**LANDMARK RECOGNITION**

**TEAM MEMBERS : S. NANDHINI (2017103559) P. AGALYA (2017103505)**

**Abstract**

Our project is about landmark recognition,in which will tell you which class the input image belongs to from the trained set of labelled images. This is done by training a neural network on few thousand images of 25 landmarks and make the learn to predict the queried landmark.

**The main development we have done is we have tried different models for our dataset and summarized its accuracy and found which has greater accuracy for our dataset and does not overfit. We have also choosed optimum epochs for the model such that it neither underfits (less number of epochs) nor does it overfit ( accuracy in the training set increases whereas that of validation set decreases).**

**Dataset**

Our image dataset consists of approximately

* 25 landmarks
* 45,180 database images (1.4K~2K per landmark)
* 10,000 positive test images for evaluation (400 per landmark)

**Script for downloading**

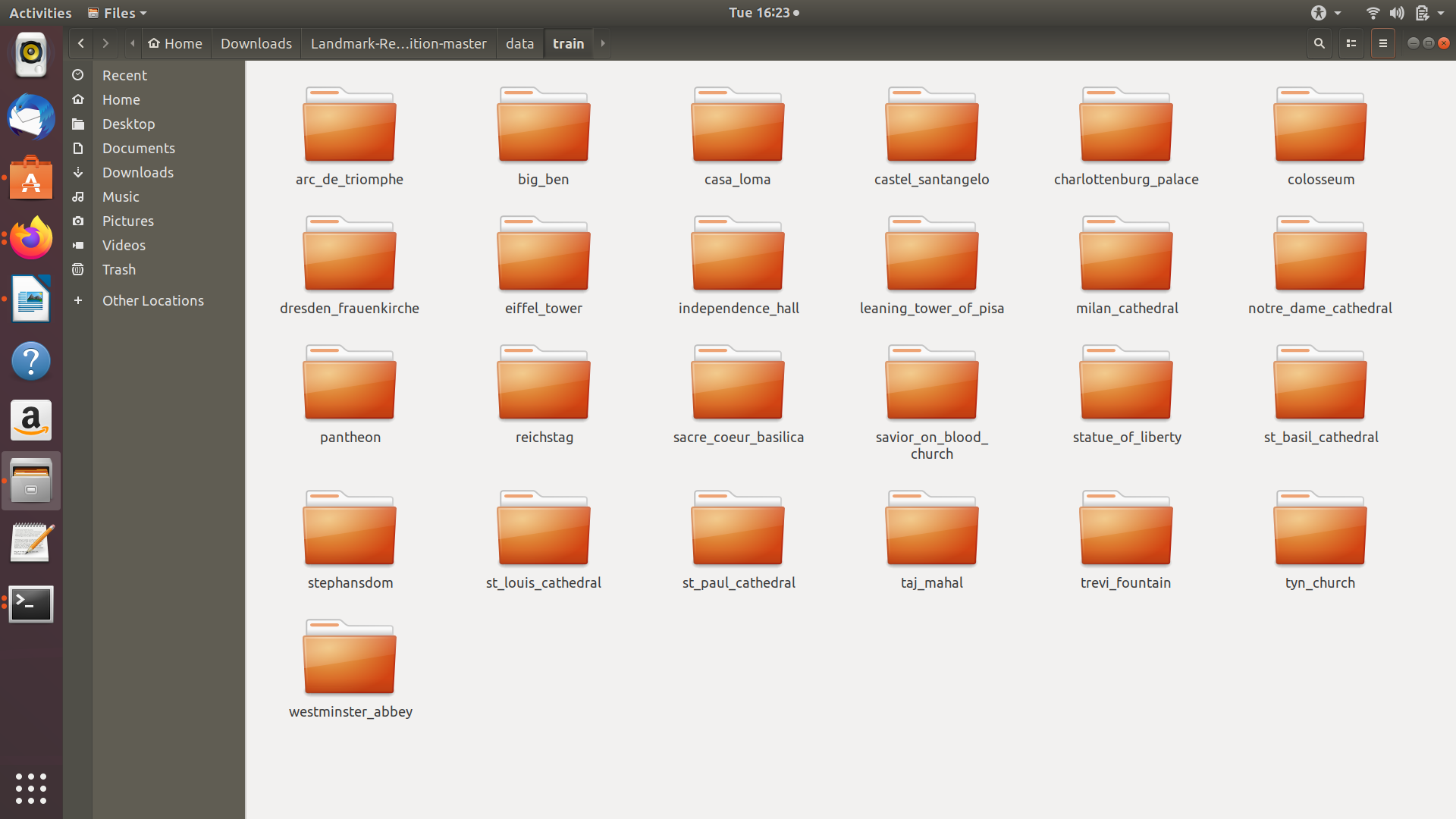
$ python3 utils/download\_data.py 'train'

$ python3 utils/download\_data.py 'validation'

After downloading the dataset, we have two folders: ‘train’ and ‘validation’ in a directory ‘data’. Each folder contains sub-folder for each landmark class and they are as follows:

**25 Landmarks**

['arc\_de\_triomphe' , 'big\_ben' , 'casa\_loma' , 'castel\_santangelo' , 'charlottenburg\_palace', 'colosseum', 'dresden\_frauenkirche', 'eiffel\_tower', 'independence\_hall', 'leaning\_tower\_of\_pisa', 'milan\_cathedral', 'notre\_dame\_cathedral', 'pantheon', 'reichstag', 'sacre\_coeur\_basilica', 'savior\_on\_blood\_church', 'statue\_of\_liberty', 'stephansdom', 'st\_basil\_cathedral', 'st\_louis\_cathedral', 'st\_paul\_cathedral', 'taj\_mahal', 'trevi\_fountain', 'tyn\_church', 'westminster\_abbey']



For each landmark, there is a subfolder containing four files:

* **list\_db.txt** -- the list of database images
* **list\_test.txt--** the list of test images

General overview of the observed results:

|  |  |  |
| --- | --- | --- |
| MODEL | LOSS | ACCURACY |
| Basic CNN model | 1.5216951492294024 | 0.7202969245582238 |
| Xception V1 model | 0.4471 | 0.8808 |
| VGG16 | 0.17596535271980915 | 0.9692272096251735 |

**Steps involved:**

1.Data Cleaning and Data Preprocessing

2.Building the model

3.Compile the model and train the model

4.Evaluate the model

5.Evaluate on test images

6.Image predictions

**I**n our project we, have tried three different models namely

* Basic CNN
* Xception V1 model, with weights pre-trained on ImageNet.
* VGG16 pre-trained on the Google ImageNet dataset

We have trained the model to fit the dataset to the model and evaluated its loss and accuracy.On comparing the three,it can be observed that the basic cnn model,

didn't achieve a decent accuracy and the model overfits.In Xception V1 model,better accuracy is achieved but still the model overfits and VGG16 model gives the best accuracy.

**Data Cleaning and data preprocessing**

In this process we removed the images that are not present or invalid(did not download properly) from the train directory and validation directory.

First we need to read and preprocess our images. In Tensorflow Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class. We set parameter rescale - is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. factor. We have used flow\_from\_directory()( a method in keras which is used to read the image from the respected folders) to generate batches of image data directly from our jpgs in their respective folders.

**flow\_from\_directory(directory)**-Takes the path to a directory, and generates batches of augmented/normalized data. Yields batches indefinitely, in an infinite loop.

#### **Basic CNN: Building a Sequential model**

We build a sequential model and add 3 convolution layers with a ReLU(Rectified Linear activation function) and followed by max-pooling which is used to reduce the dimensions of an image by taking the maximum pixel.

#### **Training the model**

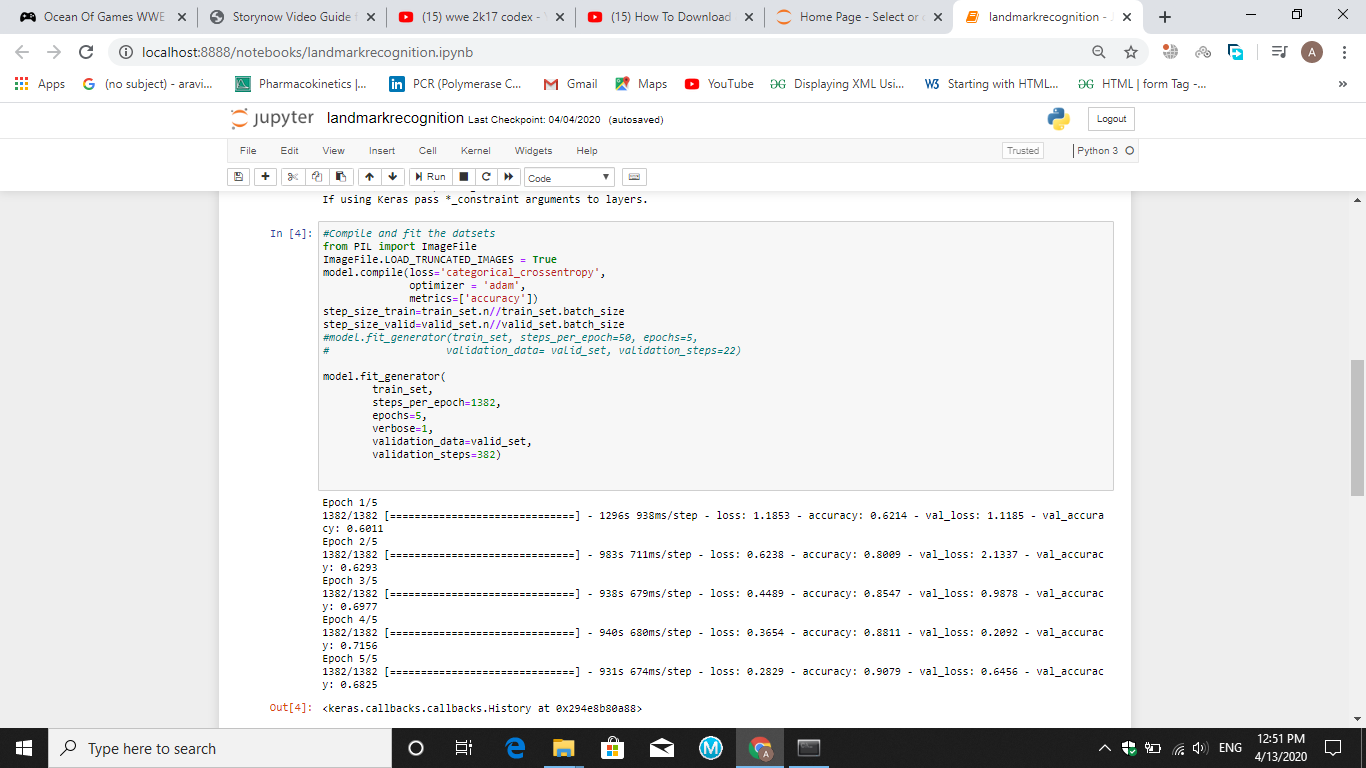
We now compile the model with a categorical cross entropy loss function.We then fit the dataset to the model, i.e **we train the model for 10 epochs**. After training the model, we evaluate the loss and accuracy of the model on the test data and print it.

tf.keras.Model.compile takes three important arguments:

**optimizer:** This object specifies the training procedure. Pass it optimizer instances from the tf.train module, such as tf.train.AdamOptimizer, tf.train.RMSPropOptimizer, or tf.train.GradientDescentOptimizer.

**loss:** The function to minimize during optimization. Common choices include mean square error (mse), categorical\_crossentropy, and binary\_crossentropy. Loss functions are specified by name or by passing a callable object from the tf.keras.losses module.

**metrics:** Used to monitor training. These are string names or callables from the tf.keras.metrics module.



**Epochs:5**

**After training the model,the loss and accuracy at the end of 5 epochs are**

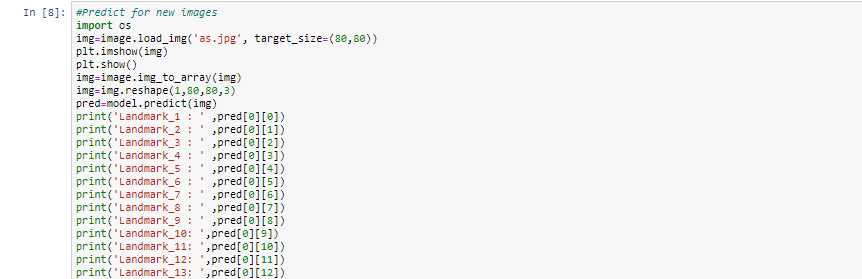
**val\_loss : 0.6456**

**val\_acc : 0.6825**

**loss : 0.2869**

**acc : 0.9079**

**Output:**





The value of 1 in Landmark\_2 indicates that the given landmark is Reichstag.

**Drawback:**

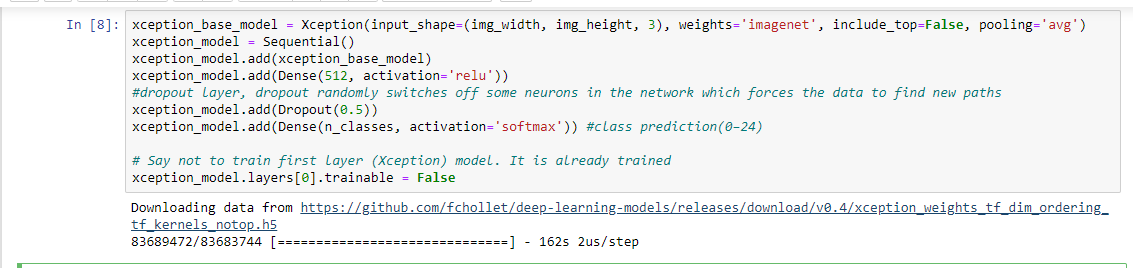
On training the model and evaluating it, we couldn’t achieve more accuracy and overfits.To overcome this and increase the accuracy,transfer learning approached was used.

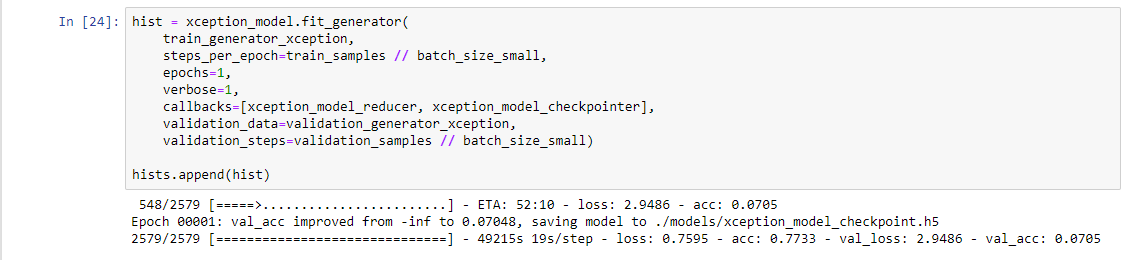
**Xception V1 model**

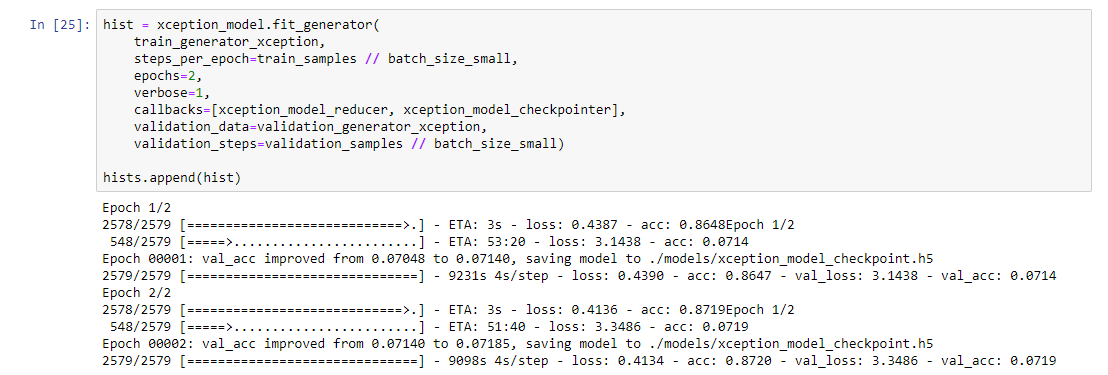
Transfer learning is about borrowing CNN architecture with its pre-trained parameters . When we train our own data on the top of the pre-trained parameters, we can easily reach to the target accuracy.The model is Xception V1 model, with weights pre-trained on ImageNet.Then the pre-trained model is trained and evaluated after data preprocessing using the same methods followed early for our built model.

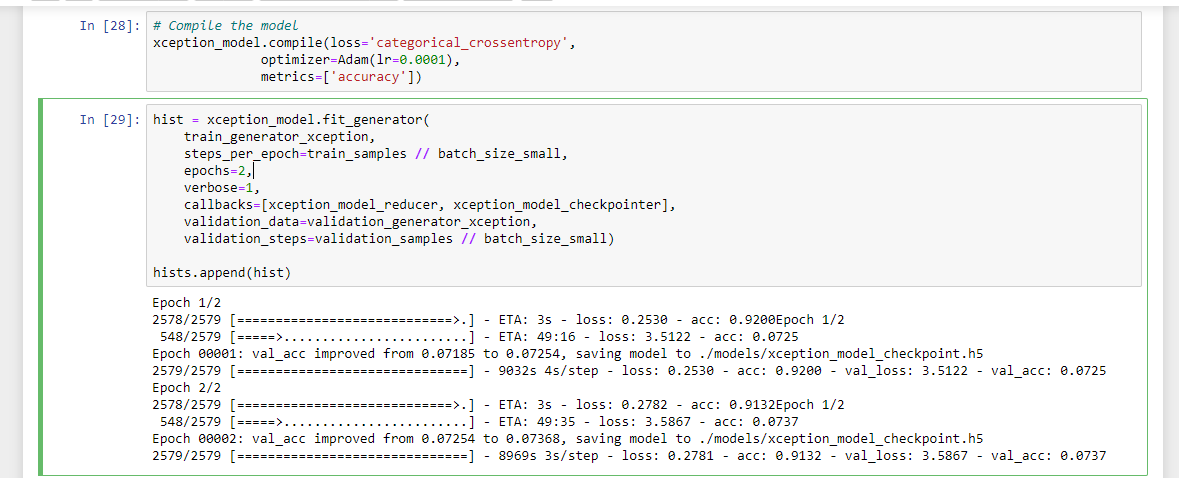
**Training**

We compile the model with Adam optimizer.Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.









**Evaluation**

**After evaluating the model,**

**val\_loss : 3.5867**

**val\_acc : 0.0737**

**loss : 0.2781**

**acc : 0.9132**

After Transfer learning,our model achieved better accuracy and still it overfits.

So we tried another way to eliminate the overfitting problem and to increase the accuracy using fine tuning.

**VGG16**

An another way used here to classify the images using a Convolutional Neural Network i.e VGG16 pre-trained on the Google ImageNet dataset.Here we have incorporated transfer learning with feature extraction.

**The architecture of VGG16**: the input layer takes an image in the size of (224 x 224 x 3), and the output layer is a softmax prediction on 1000 classes. From the input layer to the last max pooling layer (labeled by 7 x 7 x 512) is regarded as the feature extraction part of the model, while the rest of the network is regarded as the classification part of the model.

#### **Bottleneck features**

First of all we use the remaining portion of the model as a feature extractor called ‘Bottleneck Features‘ (i.e. the last activation maps before the fully-connected layers in the original model). After that we train a small fully-connected network on the bottleneck features, so we get the classes as outputs for our problem.

Then the method flow\_from\_directory()is used here to get the classes and save the corresponding bottleneck features of the VGG16. Furthermore we use an ImageDataGenerator to rescale the images.

Again our top-model is trained and evaluated to eliminate overfitting problem after using feature extraction and fine-tuning .

#### **Train the top model**

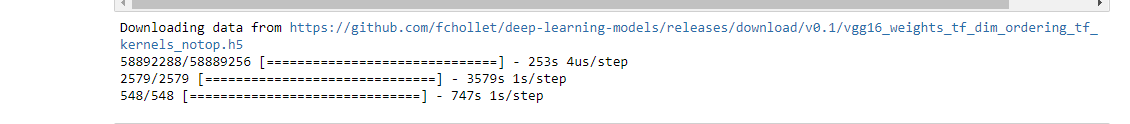
With the bottleneck features saved, now we are ready to train our top model. We create a small fully-connected network using the bottleneck features as input.

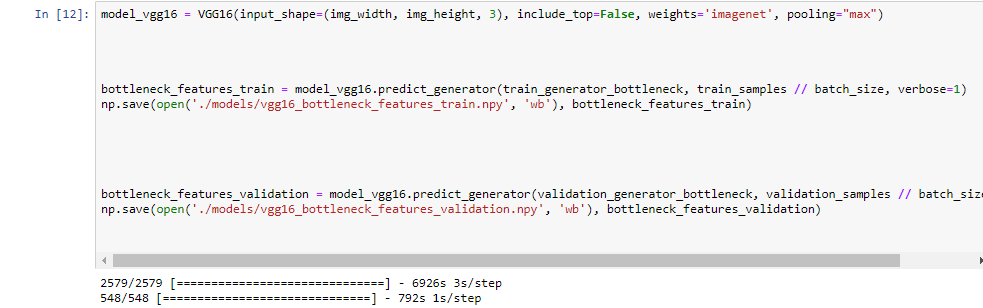
Initially the model is made to run for 3 epochs completely and then it runs for 30 epochs where early stopping is made at the thirteenth epoch to eliminate the problem of overfitting.

**The loss and accuracy after training**

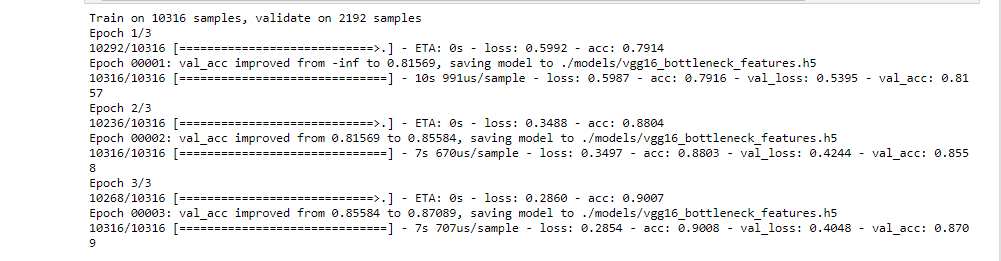
**val\_loss: 0.4048** **val\_acc: 0.870**

**loss: 0.2854 acc:0.9008**







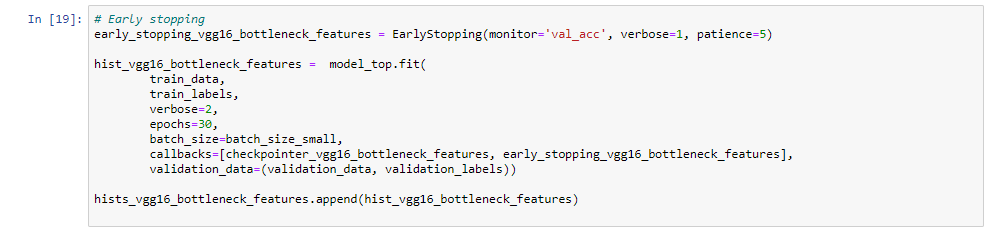


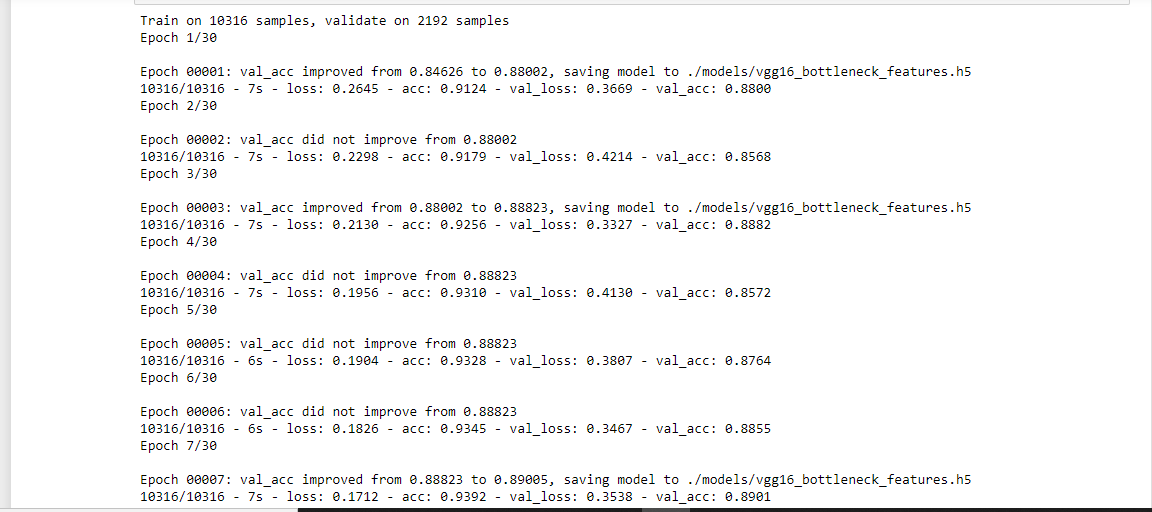
**Early stopping** is a form of [regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)) used to avoid [overfitting](https://en.wikipedia.org/wiki/Overfitting) when training a learner with an iterative method, such as [gradient descent](https://en.wikipedia.org/wiki/Gradient_descent). Such methods update the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set.

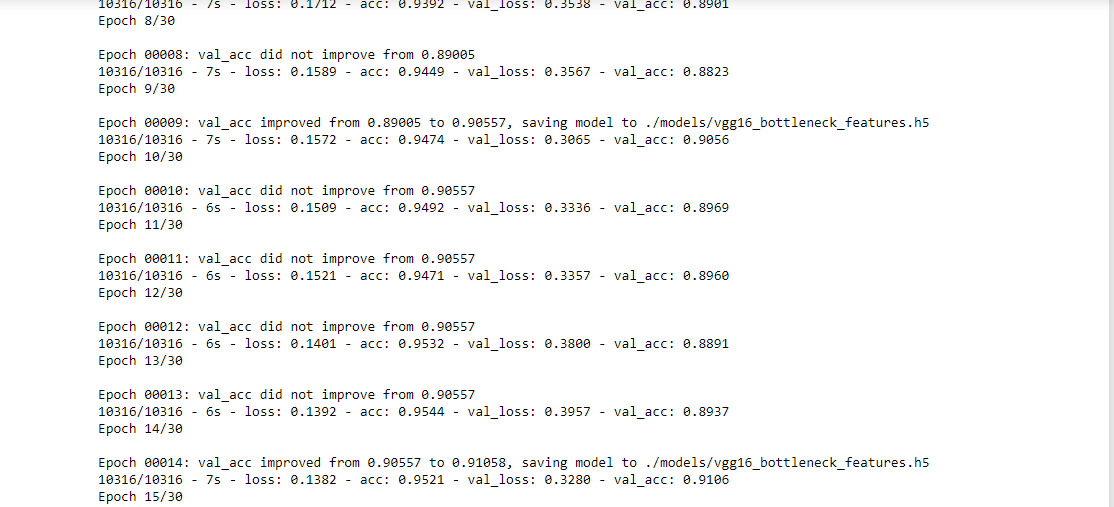
**The loss and accuracy after early-stopping in training,**

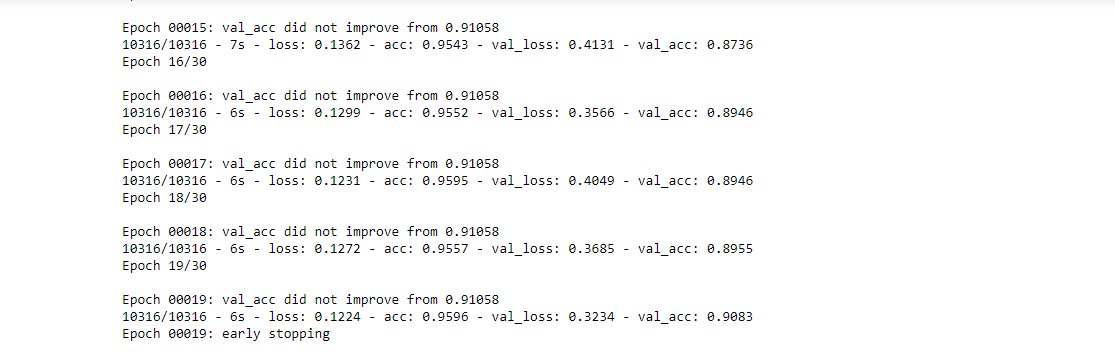
**val\_loss: 0.3234** **val\_acc: 0.9083**

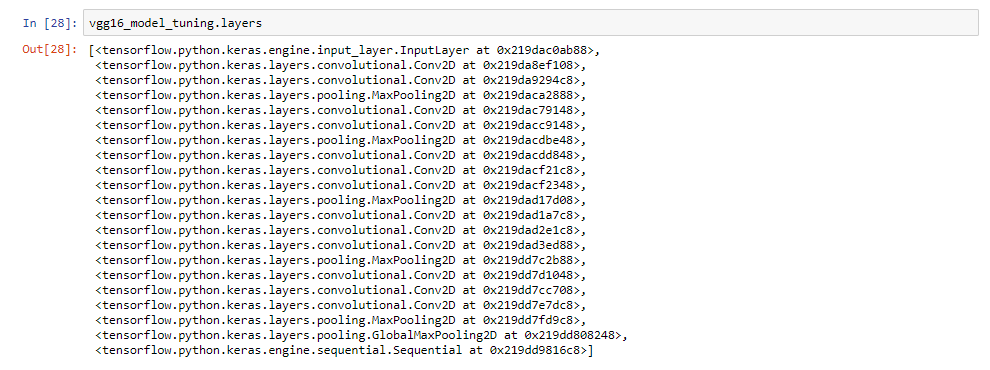
**loss: 0.1224 acc:0.9596**







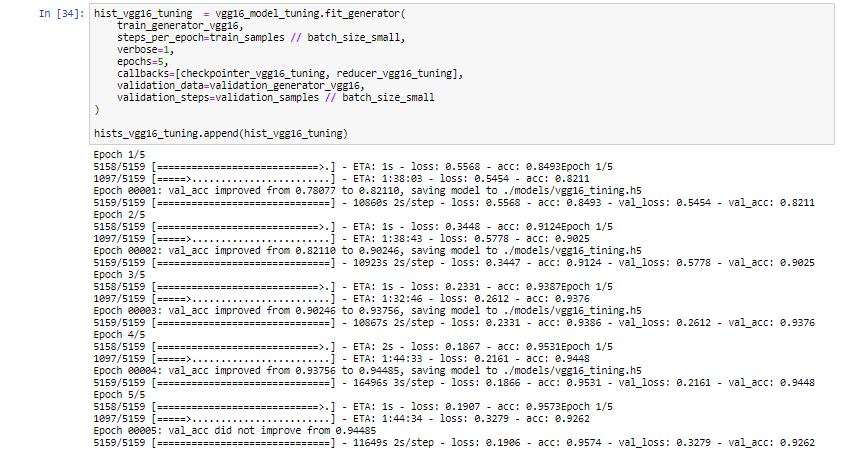


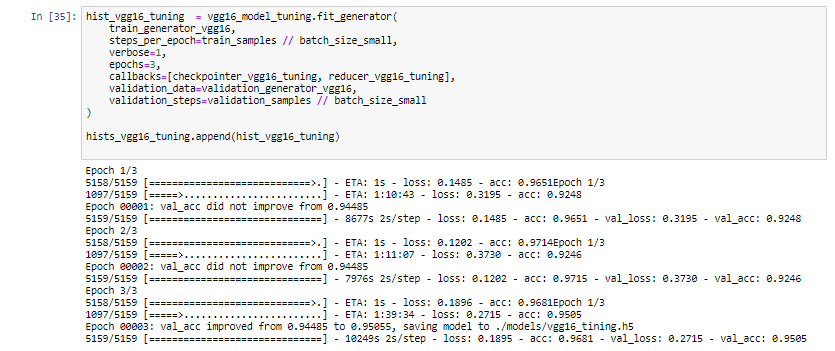


#### **Fine-tuning the top layers of a pre-trained network (VGG16)**

The model was not yet performing well, because the weights are still the weights of the ImageNet – in conclusion, so we have to fine-tune it. Then the model was fine-tuned.







**After fine-tuning,**

**val\_loss : 0.2715**

**val\_acc : 0.9505**

**loss : 0.1895**

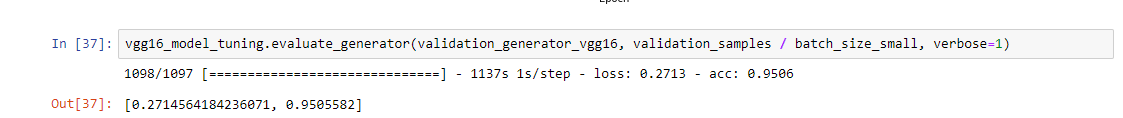
**acc : 0.9681**

#### **Evaluation of the fine-tuned model**

Then the fine-tuned model is evaluated and the accuracy of our model was increased.

**After evaluation of fine-tuned model, the loss and accuracy respectively are**

**[0.2714564184236071, 0.9505582]**

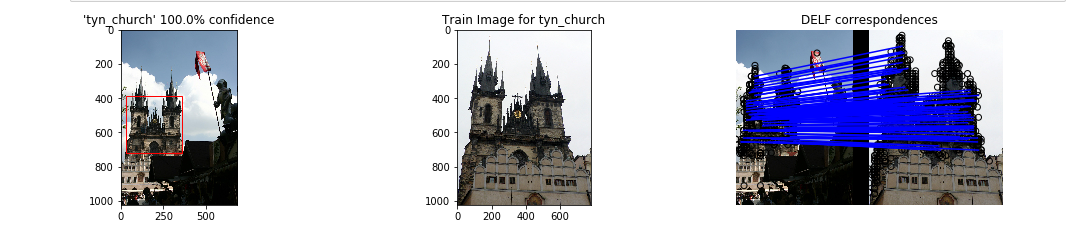


### **Prediction**

Since VGG16 model gave the best accuracy of all we have used VGG16 for landmark recognition. We will use DELF to draw matches of the image features and randomly chosen image from training dataset of the predicted class. Also based on the inliers we will draw a box around the area of the predicted landmark by considering RANSAC algorithm which provides geometric verification.

DELF: DEep Local Features detects and describes semantic local features which can be geometrically verified between images showing the same object instance.





The first image in the above figure was given for testing and the trained model predicted the landmark correctly and provided the second image as output thus recognizing the landmark correctly.

### **Conclusion**

We tried different approaches to train a NN(Neural Network) to recognize 25 landmarks. Our best model achieved 97% accuracy on validation set. We applied 'Transfer Learning' strategy with feature extraction('bottleneck features') using pre-trained VGG16 on ImageNet with top-model classifier and fine-tuning the model.