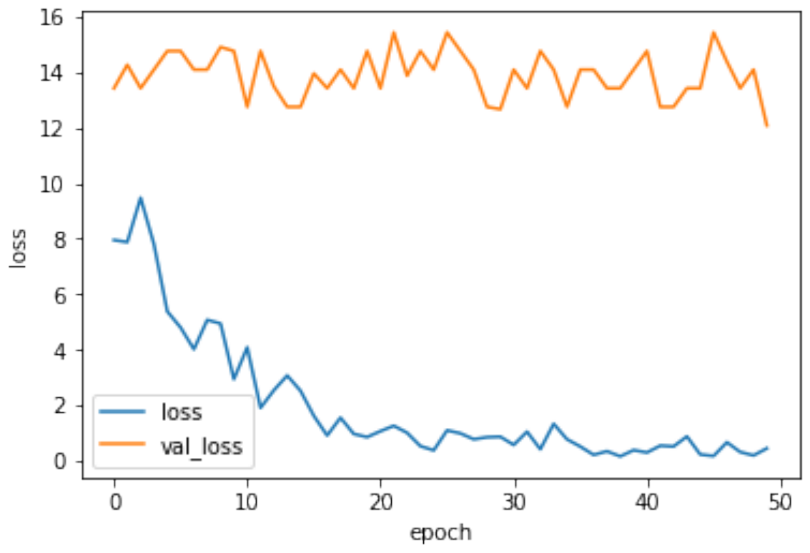


<u>Xception</u>:

I started now with Xception model. The Xception model is an extension of Inception model with with depthwise separable convolutions. Like other model, dropout and softmax layers are added. Training only the last layer shows the accuracy graph as below.

With 50 epochs, accuracy and loss trends not to converge and doesn't seem to be an option to go with but I will explore this model with larger dataset and explore the results.

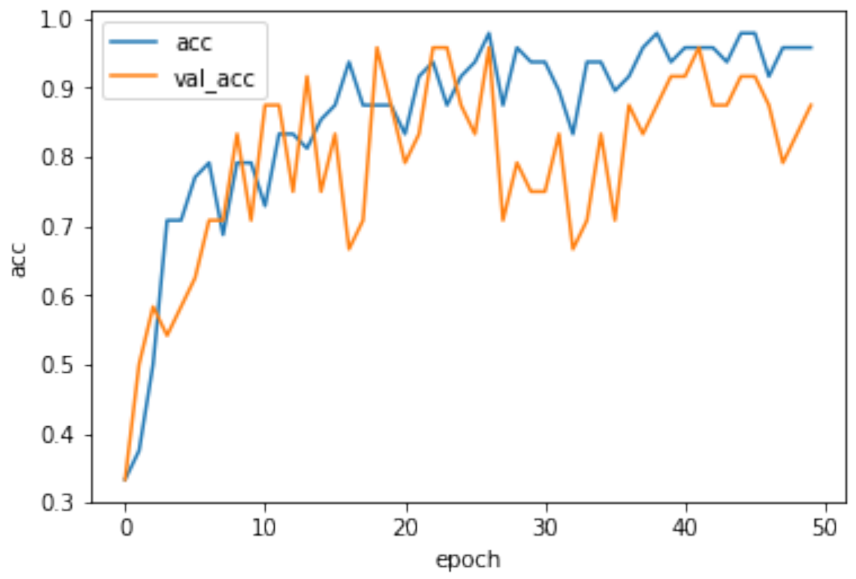


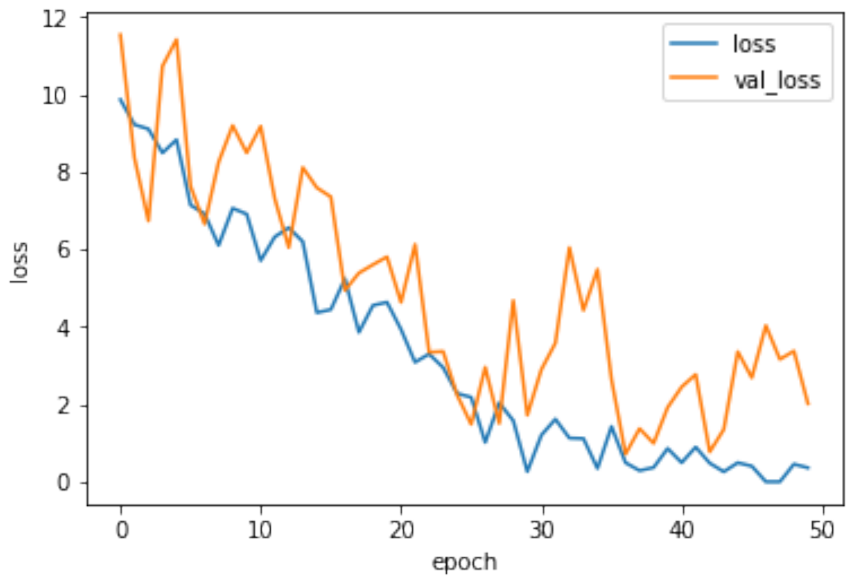


<u>Resnet50</u>:

For Residual Network model, I used Keras ResNet50 prebuild to ImageNet database. ResNet50 is a 50 layers residual network which used skip connections to avoid vanishing gradients. Again, I made the last convolutional layer trainable, added a 128-unit dense layer and an output layer with softmax activation. It gives following accuracy and loss graphs with 50 epocs.

Below plot shows signs of overfitting.





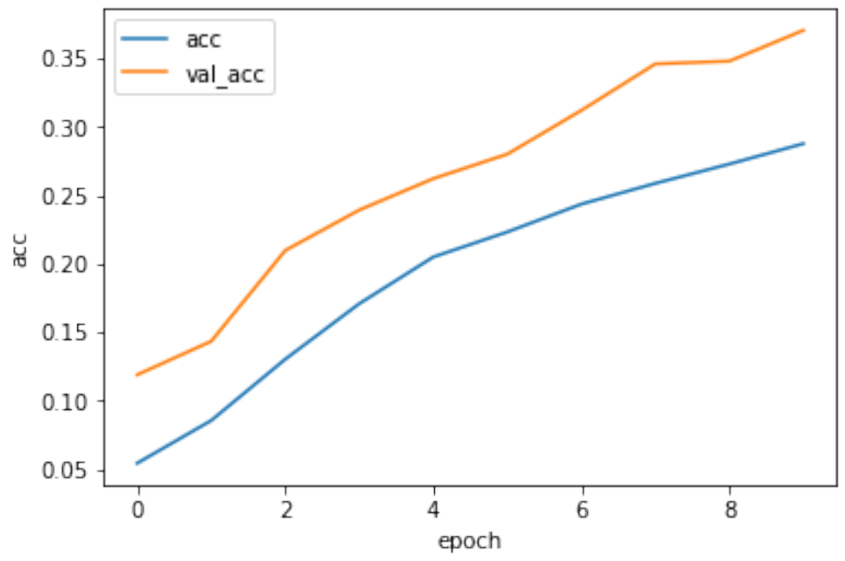
Large dataset

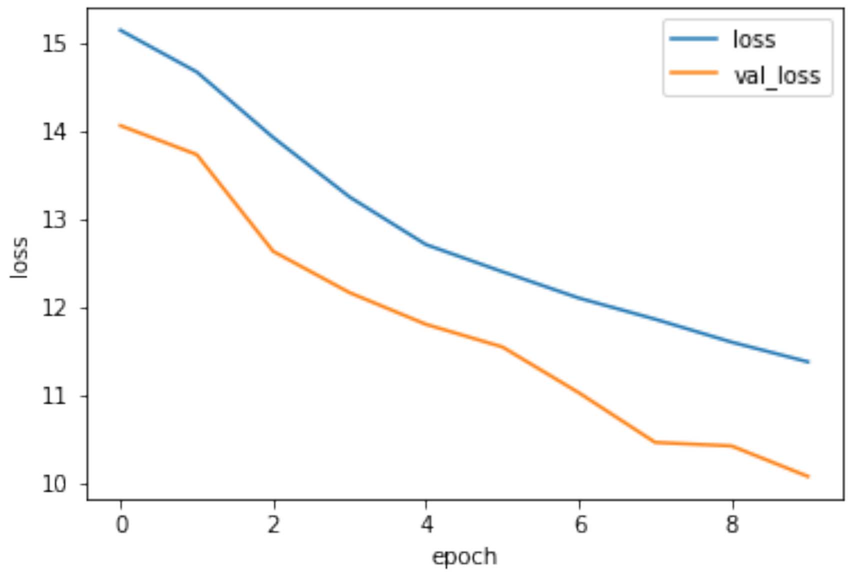
Now from train.csv, I selected a larger dataset with 50 different landmarks, each of which had images between 500 and 550 in the dataset. Comprising of more than 26000 images, this dataset is larger than the one, trained previously. Now I split the data into training, validation and testing datasets. Testing dataset is almost 10%, validation is approx. 17% and rest is used for training the model. I placed these images in separate folders using scripts to allow a flow from directories to data generators available in Keras.

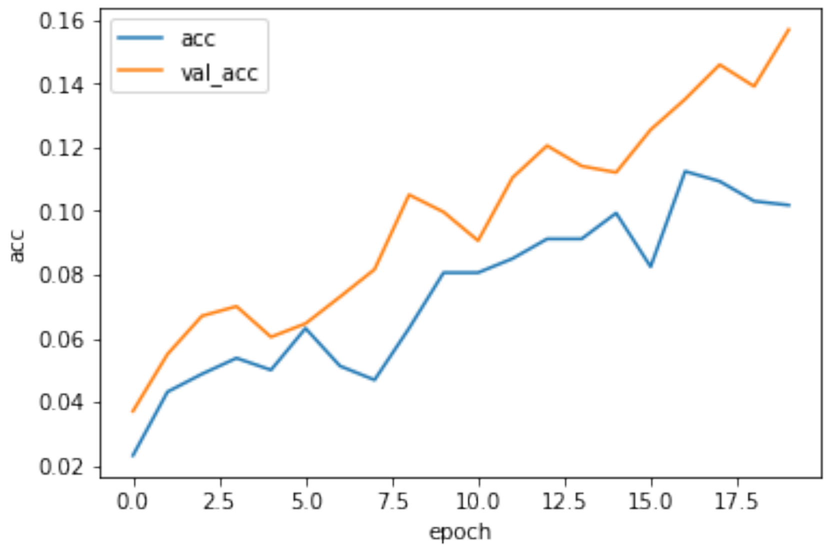
Image Augmentation: I utilized zoom, horizontal flips and rotations to account for noise in images.

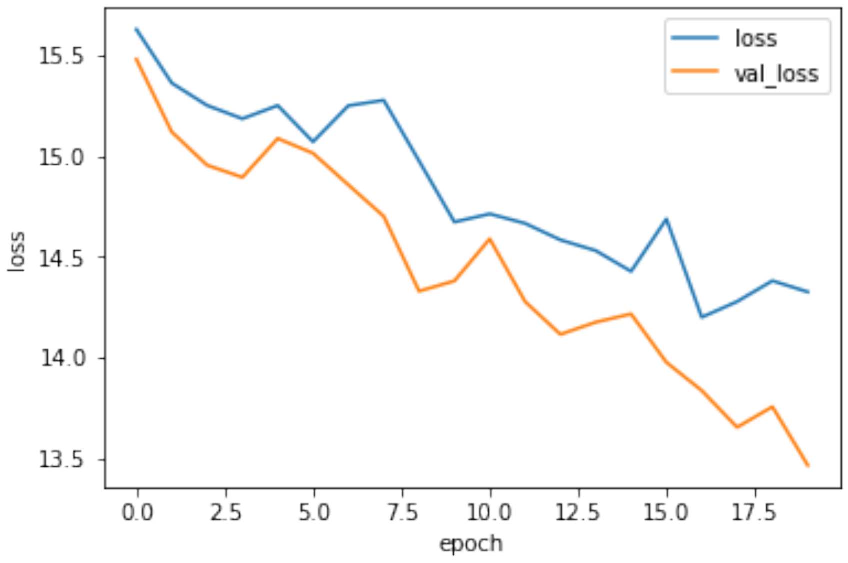
VGG16

VGG16 gives following accuracy and loss graphs with 10 and 20 epochs respectively.


acc
val_acc
0.35
0.30
0.25
0.20
0.15
0.10
0.05
acc
0
2
4
6
8
epoch

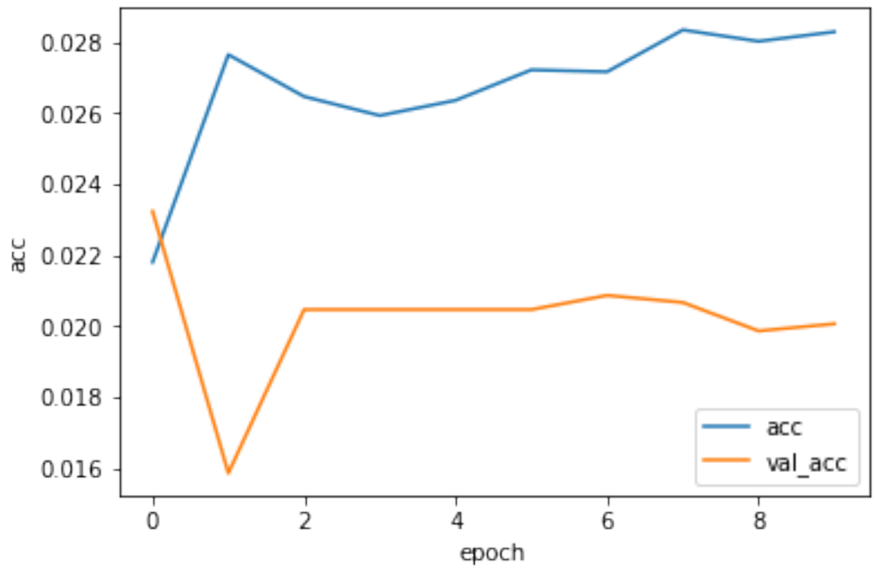

loss
val_loss
15
14
13
12
11
10
loss
0
2
4
6
8
epoch



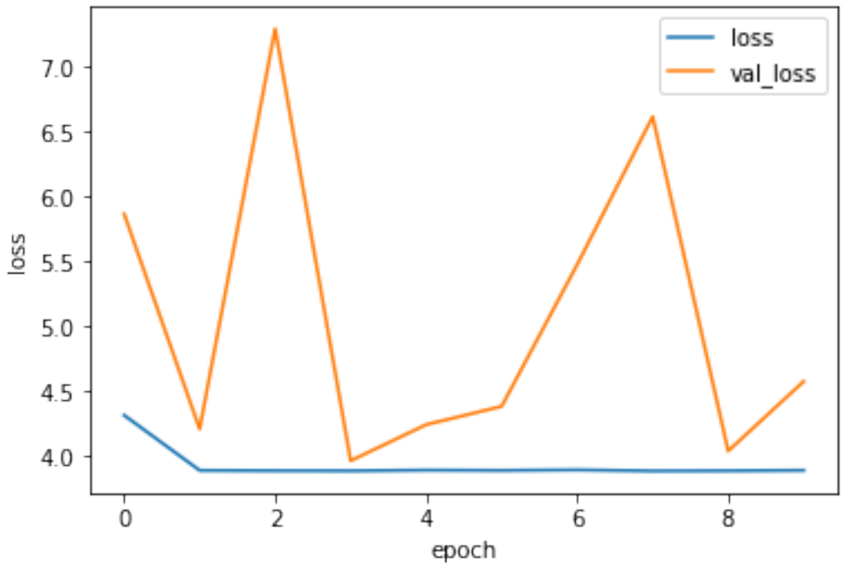


Xception:

Xception doesn’t seem to converge with larger dataset.

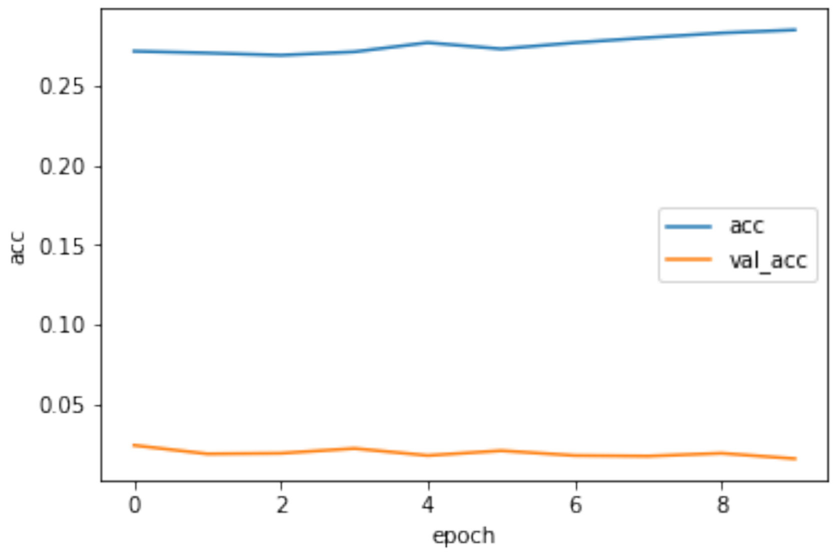


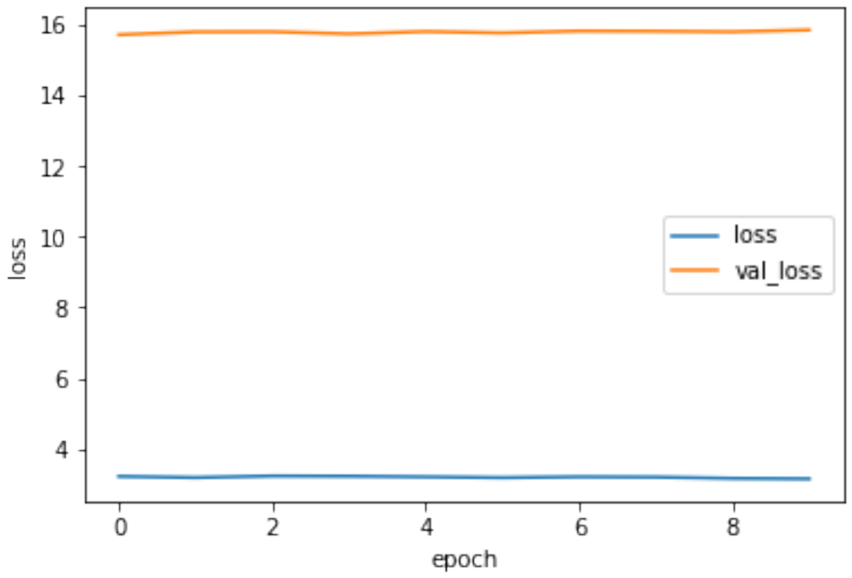
Here is the graph for validation loss.

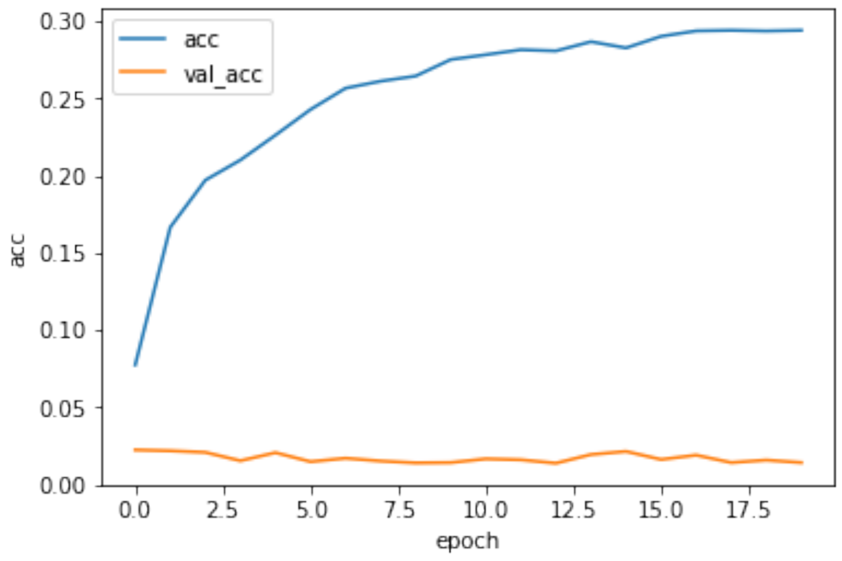


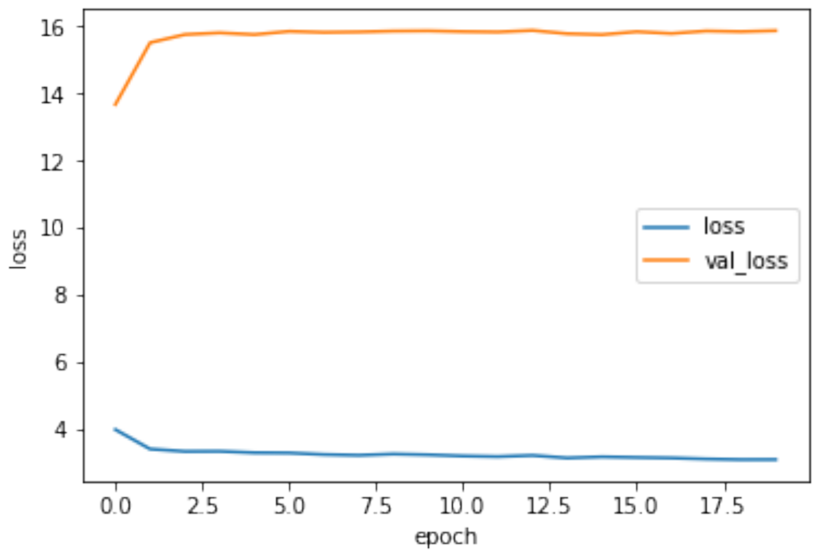
Resnet50

It gives following accuracy and loss graphs with 10 and 20 epocs respectively.


acc
val_acc
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8
epoch


loss
val_loss
16
14
12
10
8
6
4
0
2
4
6
8
epoch





Above graph for ResNet depicts that validation accuracy takes time to catch up with training accuracy. The deep residual network will take quite a long time for accuracy and loss to follow good patterns for validation set.

It requires lot of computation with increasing number of epochs and larger dataset. Given above analysis, VGG16 is the preferred model to go with.

Refinement

As mentioned in benchmark section, KNN classifier showed good accuracy. With CNN, the models considered were VGG16, Xception and ResNet50. VGG16 model performed best with small and large dataset.

The results were improved by tuning hyper parameters. Below are the analysis after tuning hyper parameters:

- Image size considered is 256x256.
- Number of epochs (training length): I started epochs as 10 for training various models and then increased it gradually. Below is the analysis with various epochs.

Small Image dataset

Epochs=10	**Batch size=16**
Accuracy: 0.6875	Loss: 4.6436
Val Accuracy: 0.8333	Validation Loss: 2.5112

Epochs=20	**Batch size=16**
Accuracy: 0.8125	Loss: 3.0221
Val Accuracy: 0.7917	Validation Loss: 3.3579

Epochs=50	**Batch size=16**
Accuracy: 0.7917	Loss: 3.3579
Val Accuracy: 0.7083	Validation Loss: 4.7011

Large Image Dataset

Epochs=10	**Batch size=16**
Accuracy: 0.2876	Loss: 11.3717
Val Accuracy: 0.3702	Validation Loss: 10.0681

Epochs=20	**Batch size=16**

Accuracy: 0.1019	Loss: 14.3254
Val Accuracy: 0.1571	Validation Loss: 13.4655

- batch size (number of images to look at once): To start, batch size used in all models were 16. Also tried batch size as 32. Result shown above used batch size as 16.

- optimizer (rmsprop, adam etc): Tuned optimizer to ‘rmsprop’ for better results. Used ‘ADAM’ optimizer to start with.

- learning rate (could be dynamic as it is how fast to learn): I tuned learning rate as well which was started with 0.001 and changed to 0.0001 during fine tuning.

IV. Results

Model Evaluation and Validation

Final model selected here is VGG16 as it has outperformed on the other two Xception and ResNet50 models. The final architecture and hyper parameters were chosen as they gave the best results among the tried options.

Following steps describe the complete steps of final model and the training steps:

1. Models used so far for evaluation here were VGG16, Xception and ResNet50.

2. VGG16 outperformed for smaller dataset with epochs as 10, 20 and 50. Even for larger dataset with 10 and 20 epochs, VGG16’ performance was better than other two models.

3. Steps per epoch is optimized with various performance improvements by adjusting hyper parameters.

4. Batch size is increased to 16 which turned out to be good for performance. I thought it to increase to 32 with increase in dataset but with all the computation involved and the result obtained I did stick to batch size as 16.

5. I originally used ‘ADAM’ optimizer but ‘RMSProp’ really helped model to converge.

6. To account for image capture conditions noise, I used Image augmentation with zoom, horizontal shift and rotation.

7. Model is evaluated as per the evaluation metric defined as Global Average Precision (GAP).

8. The robustness of the end model selected was verified with test dataset and it shows an impressive accuracy. Due to extremely large computation involved, it was hard to train the model in entire dataset but within these limitations, VGG16 came out very well.

Justification

The VGG 16 performed best given both the smaller and larger dataset, I used. If there be some way to deal with non-landmark images interacting with the classifiers described above and if I can integrate it with my VGG16 model, then I should be able to make my model extremely improved for landmark classification.

Test accuracy of final model selected (VGG16) is compared with other models alongwith the benchmark model (KNN classifier) and result looks good.

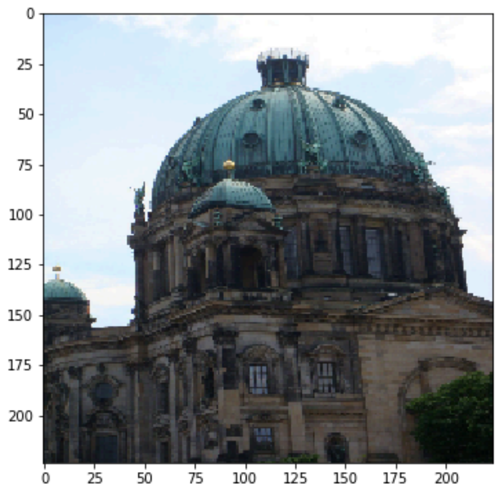
In summary, to train the model with entire set of training set (more than a million images), it will require a lot of computation and better hardware but the selected dataset and final model can easily be generalized.

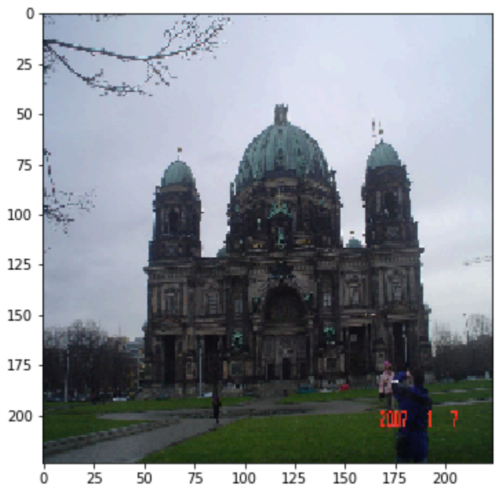
V. Conclusion

Free-Form Visualization

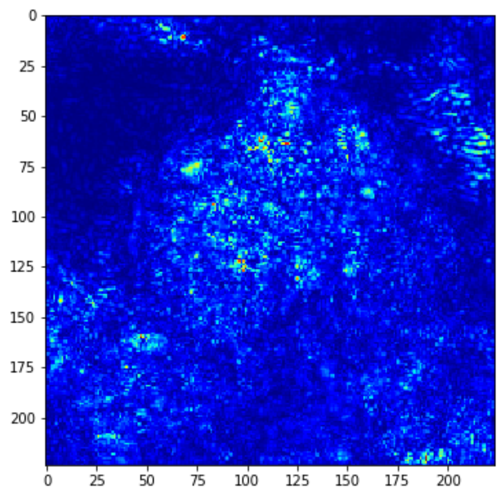
To visualize, let’s take an image from the given dataset, Berlin Cathdral, which has the category defined as 1553. I have used Keras-vis library [5] for various visualizations here. Here is the original image

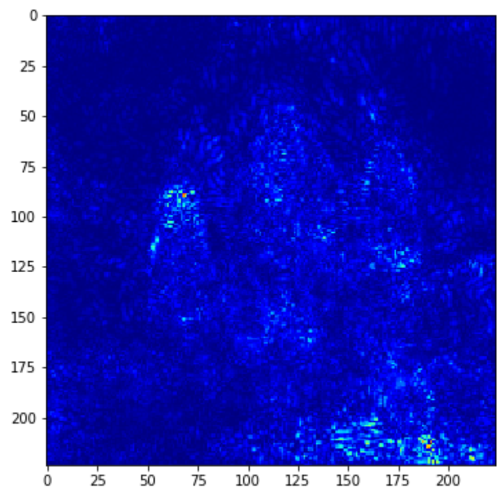
Below are the other 2 example images for the same category.



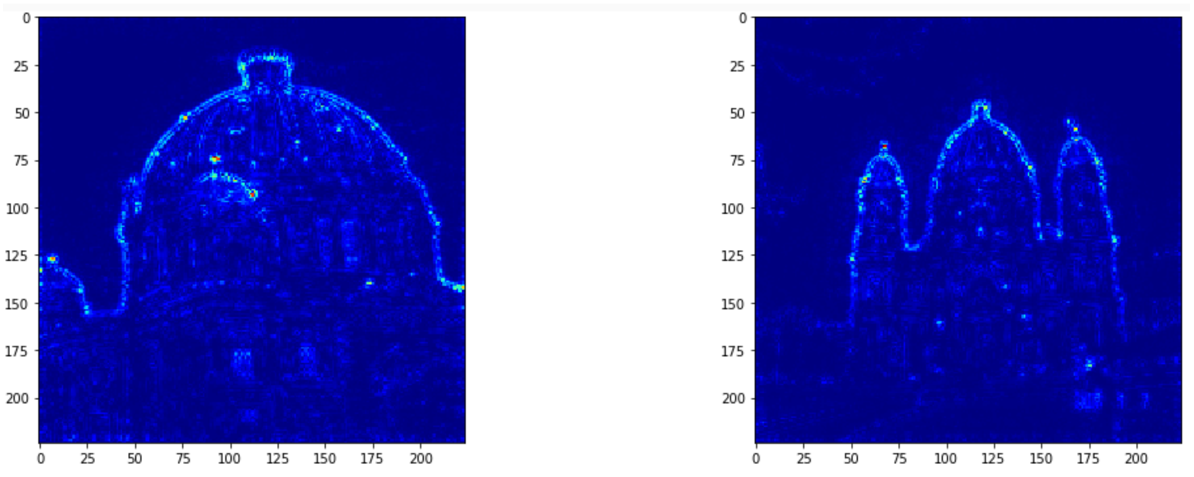


Saliency map is a good way to visualize which area of image has the most impact on the output i.e. it highlights which part in the image will play most important role in determining the category or class. Below are the saliency maps of the above images with no backprop_modifier setting.

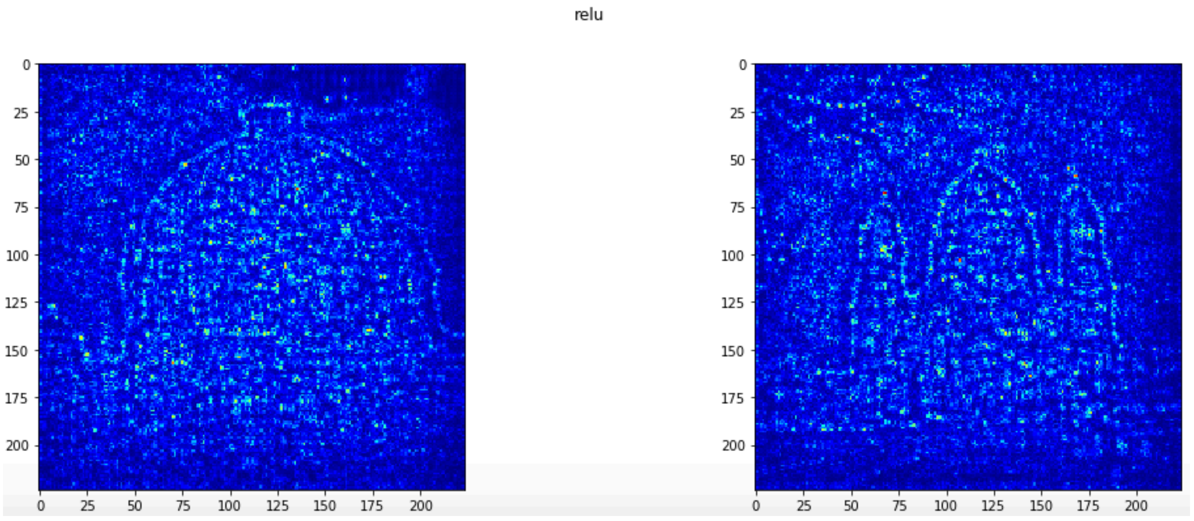




Now let’s see saliency maps with backprop_modifier set as ‘guided’ and ‘relu’ respectively. It is quite visible below that saliency maps with the ‘guided and ‘relu’ modifiers have a clear outline of Berlin Cathedral. Also, the outline of domes and spires having most impact on classifying the image.



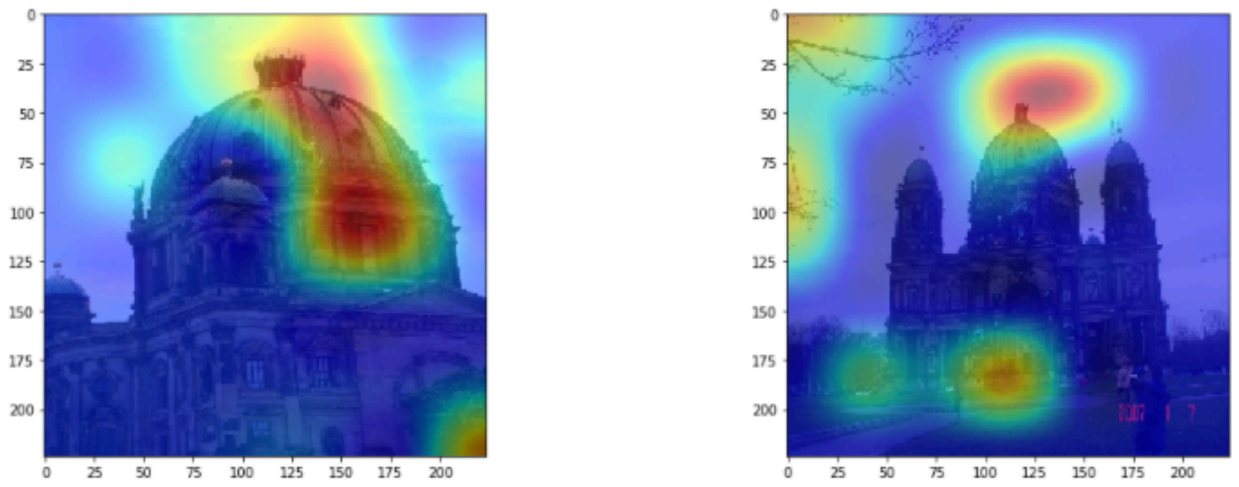
```
:matplotlib.figure.Figure at 0x1c24b4e160>
```



My next visualization uses activation maps which doesn't rely on gradient of the output. It uses penultimate (pre-Dense layer) Conv layer output. Below are the **classification maps** with first being vanilla (without any backpropagation modifier) and second and third with 'guided' and 'relu' backpropagation modifiers respectively.

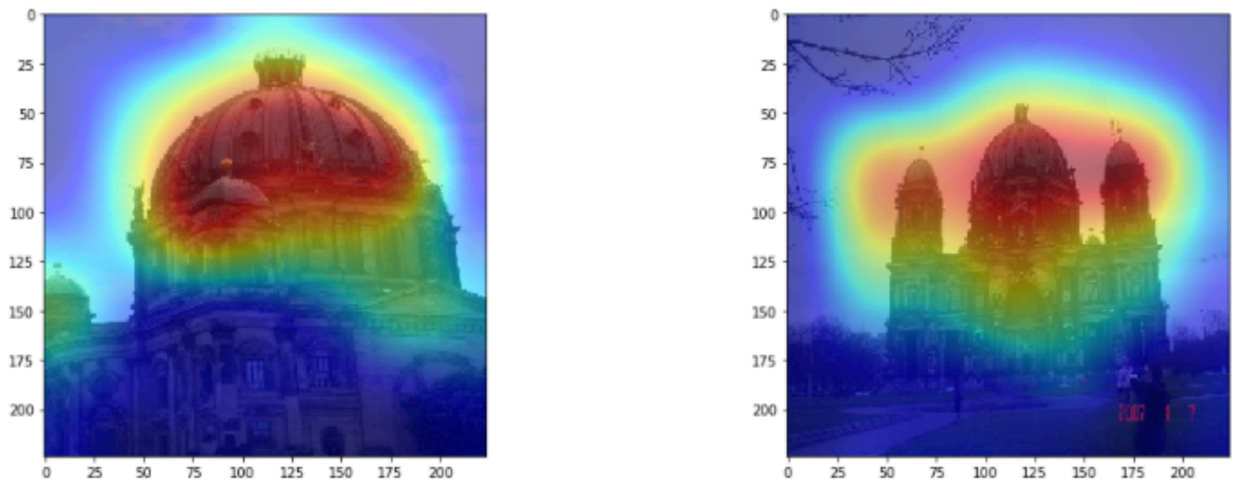
Again, with below maps its quite clear that the areas which have most activations are the dome and spires. In this case too, the 'guided' backpropagation modifier outperforms on others.

vanilla



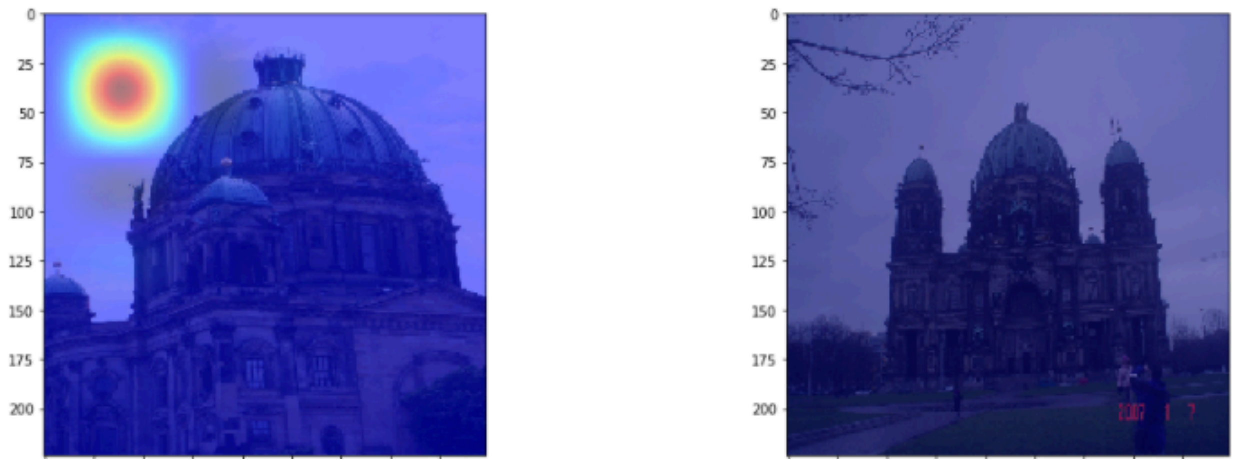
```
<matplotlib.figure.Figure at 0x1c25aa6588>
```

guided



```
<matplotlib.figure.Figure at 0x1c302aca58>
```

relu



Reflection

The process used for this project can be summarized through below steps:

1. The problem/challenge dataset was downloaded.
2. I created an AWS deep learning instance (p2/p3) to apply deep learning models on the image dataset.
3. I analyzed the data and visualized it to get insights about the dataset.
4. I took KNN (K-Nearest Neighbor) as the benchmark classifier.
5. I then applied deep learning models on the dataset which includes VGG16, Xception and ResNet50 models.
6. All these 3 models were evaluated through their accuracies and final metrics calculated.
7. I applied all models through Keras which used Tensorflow backend.

I found steps 5 and 6, the most difficult, as it required lot of back and forth computation and very time consuming. It required lot of understanding to setup GPU enabled EC2 instances to handle this large dataset and then set it up for Keras, tensorflow and jupyter notebook. Overall it was great learning and I enjoyed it in every step.

To me interesting aspects of the projects is to get hands on in Keras library. Also learned a lot on how to do cloud (AWS) set up for deep learning projects.

Improvement

As stated earlier, the VGG 16 performed best given both the smaller and larger dataset, I used. There should be some way to deal with non-landmark images interacting with the classifiers described above. If I can integrate it with my VGG16 model, then I should be able to make my model extremely improved for landmark classification. Also, I would like to run the VGG16 on the entire dataset, for many epochs, so I could have very well-trained weights, that wouldn’t be over fitted to any dataset.

Reference:

[1] F. Perronnin, Y. Liu, and J.-M. Renders, "A Family of Contextual Measures of Similarity between Distributions with Application to Image Retrieval," Proc. CVPR'09

[2] David J. Crandall, Yunpeng Li, Stefan Lee, and Daniel P. Huttenlocher "Recognizing Landmarks in Large-Scale Image Collections". IEEE, 2009.

[3] Andre Araujo and Tobias Weyand "Google-Landmarks: A New Dataset and Challenge for Landmark Recognition" Google Research, Thursday, March 1, 2018. https://ai.googleblog.com/2018/03/google-landmarks-new-dataset-and.html

[4] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions" asXiv 2016.

https://www.kaggle.com/c/landmark-recognition-challenge
https://www.kaggle.com/google/google-landmarks-dataset
https://aws.amazon.com/blogs/machine-learning/get-started-with-deep-learning-using-theaws-deep-learning-ami/
https://colab.research.google.com/notebooks/welcome.ipynb

[5] https://raghakot.github.io/keras-vis/visualizations/saliency/