**Abstract**

Recently, deep learning has gained exceptional popularity due to its outstanding performances in many machine learning and artificial intelligence applications. Among various deep learning models, convolutional neural network (CNN) is one of the representative models that solved various complex tasks in computer vision since AlexNet, a widely-used CNN model, has won the ImageNet challenge1 in 2012. Even with such a remarkable success, the issue of how it handles the underlying complexity of data so well has not been thoroughly investigated, while much effort was concentrated on pushing its performance to a new limit. Therefore, the current status of its increasing popularity and attention for various applications from both academia and industries is demanding a clearer and more detailed exposition of their inner workings. To this end, we introduce two major contributions: 1) a network visualization module for monitoring the underlying process of a convolutional neural network using a filter-level 2D embedding view and 2) an interactive module that enables real-time steering of a model. We present several use cases demonstrating benefits users can gain from our approach.

The rest of this project is organized as follows. Section 2 discusses related work. Section 3 presents detailed description of our system and its visual components. Section 4 presents usage scenarios. Finally, Section 5 concludes our discussion with plans for future work.

**Introduction**

Recently, deep learning has made major breakthroughs in many machine learning problems such as computer vision and speech recognition. A traditional neural network model is basically composed of multiple layers, each of which contains multiple nodes where each node is computed as a linear combination of nodes in the previous layer, followed by a nonlinear transformation such as a sigmoid, a tanh, a softmax function. However, neural network has not been widely used until recently since it was difficult to train due to the significant computing time, its sensitivity to initialization and hyper-parameters, and other issues. Various treatments have been proposed including dropout, batch normalization, and alternative nonlinear functions such as a rectified linear unit, which successfully handled most of the existing issues. Beyond the traditional model, the neural network structure has evolved in various forms, leading to tremendous success in important applications. Largely responsible for this recent success is convolutional neural network (CNN), a type of neural network suited for real-world image classification tasks. Although convolutional neural networks have been originally proposed by LeCun et al. back in the early 1990s, demonstrating an outstanding performance in hand-written digit recognition, it was not widely used until 2012 when Krizhevsky et al. achieved a superior performance on image classification tasks in ImageNet challenge, using a deep architecture model of convolutional neural network. This propelled major research movement towards creating variants in architectures and improving algorithms for even higher performance. In just few years, much progress has been made to the point of approaching or even surpassing human abilities in various challenging tasks. While making significant achievements, the understanding of underlying processes in these models received less examination, and the need for tools and techniques for exploring and understanding the inner workings of these various models ensued. However, complicated deep learning structures are difficult to understand. Different types of layers such as convolution, pooling, and fully connected layers interact with each other, handling different parts of data characteristics. Furthermore, each layer has different sets of hyper-parameters to determine before training the model. Thus, such a model selection process including setting the number of layers and nodes, and hyper-parameter values has not been intuitive nor straightforward, leaving users with no idea about how to properly perform this process. In addition, the significant amount of time required to train a deep learning model has made the training process largely detached

You might have seen this famous xkcd comic before. The goof is based on the idea that any 3-year-old child can recognize a photo of a bird, but figuring out how to make a computer recognize objects has puzzled the very best computer scientists for over 50 years. In the last few years, we’ve finally found a good approach to object recognition using deep convolutional neural networks. That sounds like a a bunch of made up words from a William Gibson Sci-Fi novel, but the ideas are totally understandable if you break them down one by one.

So let’s do it — let’s write a program that can recognize Natural Calamity!

**Dataset**

#**Sample Data set images Figure 1**

We have dataset of Natural Calamity Namely Hurricane, Volcanic Eruption, Earthquake, Tornado. The dataset of images was collected using Google API. We collected dataset of 400 images equally distributed among 4 classes from google which are jpeg images and relevant to project All images had 3 channels, Red, Green and Blue and were 150x150 pixels large. **Figure 1** shows sample data from our collected dataset. There were many grayscale images that would most likely only introduce noise to our model, as such, we had to filter them out.

Additionally, many pictures did not clearly correspond to the class presented. There was lot of corrupted images while downloading from API where were filtered out at later stages before preprocessing starts.

**Preprocessing**

We preprocessed the images to increase the accuracy of our models. The technique that we used were data augmentation on the dataset to make a new dataset.

**Method**

**Parameters**

• Batch size: Gcd of training samples and

• Number of filters: [32,32,64]

• Size of filters: 3 × 3

• Number of epochs: 20

The final architecture retained can be described as follows:

• 3 × 3 Conv - ReLU - 2×2 Max-Pool with 32 filters

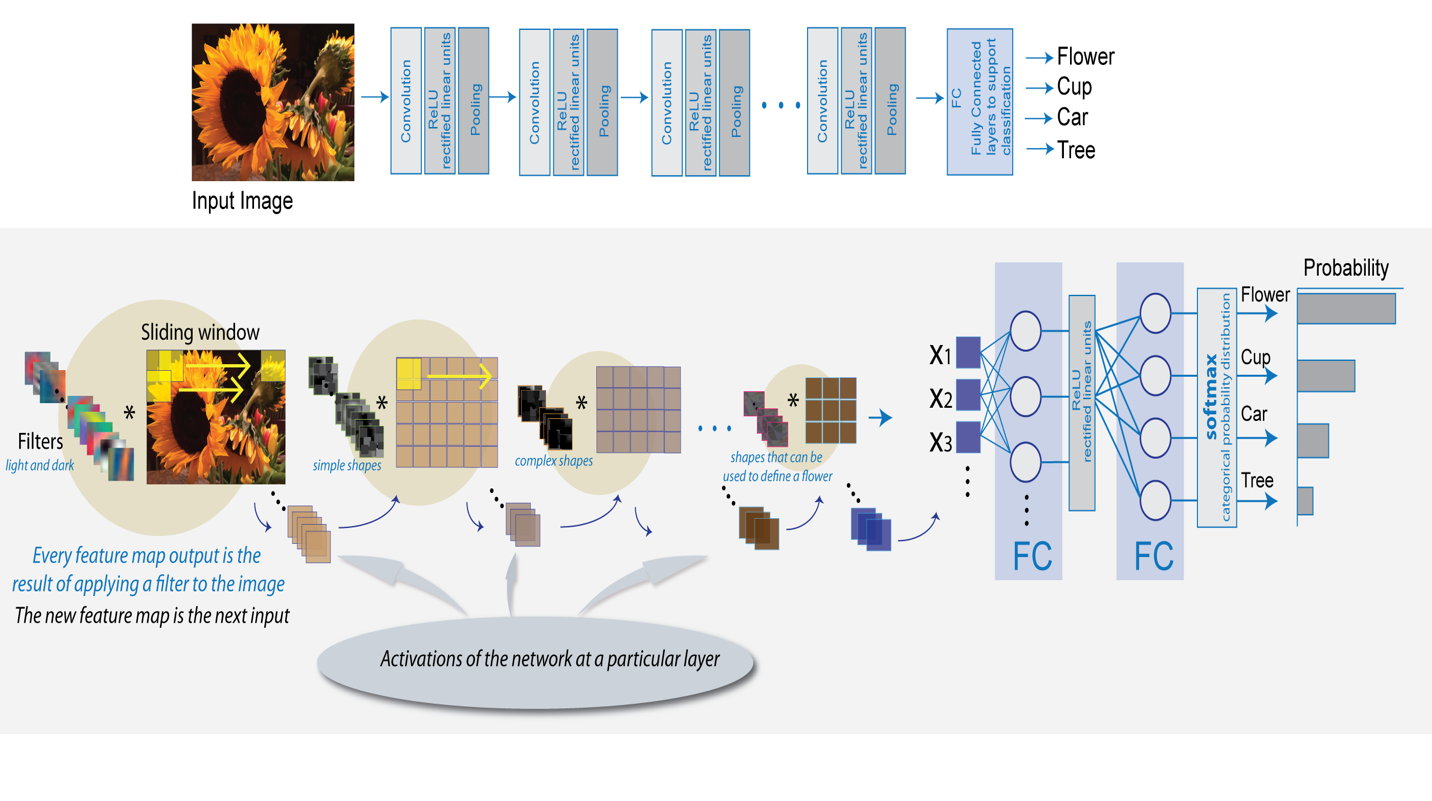
• 3 × 3 Conv - ReLU - 2×2 Max-Pool with 32 filters

• 3 × 3 Conv - ReLU - 2×2 Max-Pool with 64 filters

• FC layer to 4 class

**Architecture and Related Terms:**

A CNN is a neural network that typically contains several types of layers, one of which is a **convolutional layer**, as well as **pooling**, and **activation** layers.

****

#### Convolutional Layer

Imagine you have an image represented as a 5x5 matrix of values, and you take a 3x3 matrix and slide that 3x3 window around the image. At each position the 3x3 visits,you matrix multiply the values of your 3x3 window by the values in the image that are currently being covered by the window. This results in a single number the represents all the values in that window of the image.

The “window” that moves over the image is called a **kernel**. Kernels are typically square and 3x3 is a fairly common kernel size for small-ish images. The distance the window moves each time is called the **stride**. Additionally, of note, images are sometimes padded with zero’s around the perimeter when performing convolutions, which dampens the value of the convolutions around the edges of the image (the idea being typically the center of photos matter more).

The goal of a convolutional layer is **filtering.**As we move over an image we effective check for patterns in that section of the image. This works because of **filters,**stacks of weights represented as a vector, which are multiplied by the values outputted by the convolution. When training an image, these weights change, and so when it is time to evaluate an image, these weights return high values if it thinks it is seeing a pattern it has seen before. The combinations of high weights from various filters let the network predict the content of an image. This is why in CNN architecture diagrams, the convolution step is represented by a box, not by a rectangle

#### Pooling Layers

Pooling works very much like convoluting, where we take a **kernel** and move the kernel over the image, the only difference is the function that is applied to the kernel and the image window isn’t linear. **Max pooling** and **Average pooling** are the most common pooling functions. Max pooling takes the largest value from the window of the image currently covered by the kernel, while average pooling takes the average of all values in the window

#### Activation Layers

* Activation layers work exactly as in other neural networks; a value is passed through a function that squashes the value into a range. The most used activation function in CNNs is the relu (Rectified Linear Unit). There are a bunch of reason that people like relus, but a big one is because they are really cheap to perform, if the number is negative: zero, else: the number. Being cheap makes it faster to train networks. **Activation layers**squash the values into a range, typically [0,1] or [-1,1]

#### Fully Connected Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the [Softmax](http://cs231n.github.io/linear-classify/#softmax) as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one

#### Other ConvNet Architectures

Convolutional Neural Networks have been around since early 1990s. We discussed the LeNet above whichwas one of the very first convolutional neural networks. Some other influential architectures are listed below .

* **1990s to 2012:** In the years from late 1990s to early 2010s convolutional neural network were in incubation. As more and more data and computing power became available, tasks that convolutional neural networks could tackle became more and more interesting.
* **AlexNet (2012) –**In 2012, Alex Krizhevsky (and others) released [AlexNet](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.
* **ZF Net (2013) –** The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the [ZFNet](http://arxiv.org/abs/1311.2901) (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters.
* **GoogLeNet (2014) –**The ILSVRC 2014 winner was a Convolutional Network from [Szegedy et al.](http://arxiv.org/abs/1409.4842) from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
* **VGGNet (2014) –** The runner-up in ILSVRC 2014 was the network that became known as the [VGGNet](http://www.robots.ox.ac.uk/~vgg/research/very_deep/). Its main contribution was in showing that the depth of the network (number of layers) is a critical component for good performance.
* **ResNets (2015) –**[Residual Network](http://arxiv.org/abs/1512.03385) developed by Kaiming He (and others) was the winner of ILSVRC 2015. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 2016).
* **DenseNet (August 2016) –**Recently published by Gao Huang (and others), the [Densely Connected Convolutional Network](http://arxiv.org/abs/1608.06993) has each layer directly connected to every other layer in a feed-forward fashion. The DenseNet has been shown to obtain significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks..

**Approach**

We used TensorFlow along with other python packages such as Keras, pandas, numpy, matplotlib, etc to code all the experiments in this project. We also used Django web framework to classify object through web application. We applied the following models to our dataset to classify images into their correct categories:

1)VGGNet

2) Three Layer ConvNet

3)Five Layer ConvNet

**VGGNet**

VGGNet uses very small convolution filters (3x3), which allows the depth to be increased with less overhead than if it used larger filters [8]. The VGGNet consists of 8 sections. The first 5 sections consist of two pairs of convolution layers and a ReLU activation, which are followed by max-pooling. The last 3 sections consist of fully-connected layers. Max-pooling is 4x4 with a stride of 4 in the first section and 2x2 with a stride of 2 in the other sections. The convolution layer has 3x3 filters throughout the VGGNet, while the number of filters varies per section. The number of filters are 64, 128, 256, 512, and 512 for the first five sections respectively. The number of neurons in the fully-connected layers are 4096, 10, and 4 respectively for the last three sections. For training, SGD+Adam optimization is used along with dropout.

**Three Layer ConvNet**

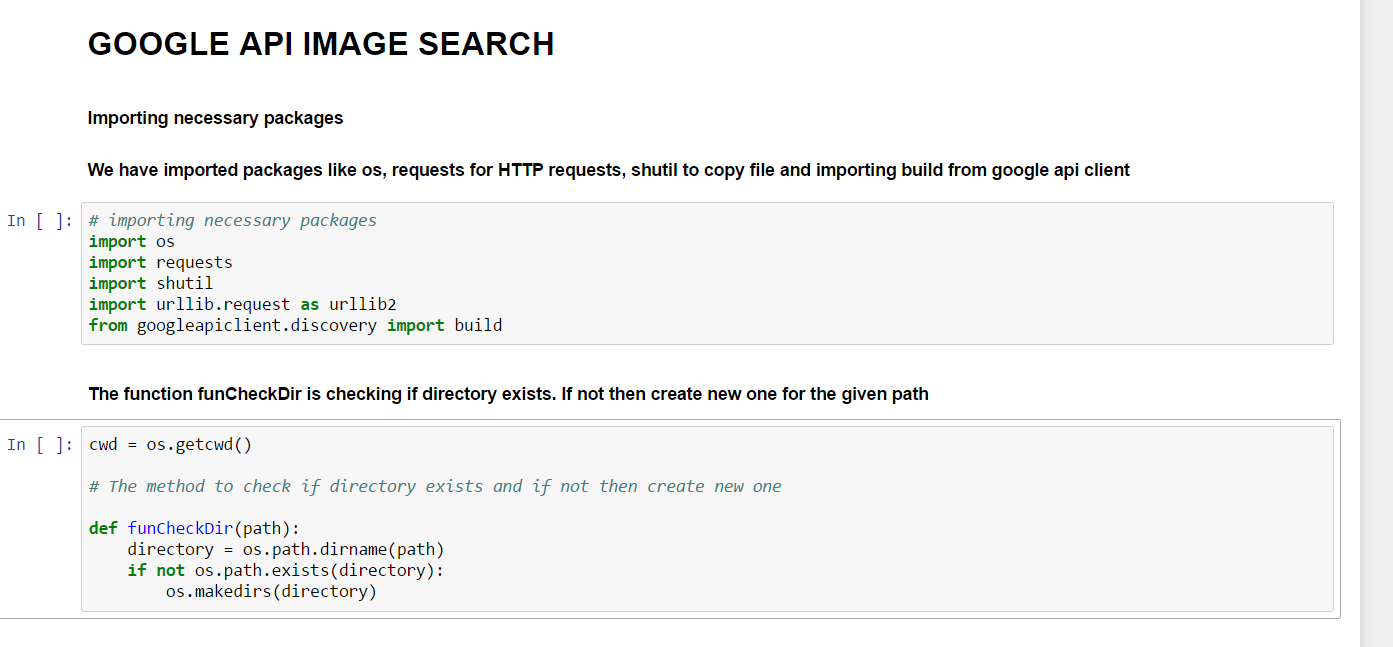
The architecture of the 3-layer ConvNet consists of three sections. The first section consists of a convolutional layer, followed by a ReLU activation, and ending with maxpooling. The second section is the same as the first. The third section consists of a fully-connected layer, followed by a ReLU activation, and ending with a linear affine. The convolution layers use 32 3x3 filters with a stride of 1. Maxpooling is 2x2 (which essentially halves the planar dimensions) with a stride of 1. For training, SGD optimization is used, and dropout is used for regularization. Adam is the state-of-the-art gradient update rule for ConvNets. It combines elements from RMSProp and momentum update. Dropout is a regularization technique that helps prevent ConvNets from overfitting. The idea that during each a training step, a random group of neurons are disabled, which helps prevent neurons from co-adapting (i.e. developing an overly strong dependence on one another). The 3-layer ConvNet takes in a raw image as a 150x150x3 dimensional array and classifies the input image to one particular tag.

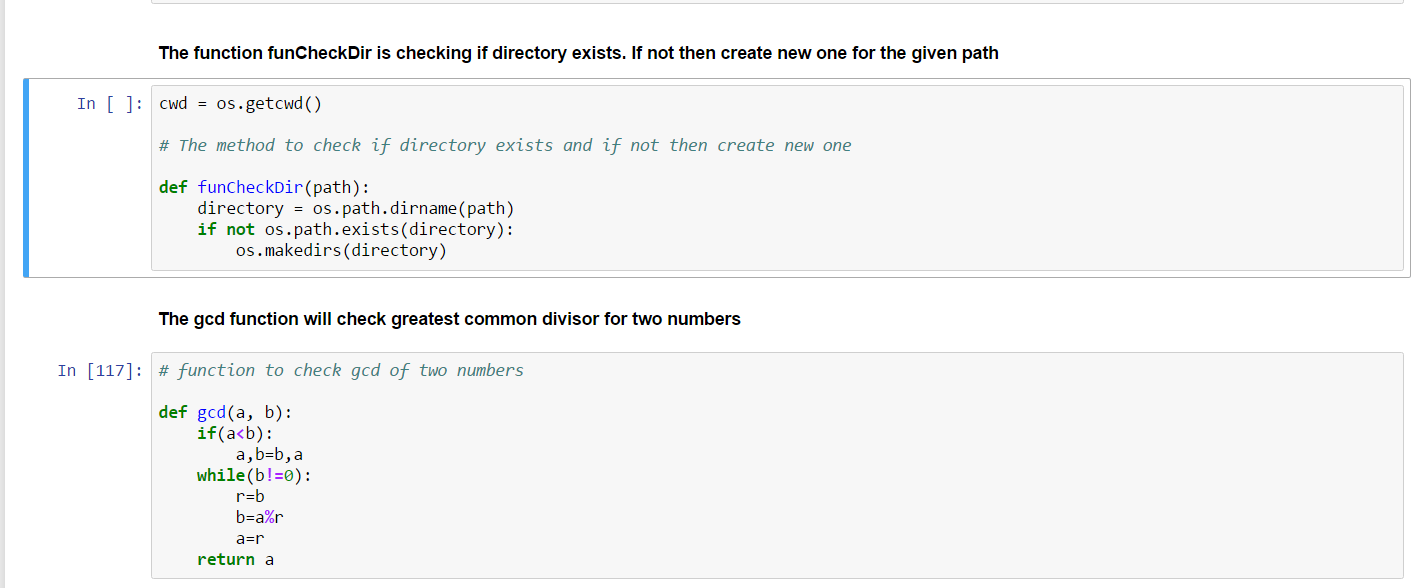
**Five** **Layer ConvNet**

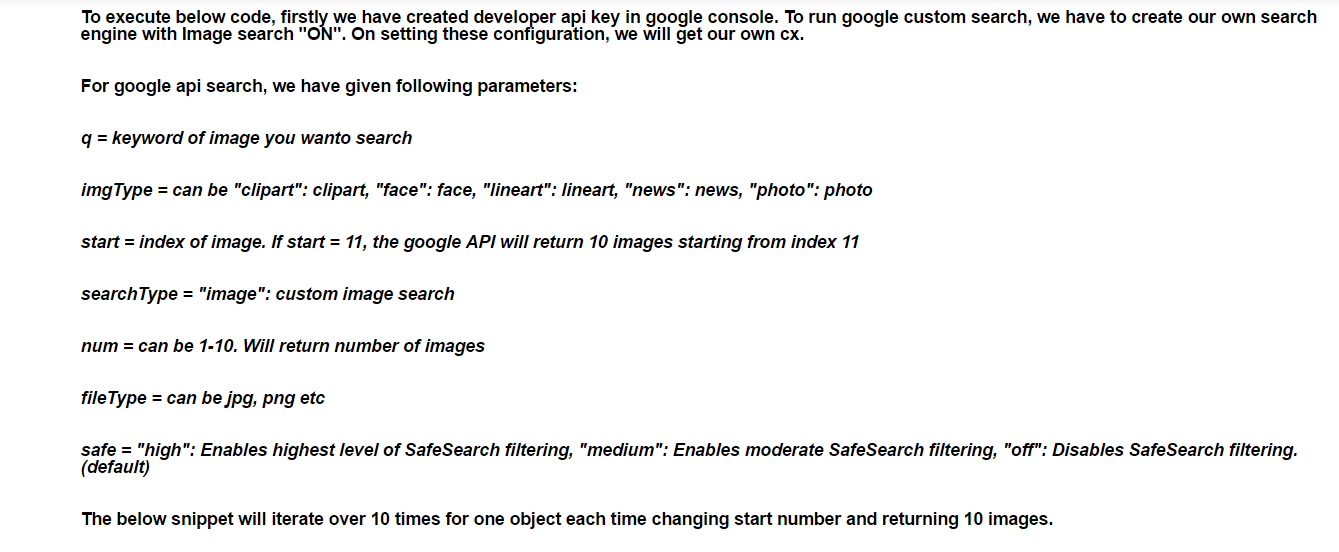
The architecture for our 5-layer ConvNet is the similar to the 3-layer ConvNet, except there are two more [conv - relu - pool] layers appended. The parameters for the convolutional layer, max-pooling, and fully-connected layer are the same, and SGD with Adam optimization and dropout are used as well.

**CODE FLOW**

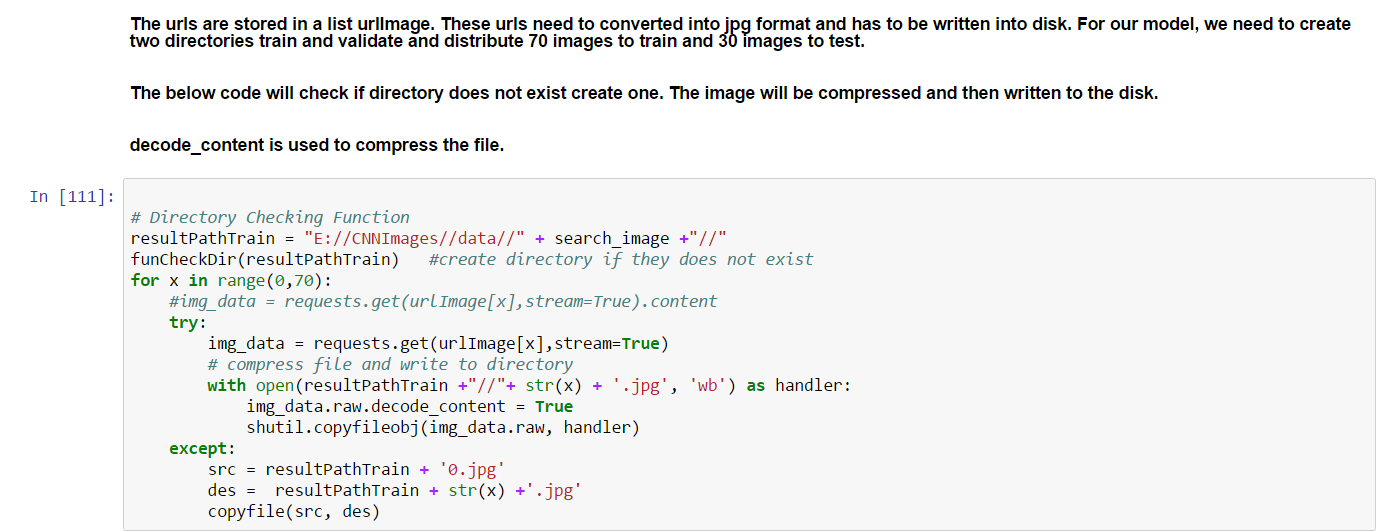
**STEPS**

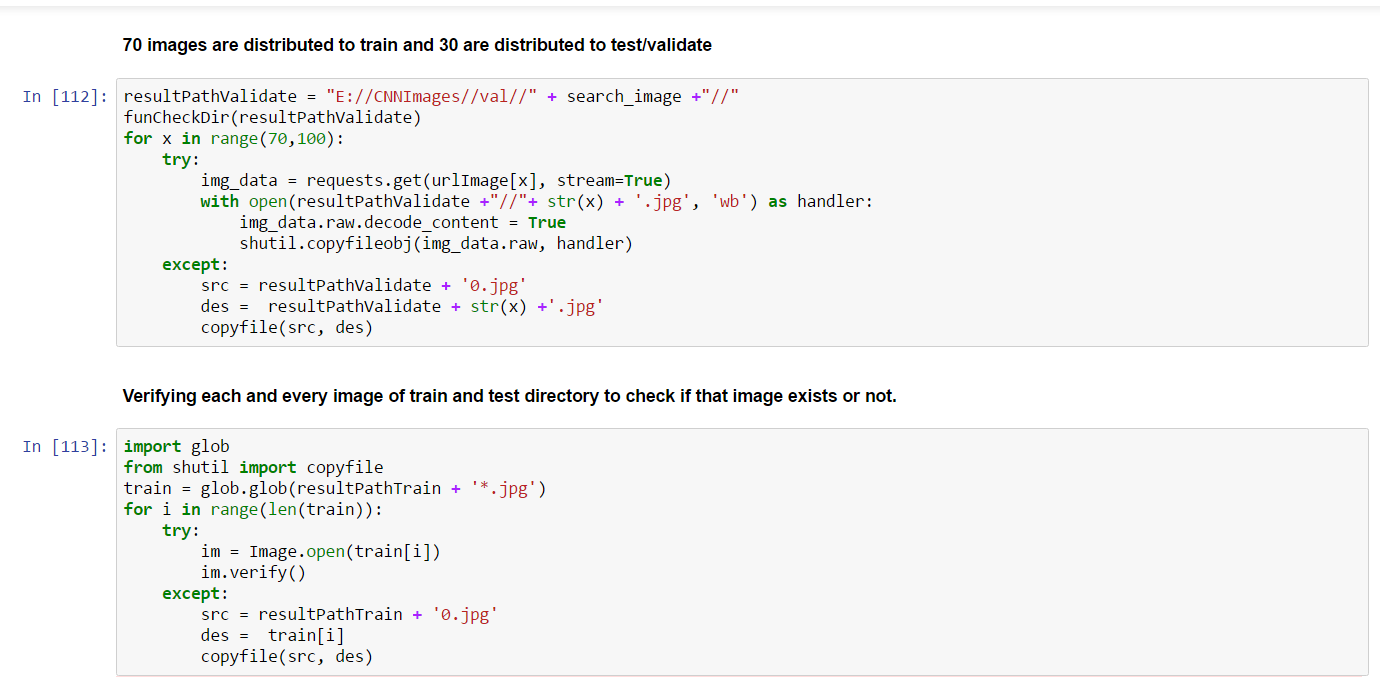


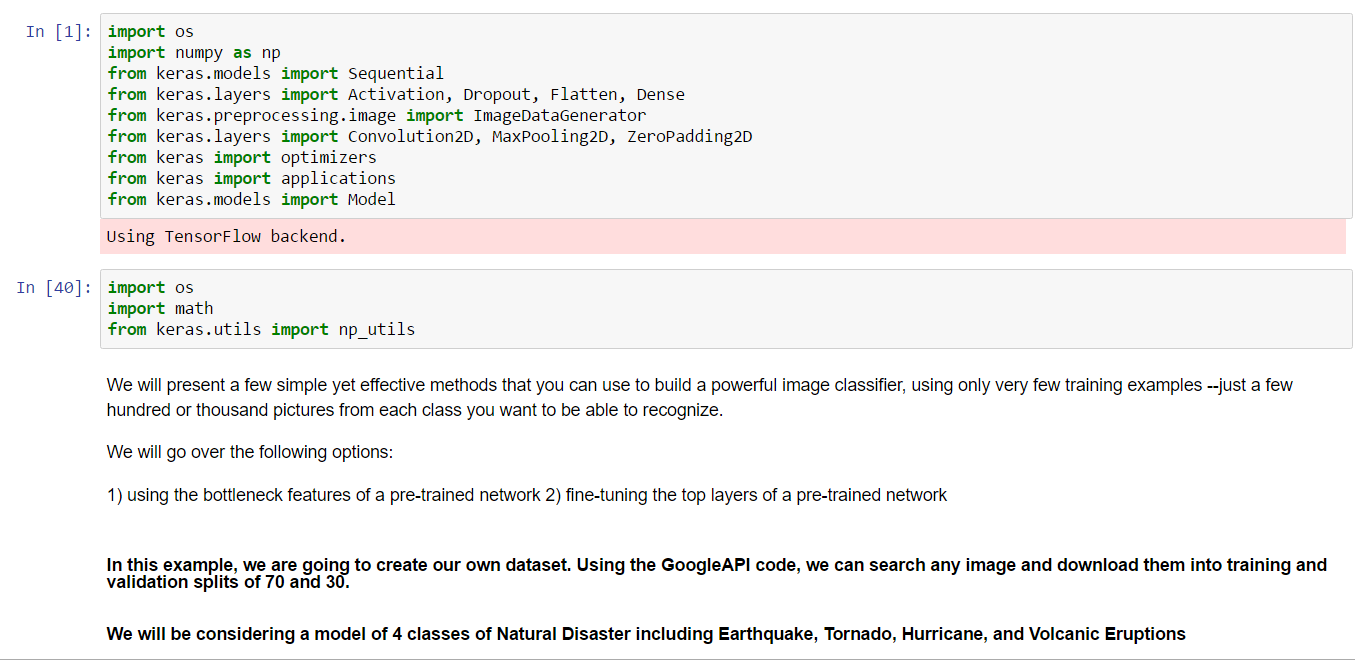






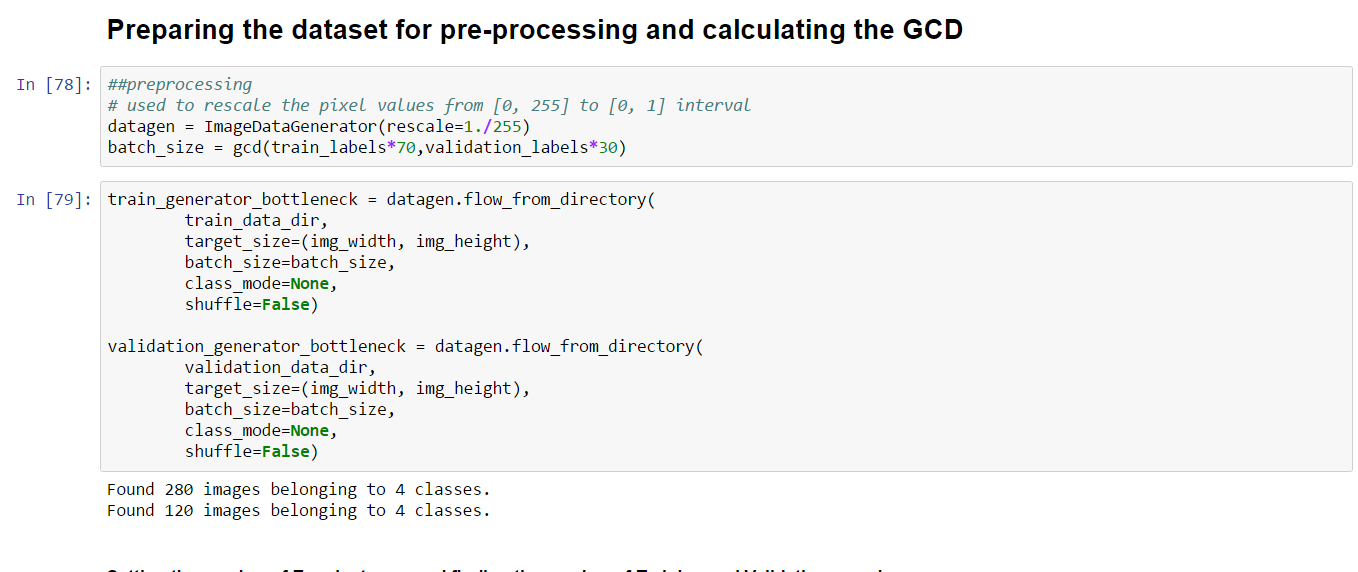






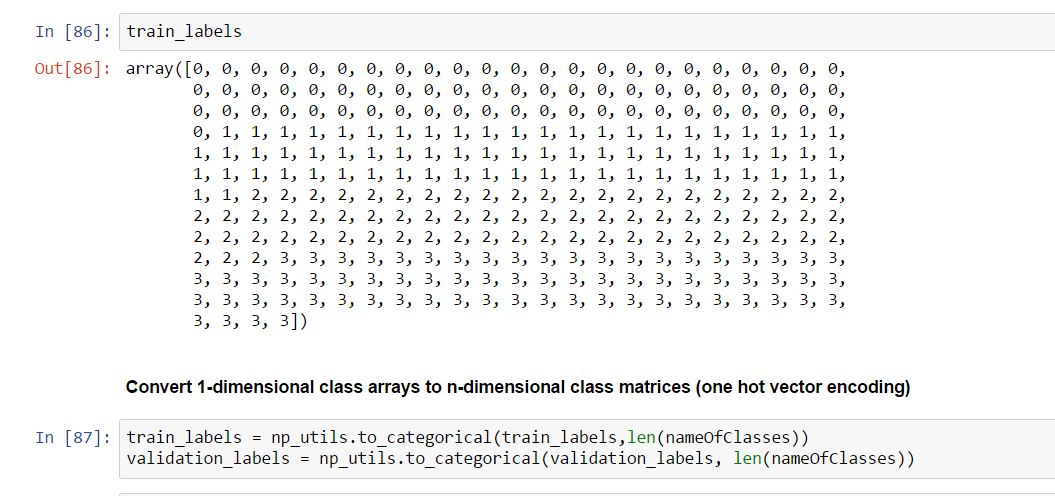
# 

# 

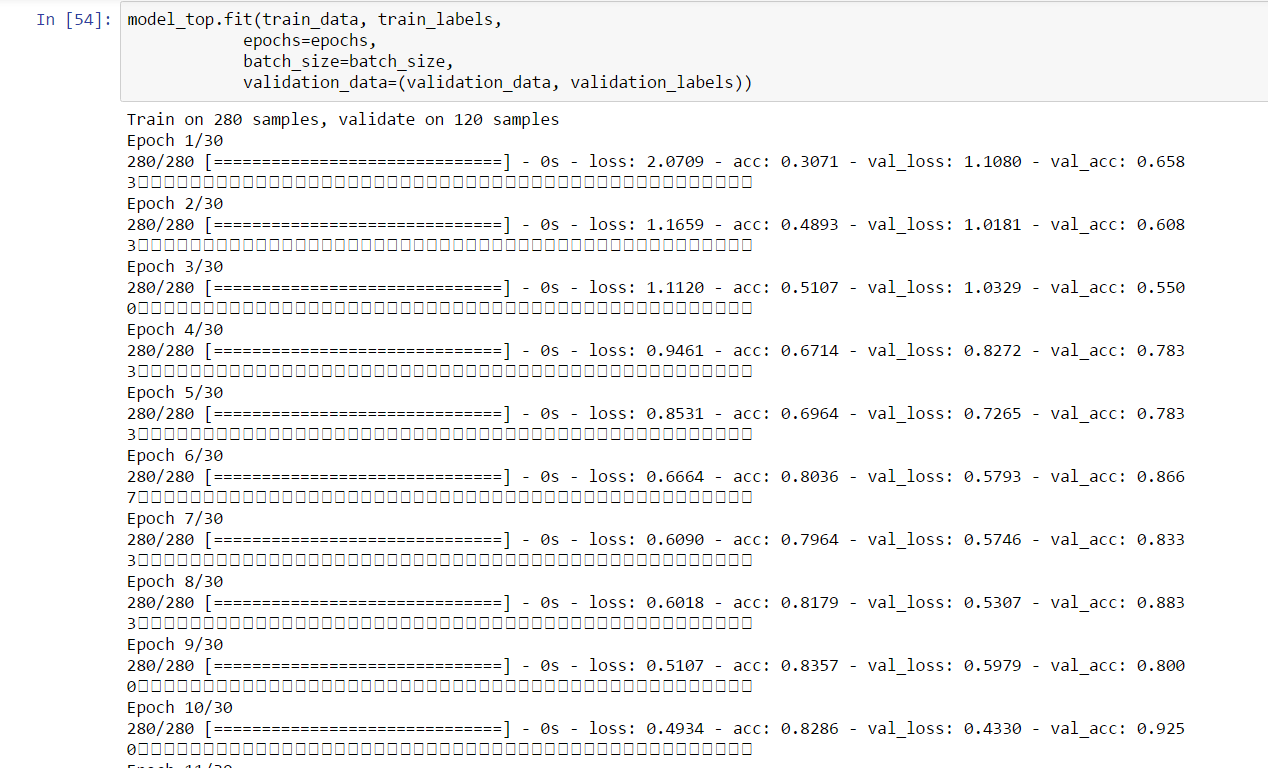


# 

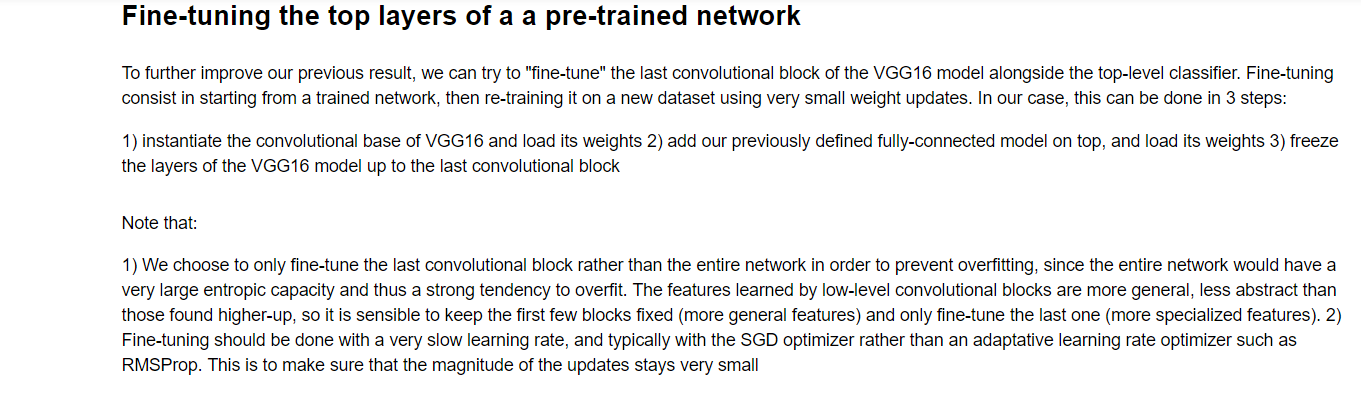
# 

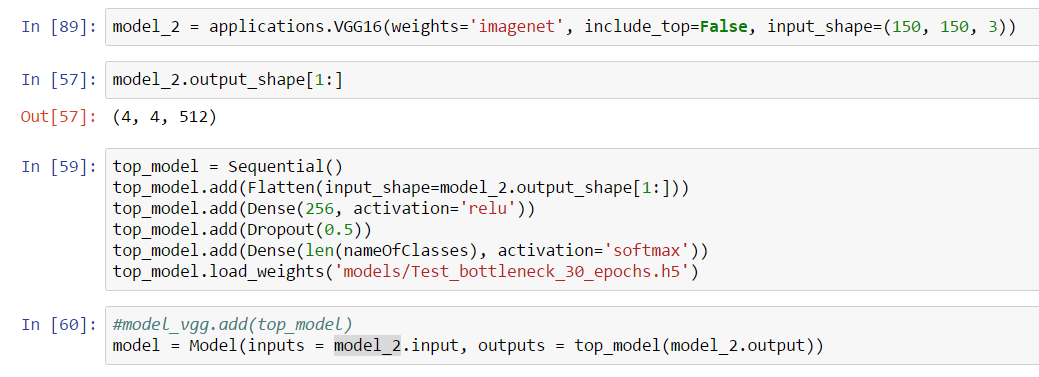


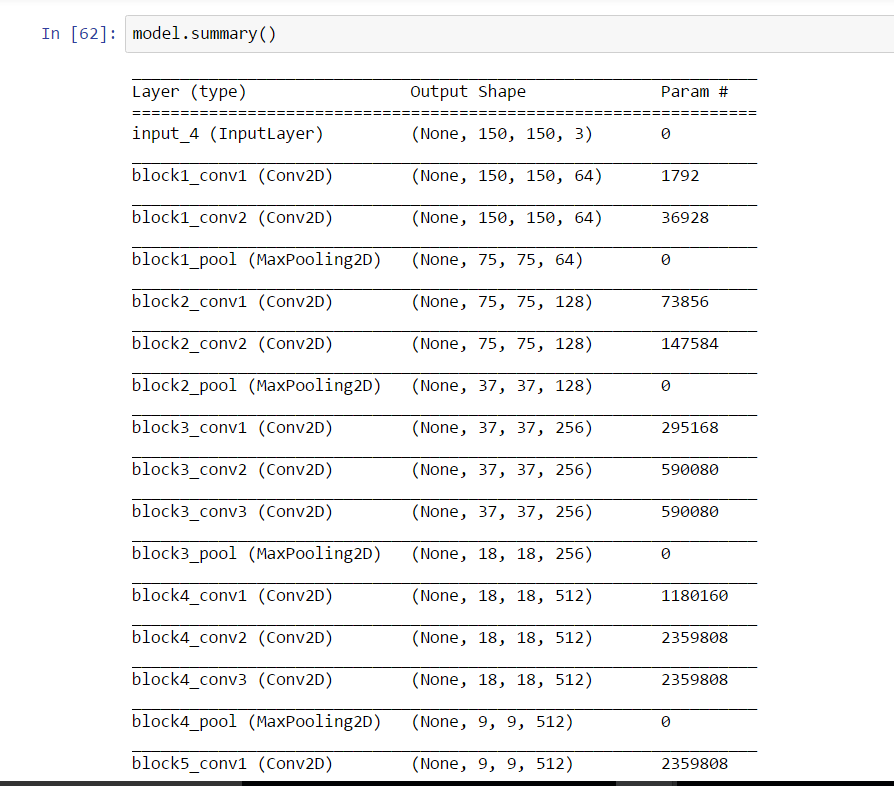


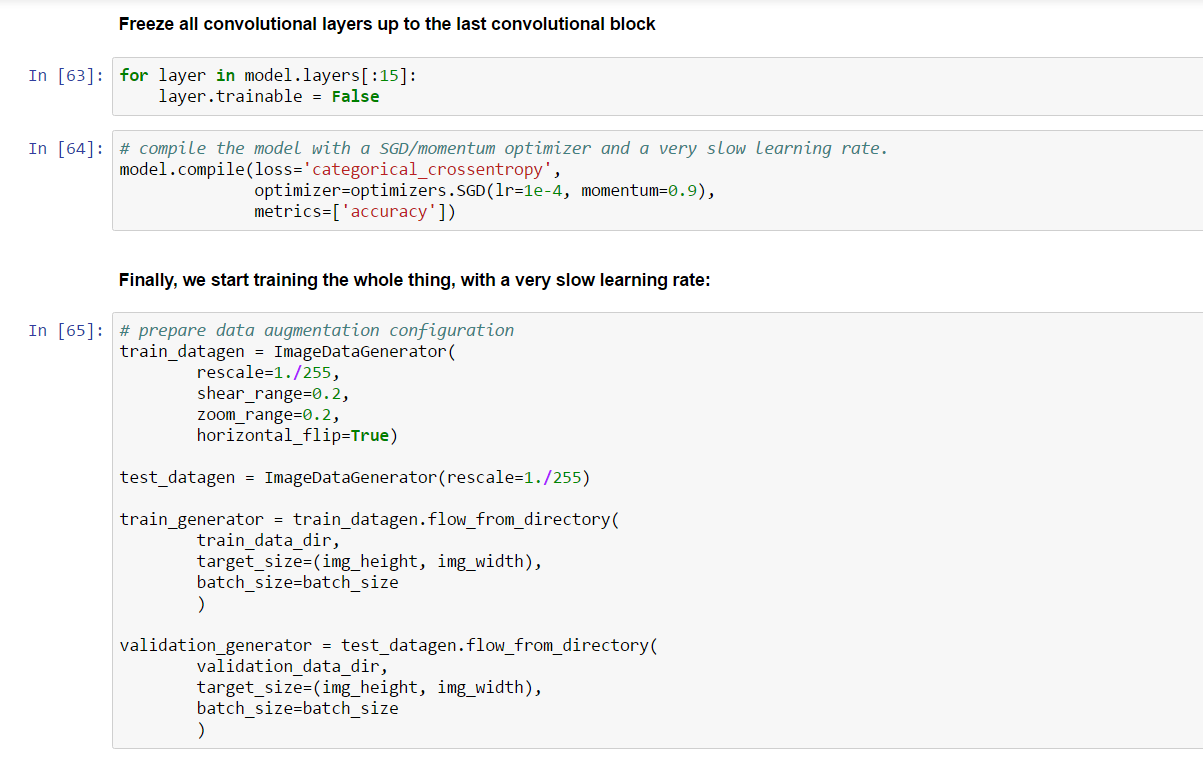


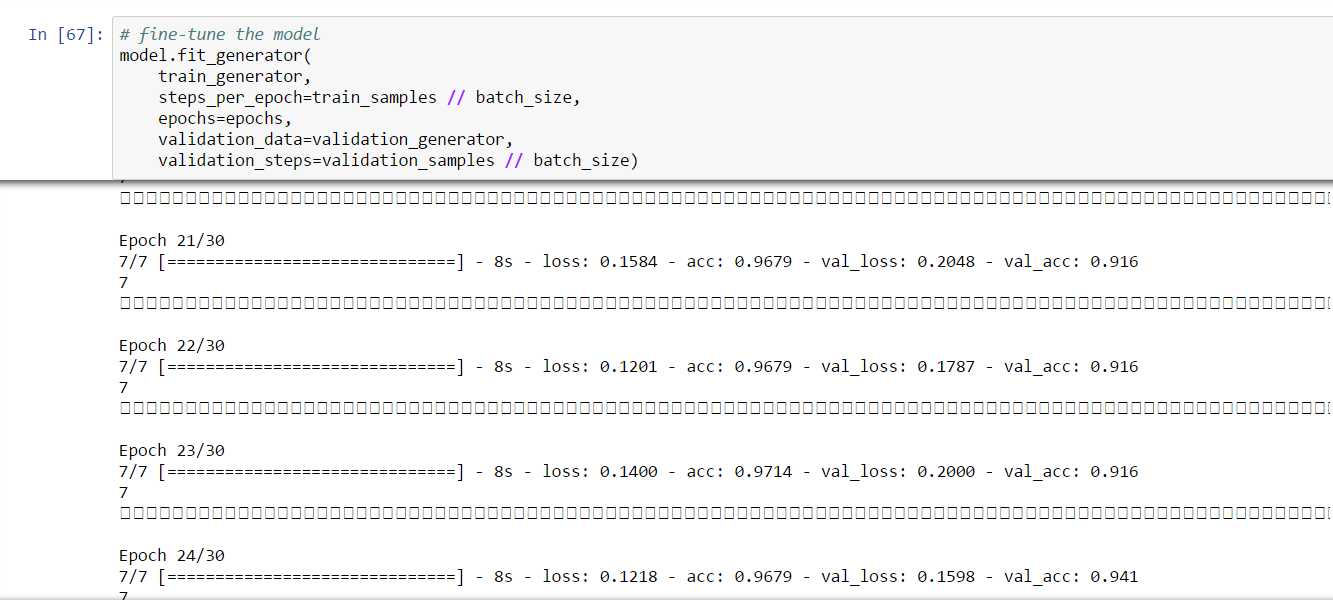


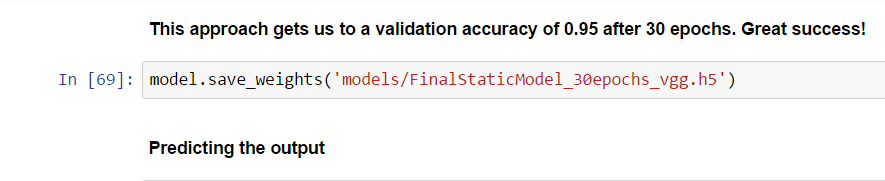


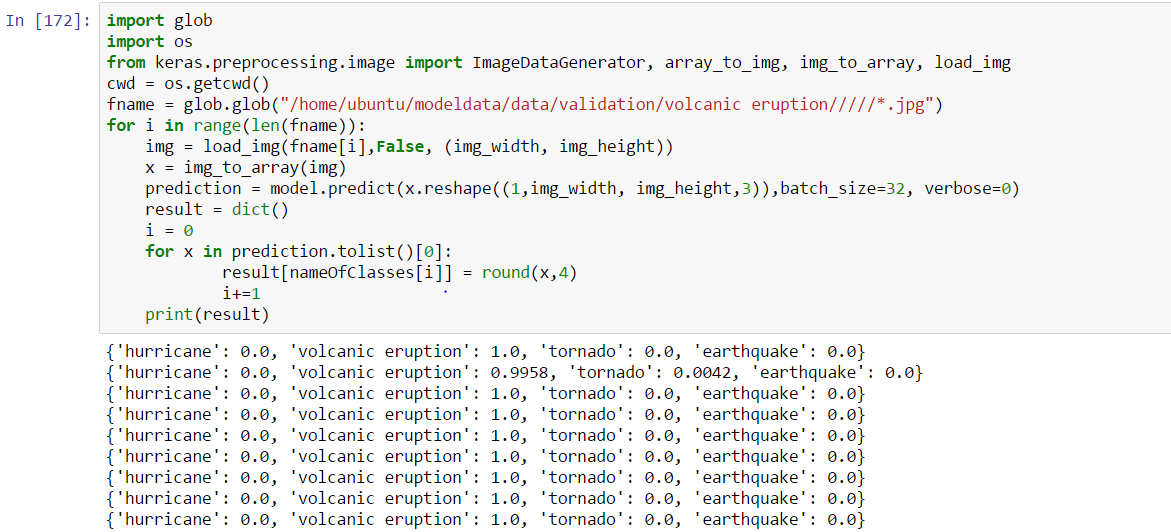






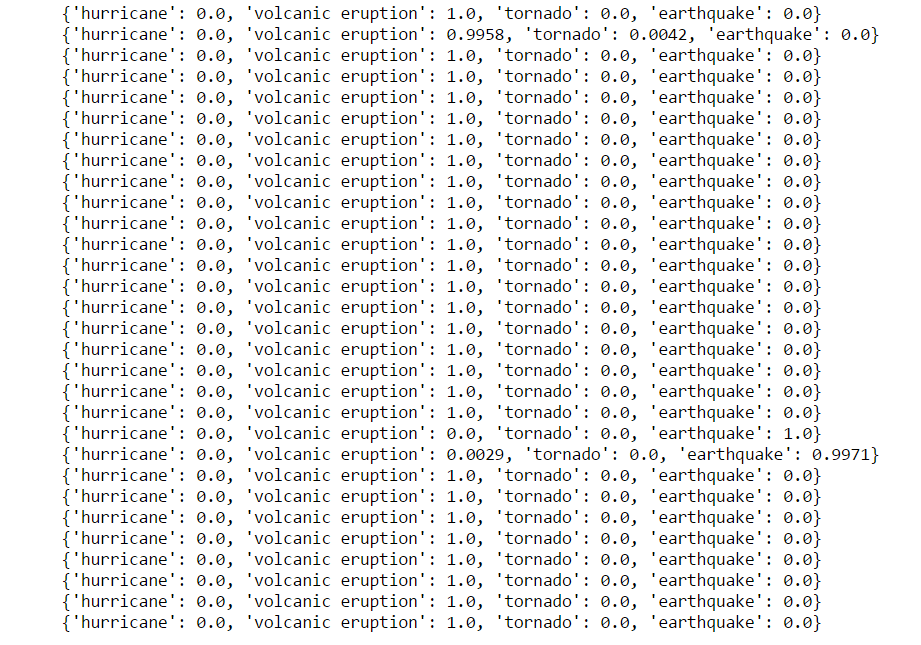






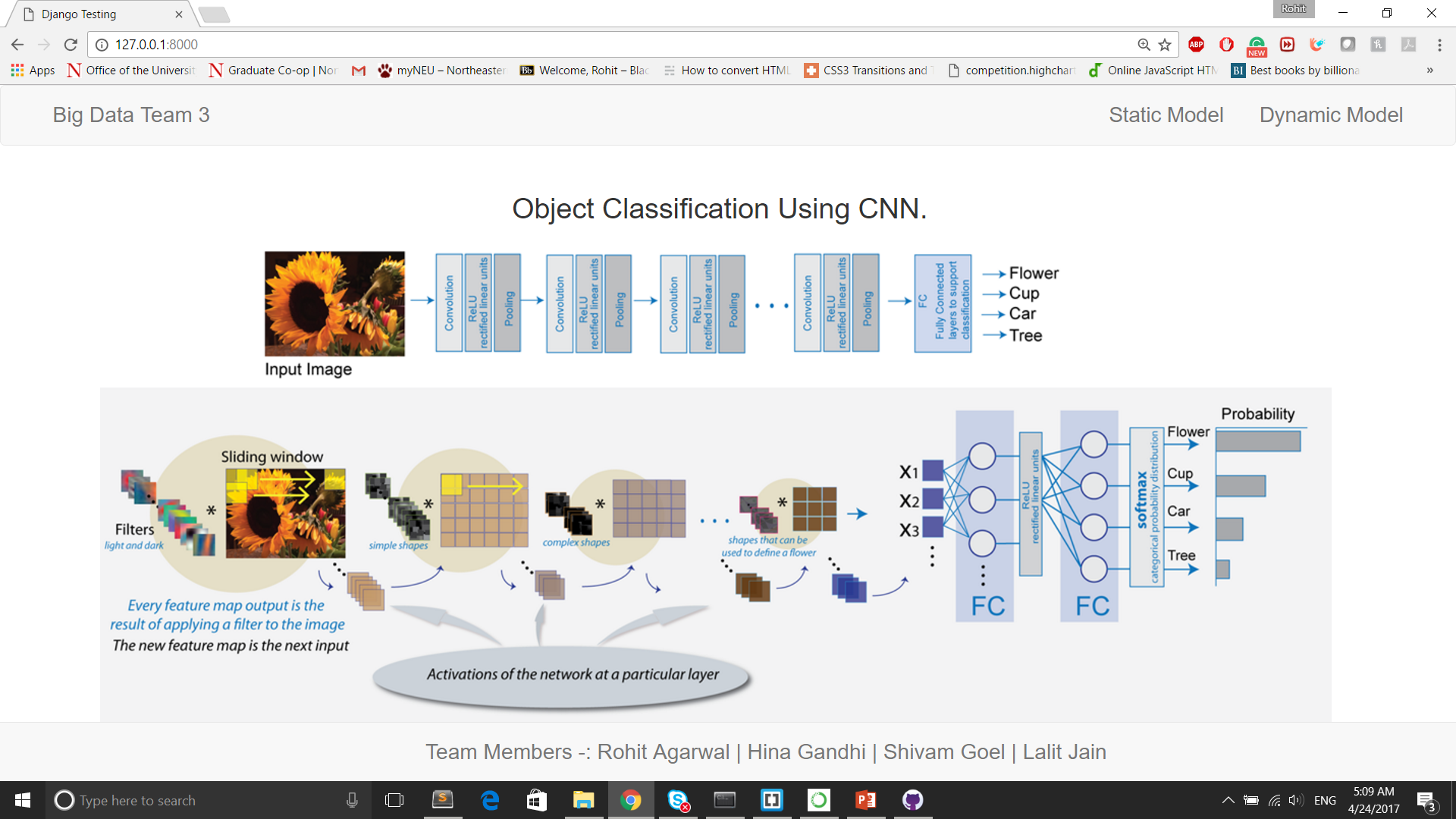
**VALIDATION:**

There are 4 classes earthquake, hurricane, tornado and volcanic eruption. When given the folder of validation of volcanic eruption for validation to the code, the below output was predicted: the last column was 1 predicting that the image is of volcanic eruption. The output will show probability of each class. This will help in telling the probability of each class if there is an image with more than one class.



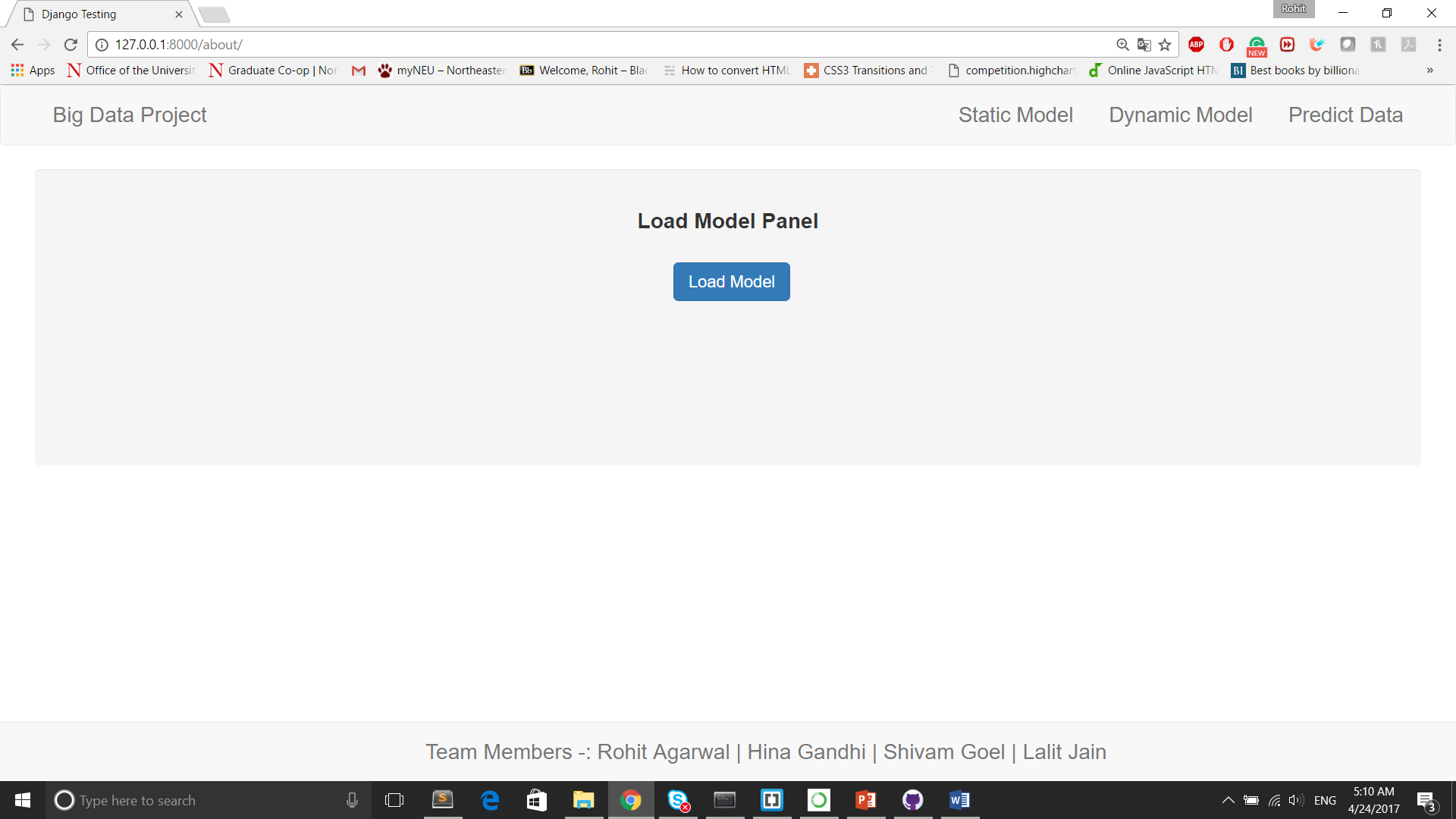
**UI Display:**

**The main page is shown below:**

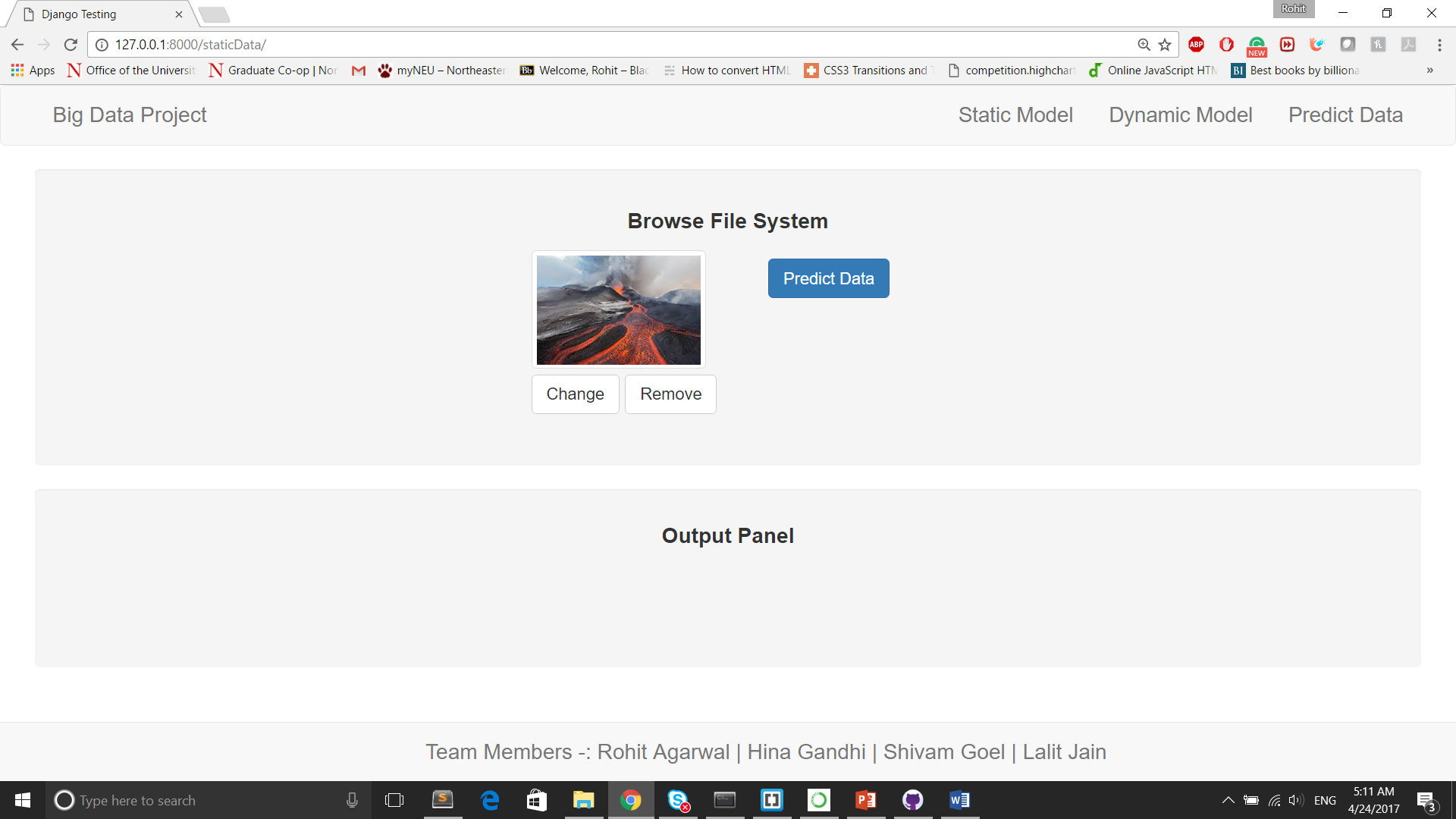


**There are two models: Static and Dynamic**

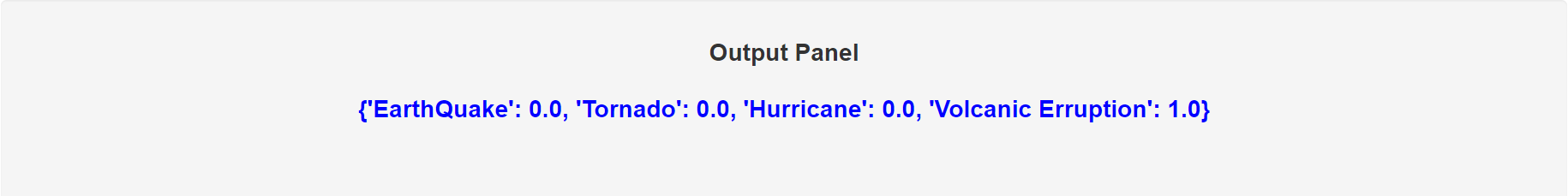
**In static, we have trained 4 classes and have selected any image and checked if it is volcanic eruption or earthquake or tornado or hurricane:**



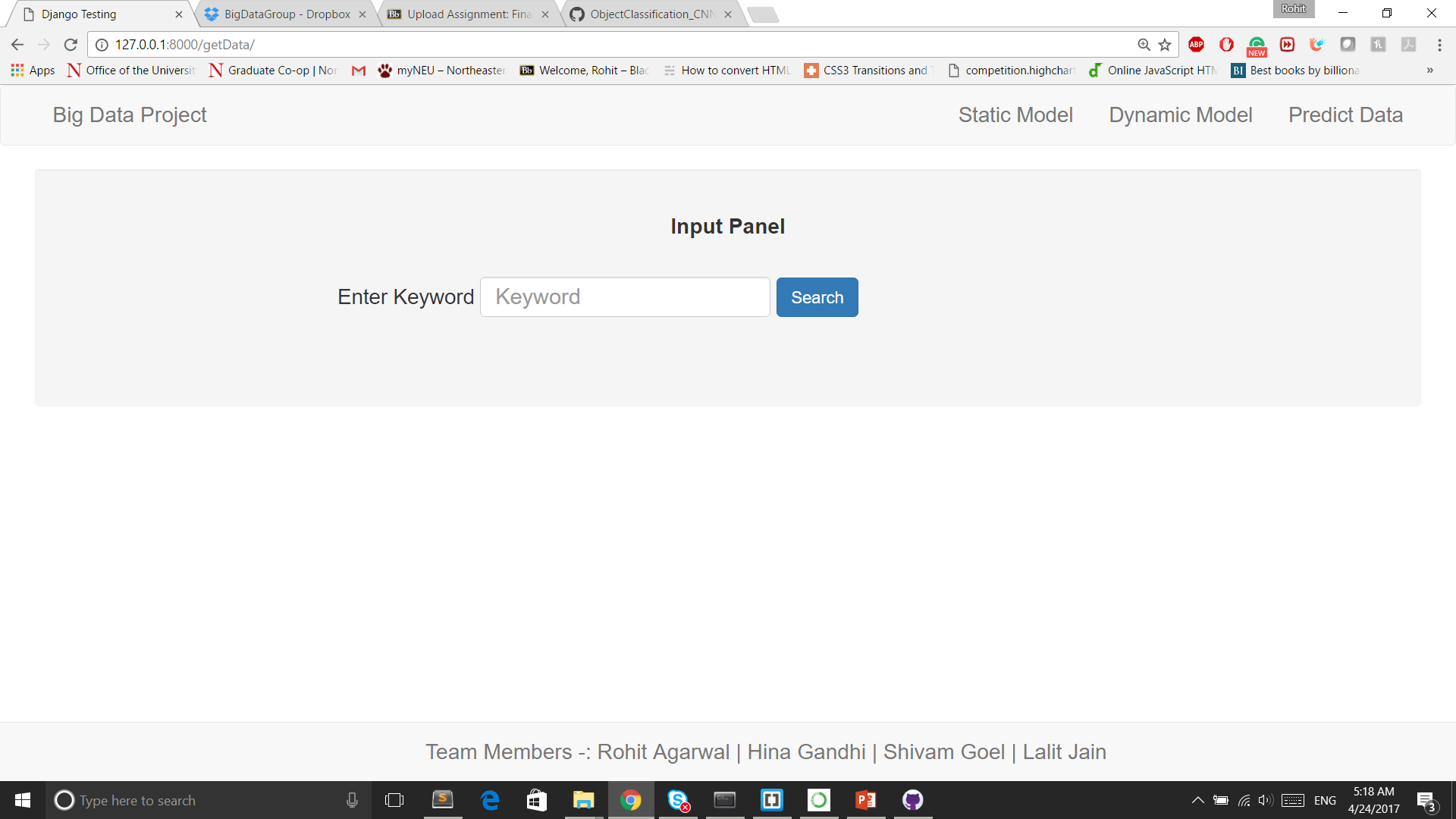
**Predict the output:**



**The output result of predicted output:**



**For Dynamic model we will enter the search query as shown below:**

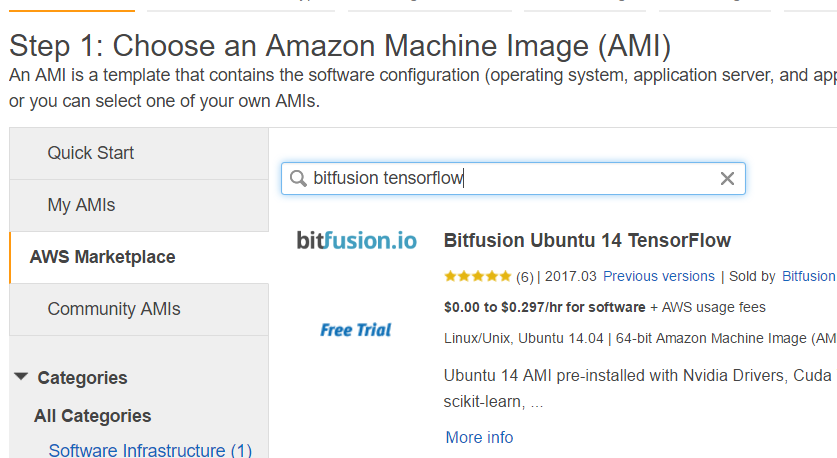


**Similarly we will train our model for all the classes including the newly added class. Predict model and get result on screen.**

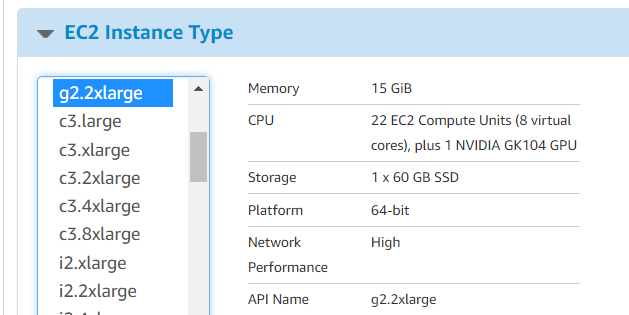
**Creating and Connecting to the GPU instance in Amazon**

Go to Amazon AWS and choose an instance from the Amazon Marketplace and look for Bitfusion Ubuntu 14 Tensorflow image:

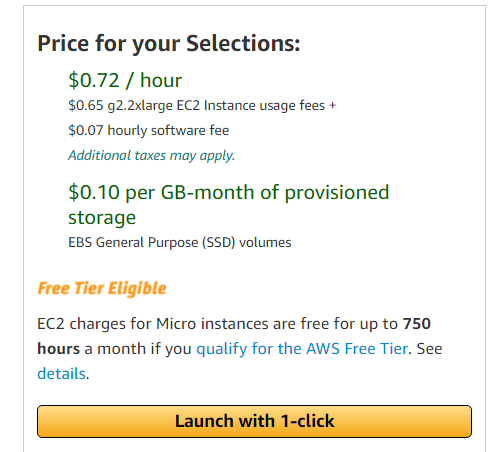
AMIDetails: <https://aws.amazon.com/marketplace/pp/B01EYKBEQ0?qid=1492926564428&sr=0-1&ref_=srh_res_product_title#product-description>

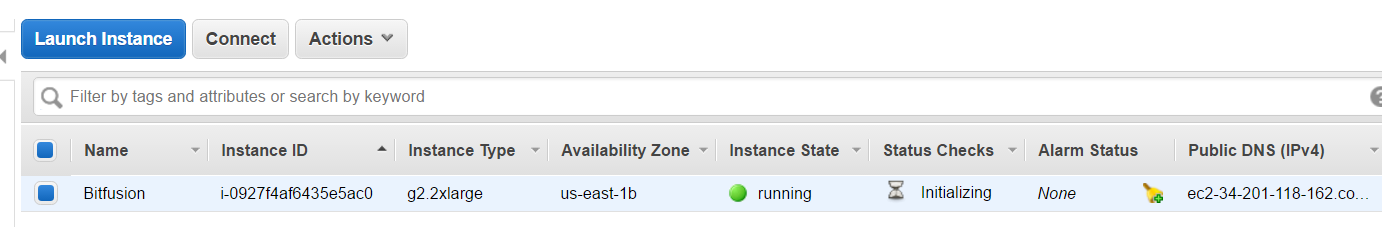


1.Go to the Bitfusion page from the image and select the EC2 Instance type as desired

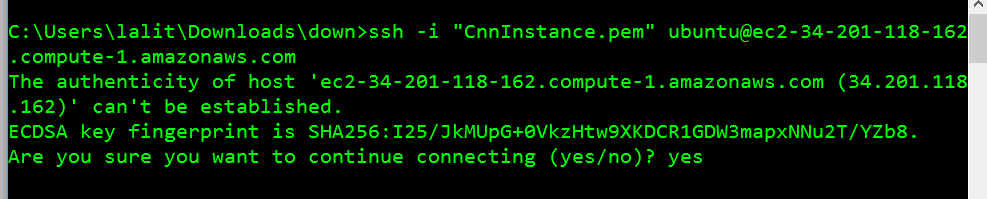


2.Click Launch with 1-click and you can see the instance up and running in your AWS management console in couple of minutes. Wait till the status check is completed

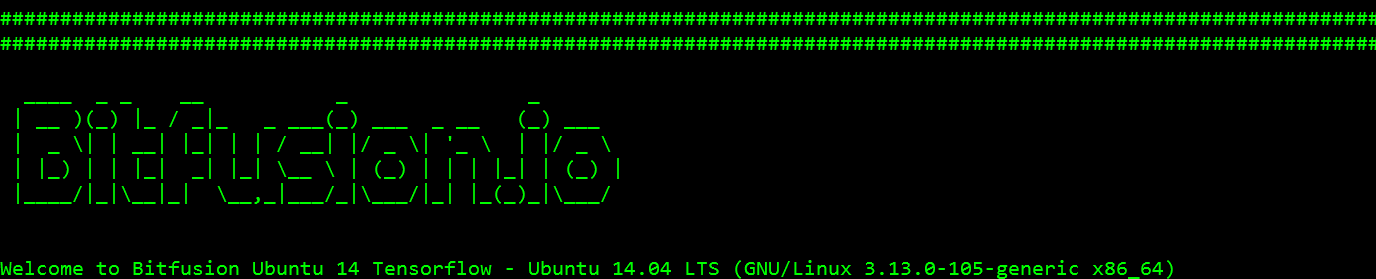




3.Ssh into machine

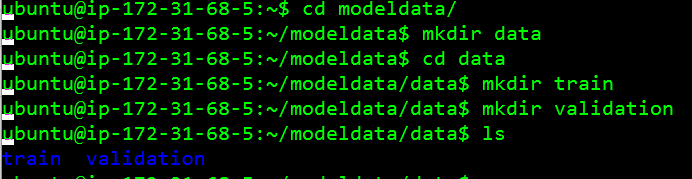


4.The below screenshot confirms the VM we want to be.

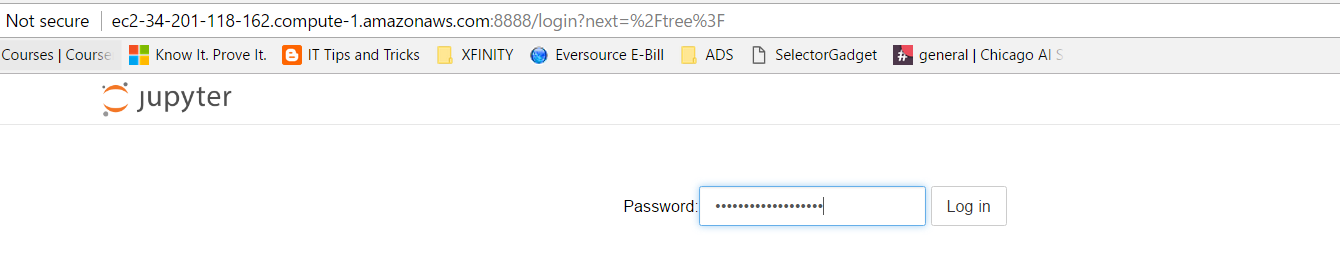


5.Run sudo-apt get update to update python binaries. Creating appropriate directories to set the machine learning environment.



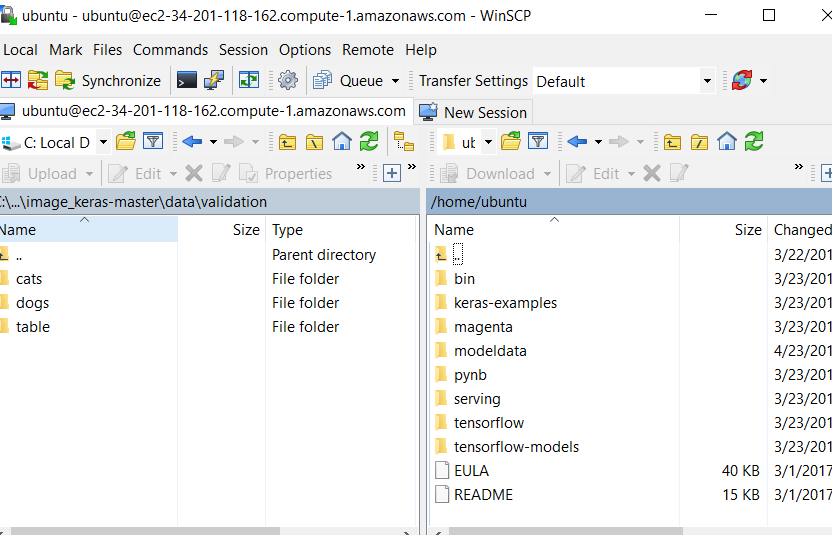


6.After installing python and Jupyter notebook, we can access jupyter notebook using our ubuntu VM i.e. ec2-34-201-118-162.compute-1.amazonaws.com:8888 on our browser.

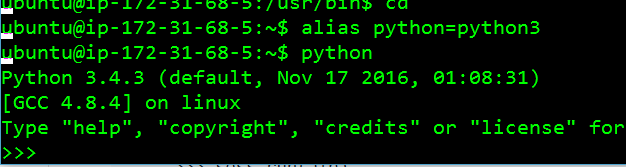




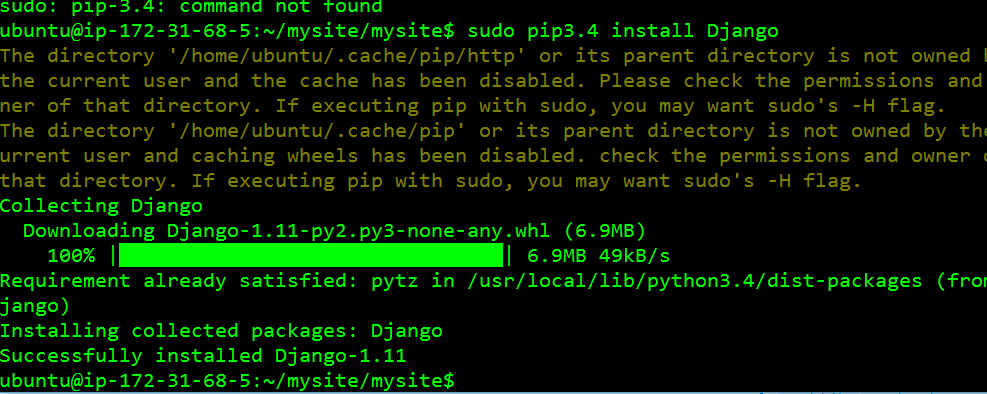
7.You can view Ubuntu machine directories in WinSCP. It is easy to move files locally to Ubuntu machine using WinSCP.



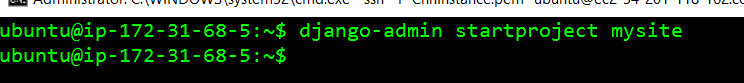
8.In our VM, we have both python2 and python3. We are using python 3 but default python for Django was python 2 so aliasing python to python3.



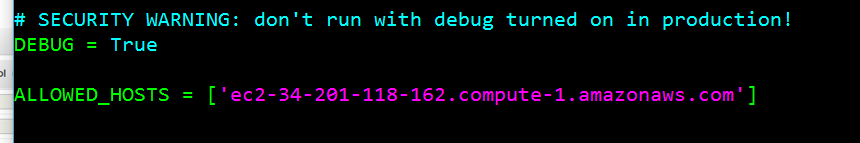
9.Installing Django using python 3



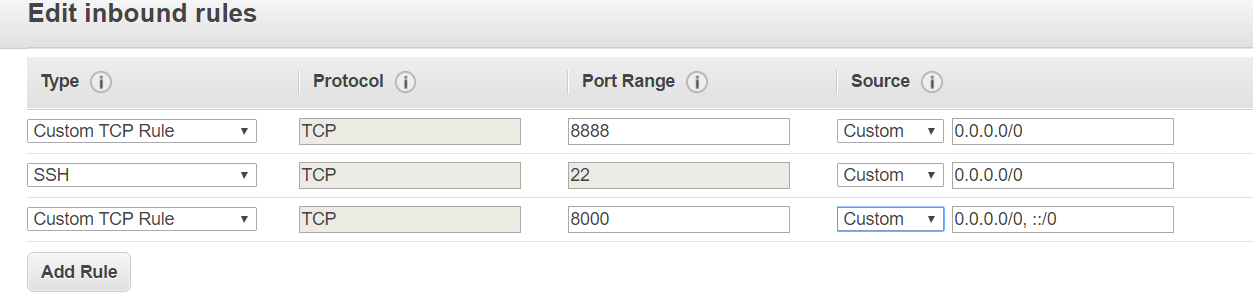
10.Creating our own website “mysite”.



11.Ubuntu machine was not allowed to run Django server. To allow it some configuration was done in settings.py where the Ubuntu machine’s DNS name was added to ALLOWED\_HOSTS.

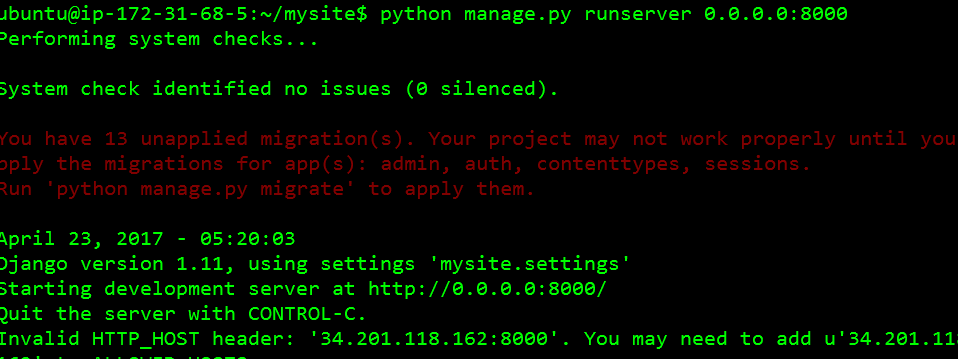


12.The inbound rules are changed where TCP added for ports 8000 and 8888

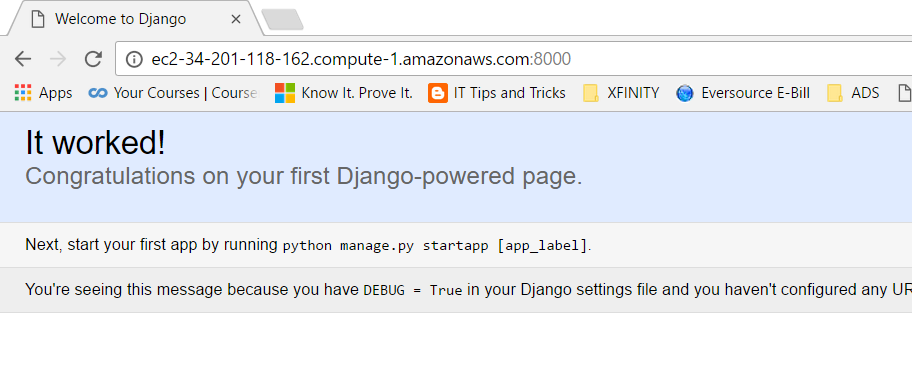


13.Running Django server using following command:

python manage.py runserver 0.0.0.0:8000



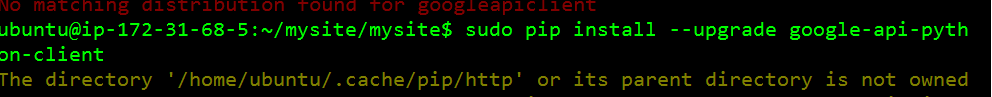
14.The server starts running without errors.



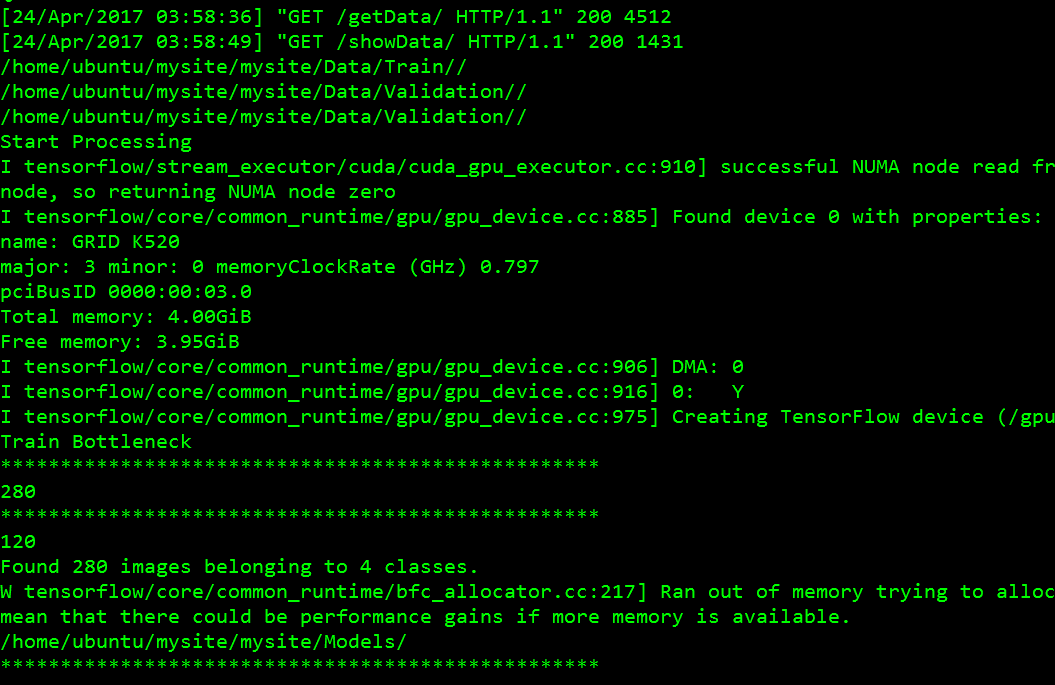
15.We need to import necessary packages in our Ubuntu machine so that python code works properly using following code:

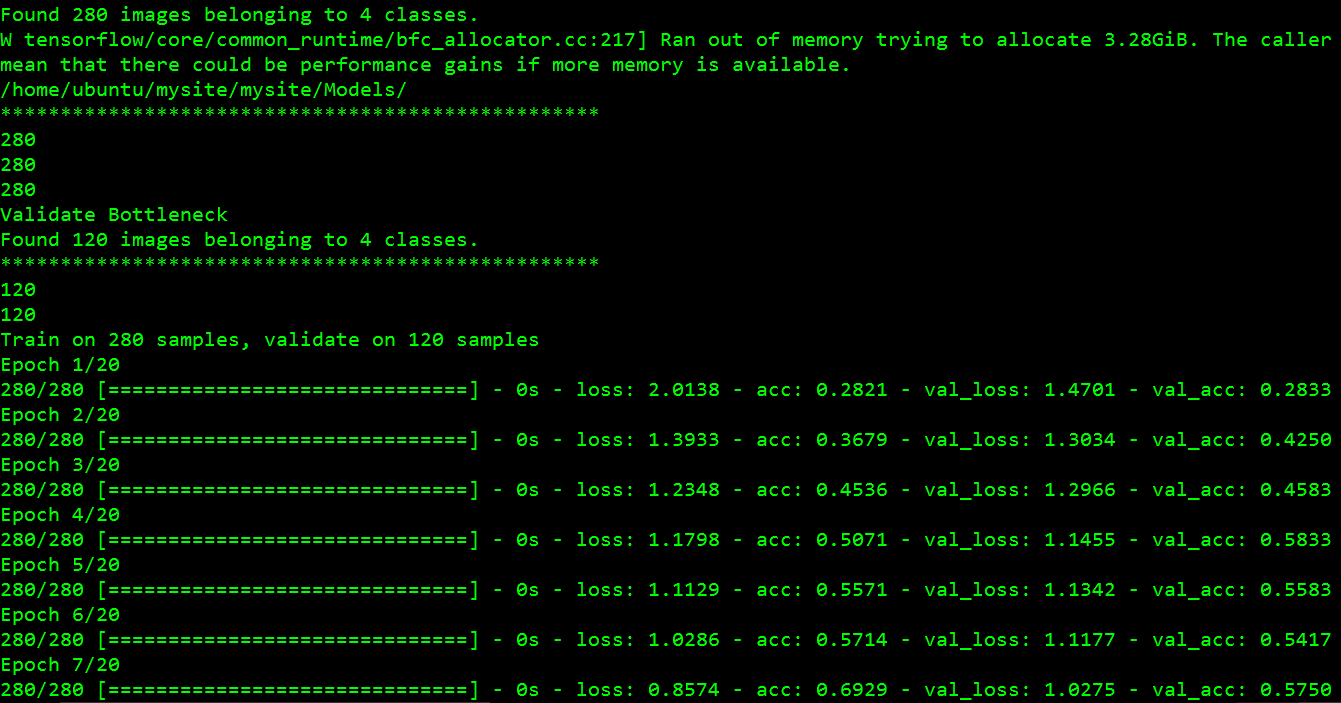
Sudo pip install –upgrade google-api-python-client

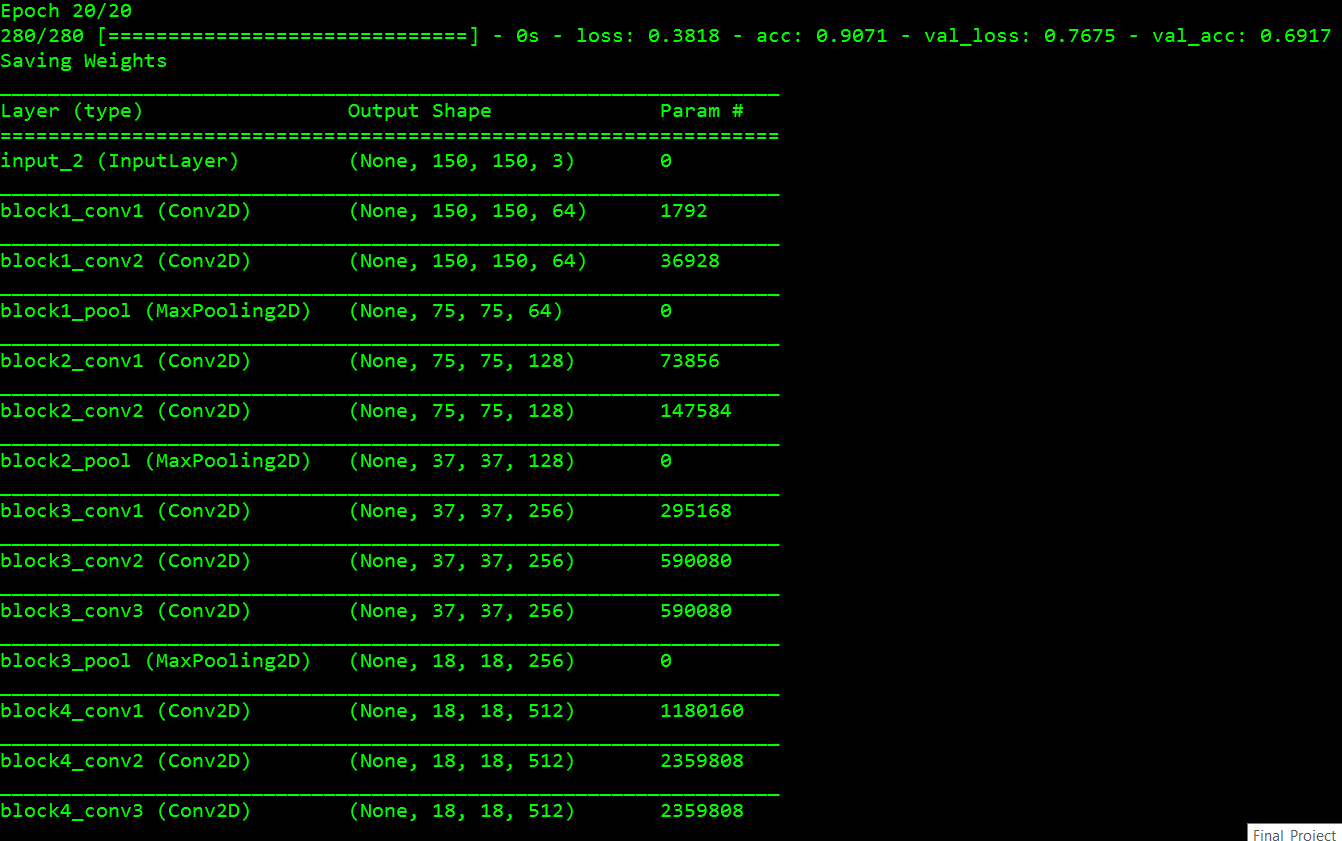
Similarly other packages were also installed

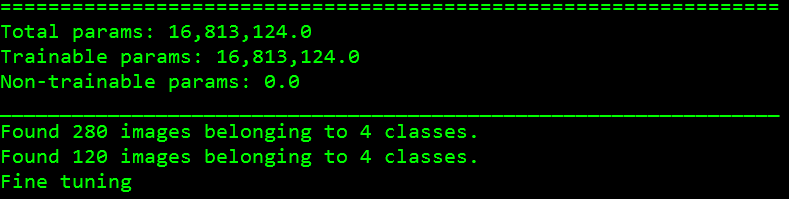


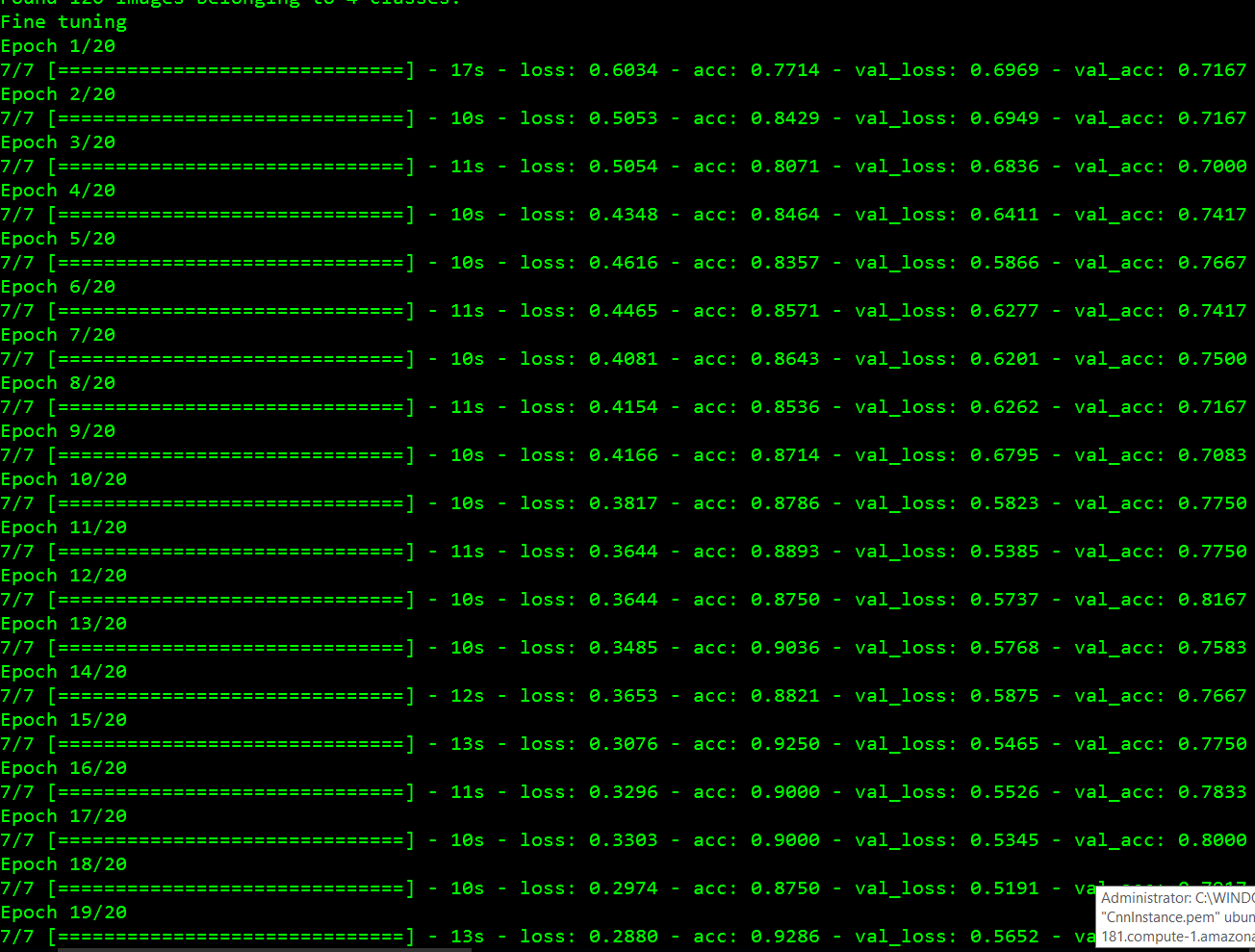
**DEMO RUN:**











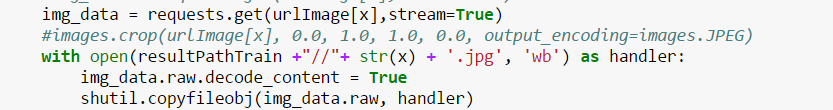
**CHALLENGES AND SOLUTIONS:**

***Challenge #1: Google API custom search issues****.*

a) *Limitation of 10 images in one call:* We had the requirement of getting 100 unique images of one object but google API only gives 10 images in one call. To fix this, we have called google API 10 times for one object each time passing new start number. The start number will take images from that index. In first call the start will be 1, second call start will be 11 and so on.

b) *Large size images:* Some large size images could not be written into our local disk. For that we have first compressed the image and then written into our local disk.

The code written for that:



***Challenge #2: CNN training model took more than 8 hours from scratch in CPU:***

The basic CNN model with 2 classes having 1000 images in training for each class and 300 images in test took more than 8 hours. To fix this, we have used VGG16 model. After using VGG16, it took 4-5 hours in CPU machine to train our model for 2 classes. Finally, to run our algorithm in few seconds, we have created AWS Ubuntu instance with GPU configuration. The steps are mentioned above.

***Challenge #3: CNN Model Batch Size and classes size problem***

a) The model will run only when all the train folders of each class have same number of images. To fix this, we have distributed our 70 images to train directory and 30 images to test directory.

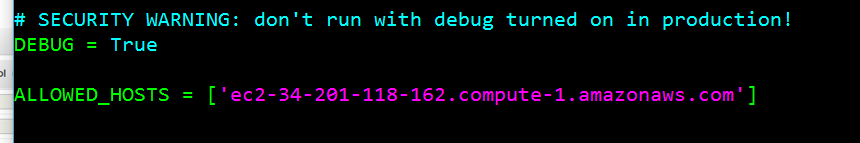
b) The batch size will be the GCD of images in train and test class. For example, if there are 70 images in 3 classes then total images are 210 for train. If there are 30 images in test folder for each class, then total images are 90. The gcd of 210 and 90 is 30. The batch size will be modified if classes are increased to 4,5 and so on.

***Challenge #4: GPU machine:***

a) The CPU machine trained the model very slow. For faster processing, we have used GPU. Installing GPU instance required few challenging steps that have been mentioned in topic: Creating and connecting to the GPU instance in Amazon

***Challenge #5: Django installation in AWS:***

a) Django server was not running on Ubuntu instance and was giving error: ‘Host not allowed’ To fix that, we have changed settings.py where we have modified below line:



b) The default python was python2 but we want to use python3 so changed our configuration to python3.

***Challenge #6: Quiver Engine***

a) Quiver engine is used to show the output of each convoluted layers. We could not do that part as we could not see output of each layer. Firstly, we faced problem with installation process. The command pip install quiver engine did not work. To make it work we changed the code in below github link code:

<https://github.com/keplr-io/quiver>

b) After installing it successfully, we tried to run the below command:

from quiver\_engine import server

server.launch(model)

The server was launched but it returned 404.

After going through several issues raised in github, we found that quiver engine is not good for complex models. In our case, the model is complex with many convoluted layers.

***Challenge #7: Image size more than 1 MB:***

a) If image in our dataset is more than 1 MB size, the model will not run accurately. To fix this, we have compressed our image when writing from google api.

**Future Scope**

**Facial key points detection**

facial key points detection has become a very popular topic and its applications include Snapchat, How old are you, have attracted a large number of users. The objective of facial keypoints detection is to find the facial keypoints in a given face, which is very challenging due to very different facial features from person to person. under different illumination conditions, positions, sizes, detecting facial keypoints would be very challenging

# **Analysis of satellite images for disaster detection**

It is difficult to obtain performance enhancement on disaster detection and management immediately, therefore CNN can be used to establish automatic disaster detection system by observing occurrence of disaster in a wider range through satellite images and observing every single disaster assisted by deep learning techniques.

### **Real Time Criminal Detection through Video analysis**

object detection and counting in low quality images and videos. Video is more complex than images since it has another (temporal) dimension. However, some extensions of CNNs into the video domain have been explored. One approach is to treat space and time as equivalent dimensions of the input and perform convolutions in both time and space. To deal with densely grouped objects with severe occlusion

**References**

<https://xkcd.com/1425/>

<http://cs231n.stanford.edu/reports/2016/pdfs/267_Report.pdf>

<https://en.wikipedia.org/wiki/Kernel_(image_processing)>

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>

<http://culurciello.github.io/tech/2016/06/04/nets.html>

<https://en.wikipedia.org/wiki/Convolutional_neural_network>