A WEB API for an Image Recognition System

Using Deep Learning Models (ResNet50 and VGG19)

CIS 600: Master's Project Report
Instructor: Dr. Ming Shao

Kevin D'Cruz
ID: 01656193
kdcruz@umassd.edu

Overview: An overview of the Project Report is as follows:

- Abstract
- Goal
- Technology Used
- Introduction
- Literature Survey
- Models Used
- Project Details
  - Part 1
  - Part 2
- Project Walkthrough
- API Screenshots
- Conclusion and Future Scope
- Code Snippets
- References

Abstract

Machine Learning applications with respect to text has been going around for quite a few years now. The next wave is in using state of the art machine learning models in image, audio and video data. Deep Learning or Deep Convolutional Neural Networks form the base for tackling some of the simplest as well as complex Computer Vision Use Cases or problems. Training a Convolutional Neural Network from scratch is complex as well as very resource dependent. This project is done with respect to image data and learning how CNN's are trained for recognizing images. The models that are pre-trained on CNN's are used, which provide the prediction of the image inputted along with its probability. Using these pre-trained models, an Image Recognition API is built, which, when given an input image, returns a bar graph showing the top five predictions as per the model and also shows the respective probability values of the same.

Goal:

The main goal or objective for this project was to have an in depth understanding of Deep Learning concepts were and how Convolutional Neural Networks are trained. Followed by this, building a WEB API where a user could input any image and the system would display the predictions of the image along with the corresponding probabilities.

Technology Used:

- Backend System:
  - Keras (Neural Network API)
  - Tensorflow (Google's Machine Learning API, serves as Keras Backend)
  - Python3 (Libraries: NumPy, Pandas, MatPlotLib)

- Frontend System (Web API)
  - Django
  - Bootstrap (HTML, CSS Javascript)

- Jupyter Notebook, Sublime Text 3

Introduction:

Image Classification is one of the most popular applications of machine learning. The algorithms used for this help in facial recognition, detection of vehicles, street signs and pedestrians on roads. They can also be used in Healthcare and other applications where visual data is necessary. Data Driven applications have transformed today's world and so has Image classification. Convolutional Neural Networks help with NLP as well as with graph convolutional networks. Computer Vision applications involve self-driving cars, Drones and robots. The efficiency provided by CNN's with Image, audio and video data has been far overwhelming, when compared with previous models in machine learning. These are an advancement of Neural Networks. This project focusses on using pre-trained deep learning models for recognizing household objects and building an API for the same. The Models make use of weights from ImageNet [12] and predict the object in the inputted image along with the probability of the top 5 predictions of the image.

Literature Survey:

Artificial Neural Network (ANN) is like that of a biological neural network, which is how the human brain processes information. Feedforward Neural Networks consist of multiple nodes or neurons that are arranged in layers. These Nodes are connected to each other and have certain weights associated to them. In other words, there is an input layer, a hidden layer and an output layer. These are of two types, Single Layer Perceptron and Multi-Layer Perceptron(MLP). An MLP can contain multiple hidden layers and work with non-linear functions. MLP's are nothing but full connected layers (in CNN terms) [13].

Back-Propagation Algorithm is the algorithm used for training a Multi-Layer Perceptron, and it works by assigning random values or weights to every node and then checking the output obtained. This output obtained is then compared with the already known output and the error obtained propagates back to the previous layer, followed by adjusting the weights. This process helps reducing the errors significantly and upon completion, the ANN is ready which can be used to learn new images [14] .

Classification Tasks, with respect to Back-Propagation uses SoftMax function as its activation function. Since most of the data is non-linear, this is responsible for giving non-linearity to the neurons output. Activation functions include: Sigmoid, tanh and ReLU. The Learning of a Multi-Layer Perceptron is mostly a two-step process, the first step being Forward Propagation and second, the Back-Propagation and weight updating, upon which the ANN is ready to predict new images, based on previously trained samples.

Convolution Neural Networks: These are specialized Neural Networks, which have proven to be very effective in the applications discussed above. LeNet architecture, which started off with its application of recognizing digits, formed the basis for CNN's [15]. CNN's include four main operations consisting:

- Convolution: Extracting the features from the input image, where every image is broken down into a matrix of pixel values. The convolved feature or feature map is obtained by performing the element wise multiplication and the outputs, forming a single element in the matrix. The image which is used for multiplication is the filter. Filters are the feature detectors to the original image. The size of the convolved feature is based on the parameters of:
    - Depth: Filters used for convolution
    - Stride: Number of pixels by which filter slides over first matrix
    - Zero-padding: Pads the matrix, puts zero's in the borders, which helps controlling the size of the convolved feature [16]

- Non-Linearity (ReLU): Rectified Linear unit, responsible for replacing all the negative pixel values with zeros. Non-Linearity can be achieved in the ConvNet, with the help of ReLU. Removing the negative values helps getting a much clearer image than before. The negative values are all nullified to zero[16].

- Pooling or Subsampling: This operation reduces the dimensionality of each convolved feature but retains the most important features of the image. Max pooling, Average pooling and sum pooling are different types, max being the mostly used type. Pooling mainly helps in reducing the spatial size of the input image representation. It thereby helps control overfitting, by reducing or removing unnecessary information from the image [16].

- Classification (Fully Connected Layer): It uses a SoftMax layer, which is the activation function. Since it's fully connected, every neuron in the network is connected to every other neuron with the network. By this time, the high-level features are obtained from the earlier operations (convolution and pooling). These high-level features are then used for classifying the image inputted into various classes, based on the training dataset. The SoftMax layer ensures that the sum of the output probabilities from a fully connected layer is always one[16].

Models Used:

Keras is a Neural Network API build using Python, where the Backend can be of TensorFlow, CNTK or Theano. Keras have made available some Deep learning models which use pre-trained weights for every neuron and these could be used for prediction, feature extraction and fine tuning. The models that I worked on along with their description are as follows:

- ResNet50 (Residual Network): This model uses weights that are pre-trained on ImageNet. The input image to be passed through this model needs to have the dimensions of 224*224. ResNets are state of the art CNN models and are usually the default model used with respect to Deep Learning applications. These Nets evaluated with a depth of up to 152 layers, which are 8x deeper than VGG nets on the ImageNet dataset. ImageNet is a project which did manual labeling and categorizing of images into roughly about 22,000 separate categories for research. ResNet won the first place on the ILSVRC 2015 classification task[10]. In a Residual Network, there are multiple stacked layers and are trained as per the task which is ongoing. This network learns low, mid, high level features at the output layer. Here, residual is learnt instead of trying to learn some features. It is the feature subtraction learnt from the layers input. Training such a network seems to be easier than training simple deep convolutional neural networks and degrading accuracy problem is resolved [2].
- VGG19: VGGNet has its main contribution in showing that the depth, or, the number of layers in a network, plays a critical role in obtaining a good measure of performance of the network. VGG16 and VGG19 are two popular Deep Learning Models. Like ResNet50, the target size for the input image of this model is 224*224 and the weights are picked from ImageNet. VGGNet uses only 3×3 convolutional layers stacked on top of each other in increasing depth. Max pooling is responsible for handling the reducing volume size. Two fully-connected layers, each with 4,096 nodes are then followed by a SoftMax classifier (above).
  The “16” and “19” are the number of weight layers in the network. VGG19 has 2 more layers than VGG16. These 2 models were state of the art in the earlier years, that is, before ResNet was created. VGGNets, however, are slow to train, because of the pretraining that is carried out and there are large network architecture weights. Overall, VGGNets are considered tiresome for deployment[10].

- Apart from ResNet and VGGNet there are InceptionV3 and DenseNet. InceptionV3, unlike the earlier models, takes in a input image size of 299*299. DenseNet model

has each layer directly connected to every other layer in the network in a feed-forward fashion. DenseNet is the most recent model which has shown significant results over the precious Deep learning Models. The Input size of the Image for DenseNet is 224*224 [1].

Project Details:

I am dividing the project I carried out into 2 parts, as follows:

Part 1:

Implementation of the Back-End for the Image Recognition System, in Keras with a TensorFlow Backend and Python Libraries (NumPy, MatPlotLib and Pandas). I used Jupyter Notebook as my text editor, followed by Spyder. Both these editors are from the Anaconda Package.

I started working for the BackEnd system by creating an environment of TensorFlow and Keras. Neural Networks was always an intriguing topic for me, which made me curious into the technology that is, deep learning. I started getting familiarized by reading the popular Deep Learning papers and understanding how the popular models are built. With a brief understanding of what Deep Learning is, I made use of online resources and created a small digit recognition system, which used the Cifar10 dataset and the sequential model, another model available in the Keras API.

Followed by this, the first model I implemented was the VGG16 model. The results provided were good, however I wanted to know how the other pre-trained models faired when the same image was fed into the system. Followed by VGG16, I carried out the same process with the Deep Learning Models of VGG19, ResNet50, InceptionV3 and DenseNet. ResNet50 and DenseNet performed equally well whereas VGG19 performance was slightly less than ResNet50, in terms of probabilities of the highest predicted outcome. Hence, the BackEnd System, with respect to the five Deep Learning models were ready.

My next step in the project was to create a Web API for the Image Recognition System, where a user can enter any image and the chosen model predicts what the image is and returns the top5 probabilities as per it's prediction.

Part 2:

I used Django for creating the Web API and Bootstrap for the GUI. The ResNet50 and VGG19 models created in Part 1 are integrated with the UI in this part. I planned on integrating all the 5 pre-trained models and then compare the probabilities of the model which gave the highest prediction of the output as well as perform some functionalities such as average. However, my local machine was unable to load multiple models at the same time and TensorFlow would not let me through. At the same time, 5 models reduced the probabilities of the predicted outcomes significantly. Hence, I had to remove VGG16, InceptionV3 and DenseNet from the API's. These models worked fine upon loading them one at a time and showed good probability values (in Part 1).

Project Walkthrough:

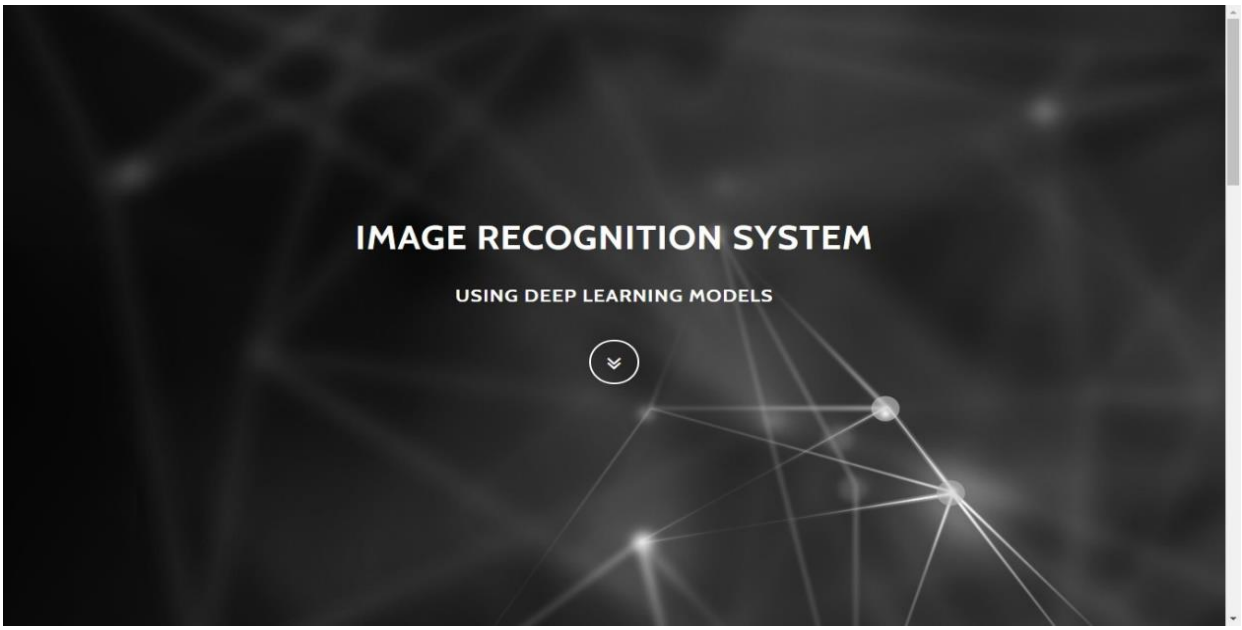
Steps to be carried out to use the Image Recognition System, (by any user):

Demo: https://www.youtube.com/watch?v=OAcn6y5S23Y

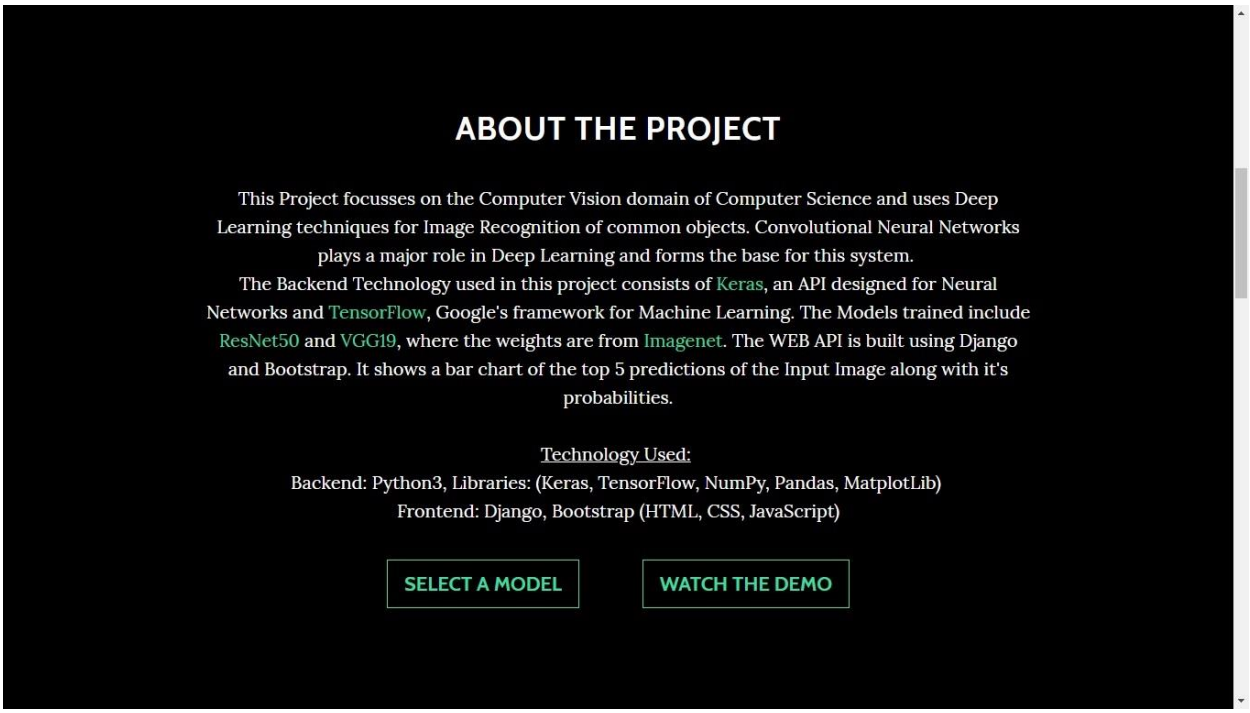
To use the Image Recognition System, the user will have to create the environment, such that all the depending python libraries are installed. I will be sharing my GitHub link, where the README.md file will show steps on creating a virtual environment and install the required dependencies using the '*pip freeze > requirements.txt*' command in CMD.

Once, the environment is set and the server is started, the user gets an option of selecting a model. I had integrated all the five models in the system, such a way that the user could select an image and then view the average prediction of all the five models. However, upon integration of all 5 models, the probability of the models was deteriorating and hence, had to be removed. The final pre-trained models used for image recognition in the system are ResNet50 and VGG19.

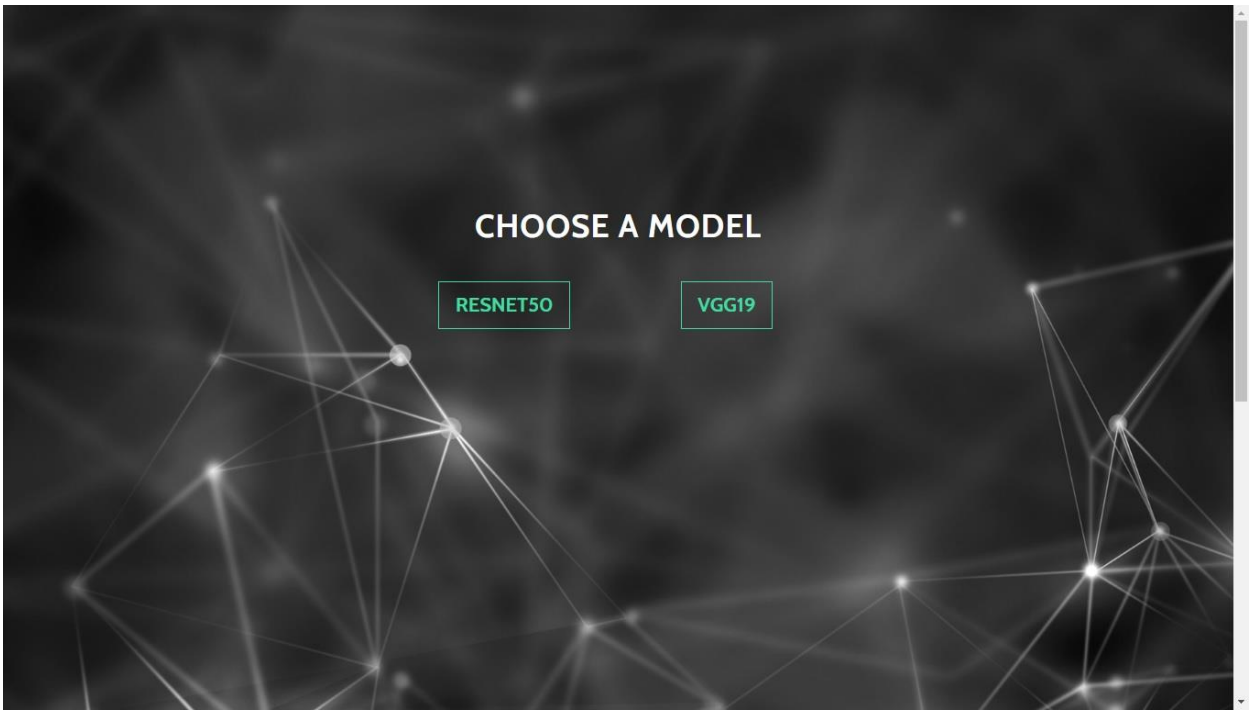
The user selects the model, browses for an image from his/her local machine and then hits the predict button, which returns a bar graph showing the highest probability of the image. The GitHub Link to my repository is referenced below, reference no. 16.



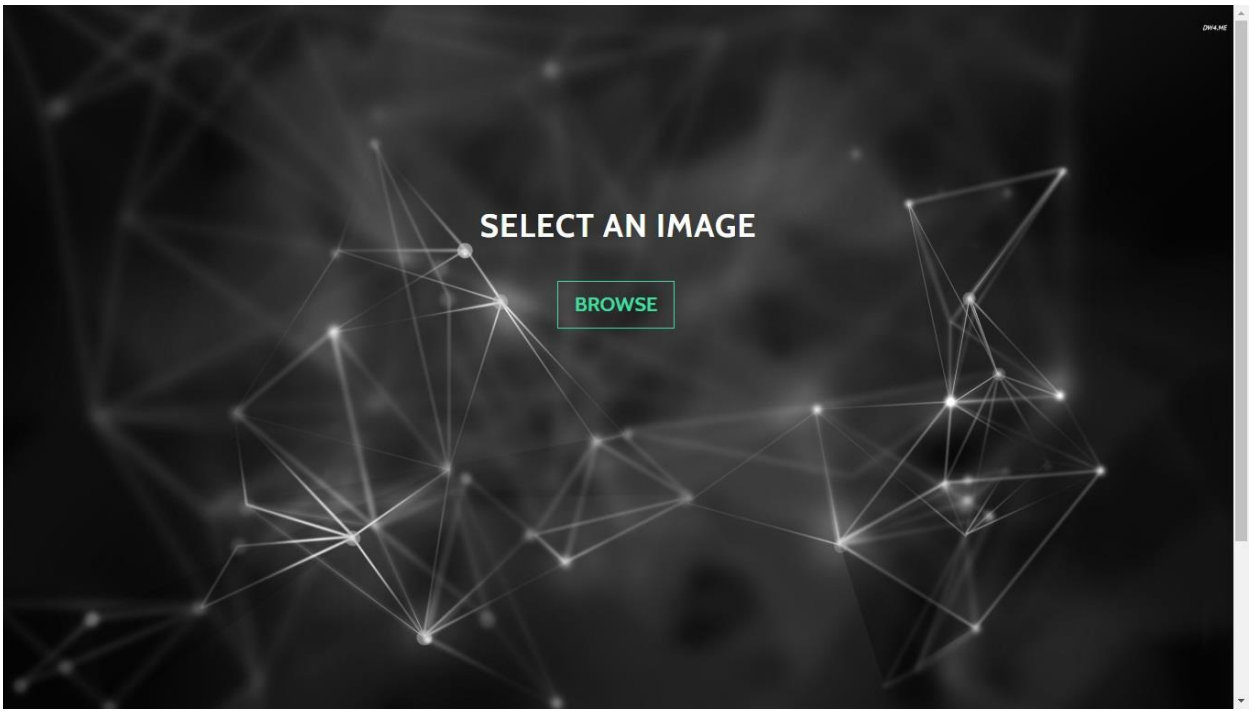
Screenshot 1: Homepage

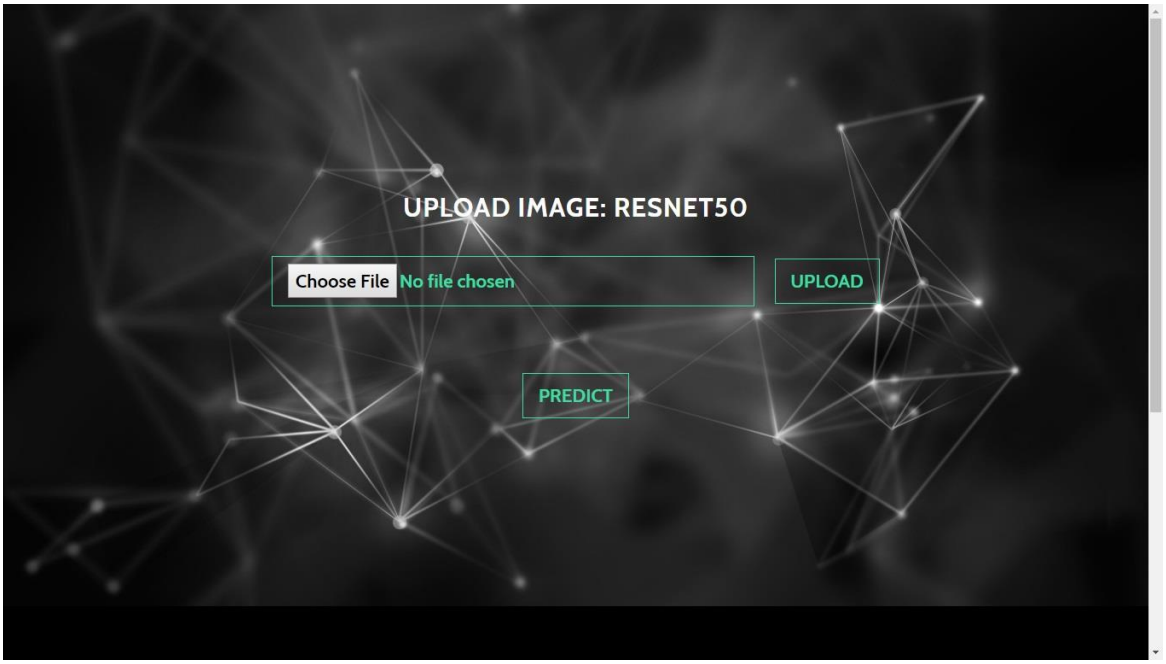


Screenshot 2: About the Project

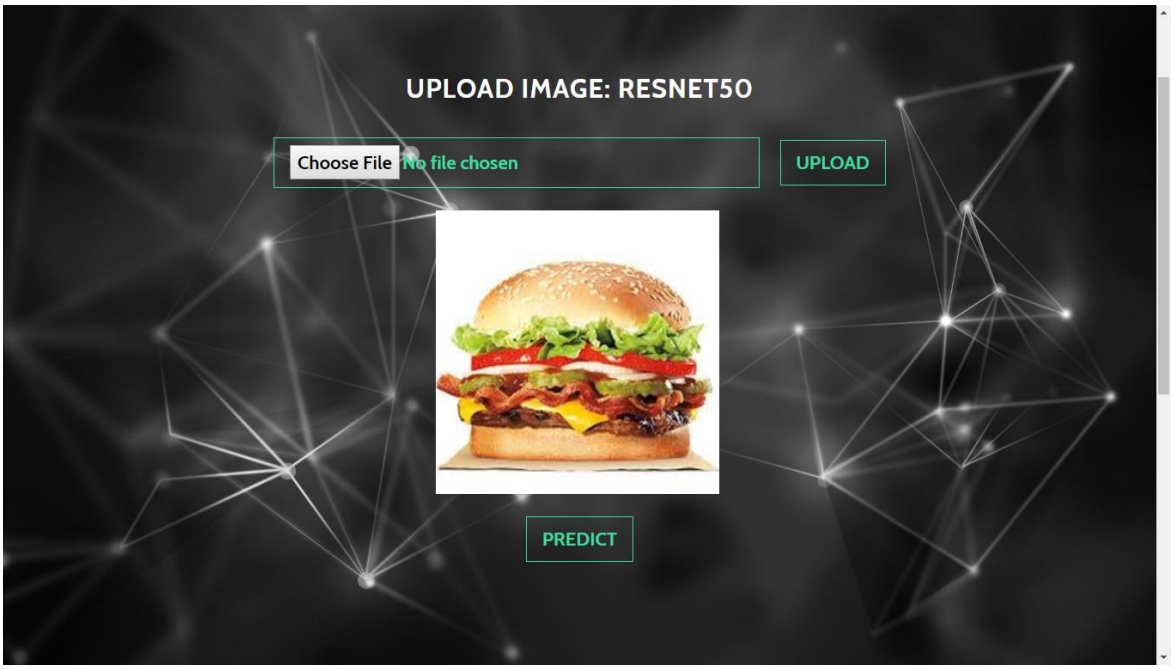


Screenshot 3: Choosing a Model (Chose ResNet50 Here)

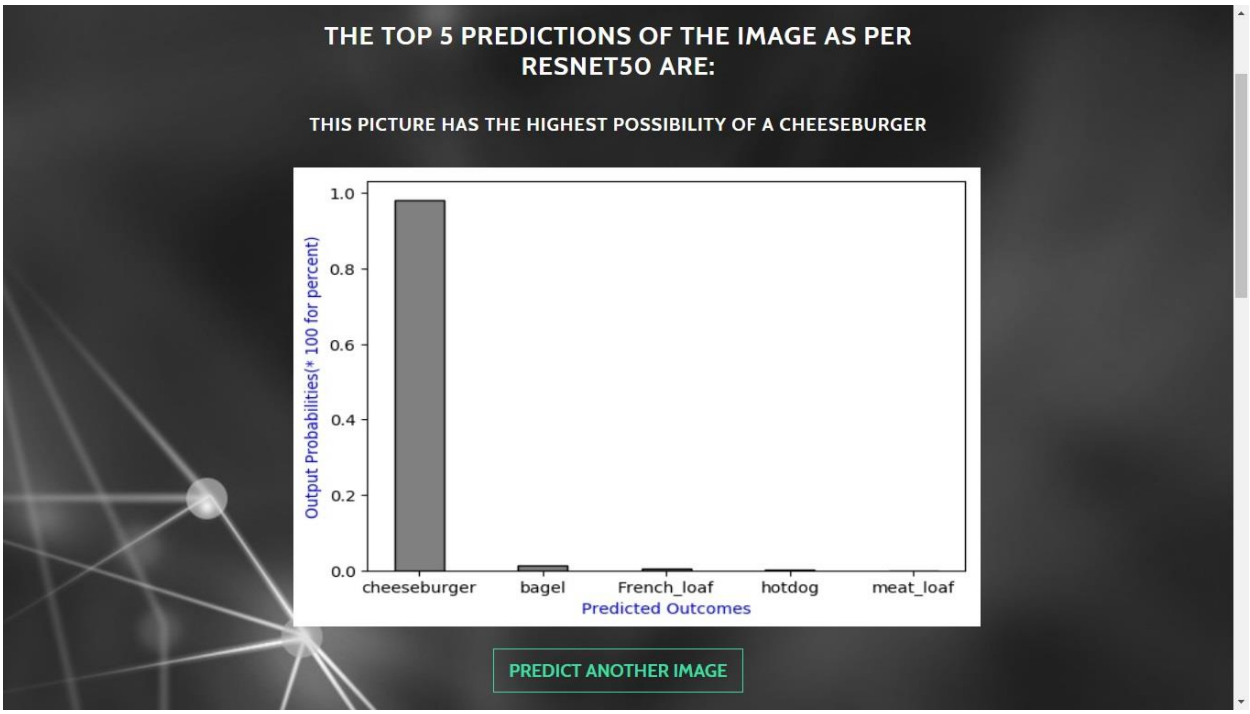




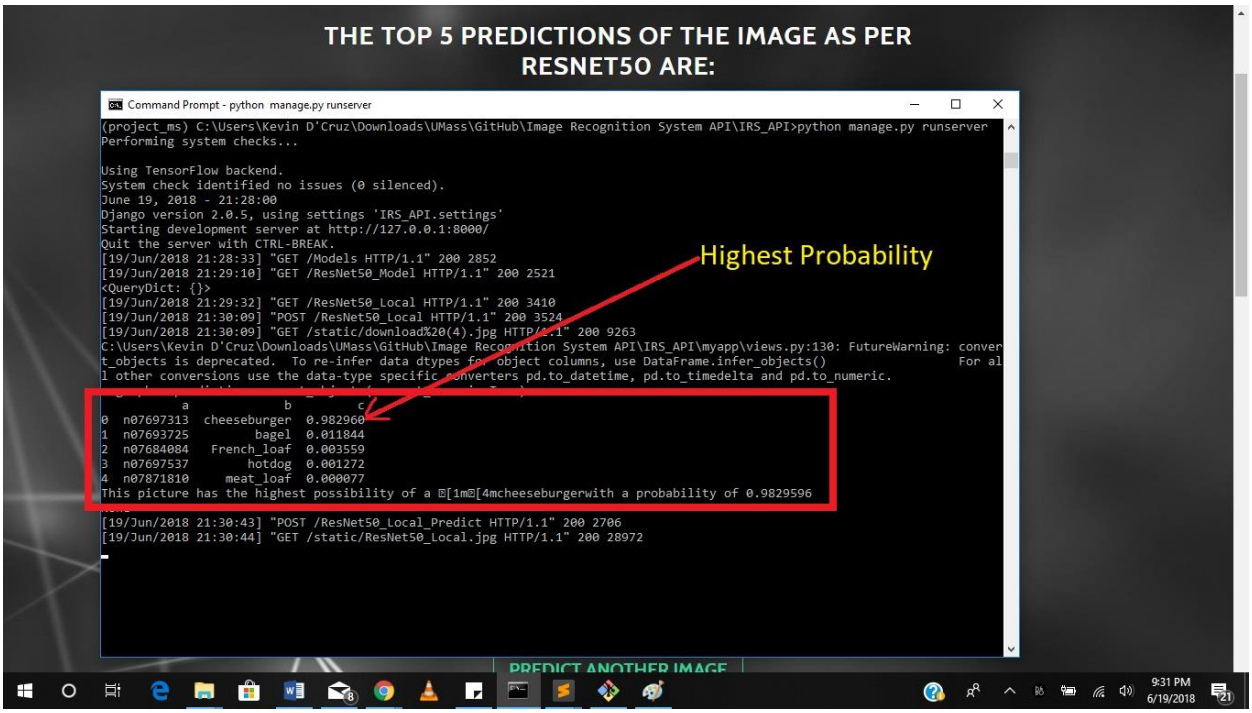
Screenshot 4 and 5: Browsing for an Image



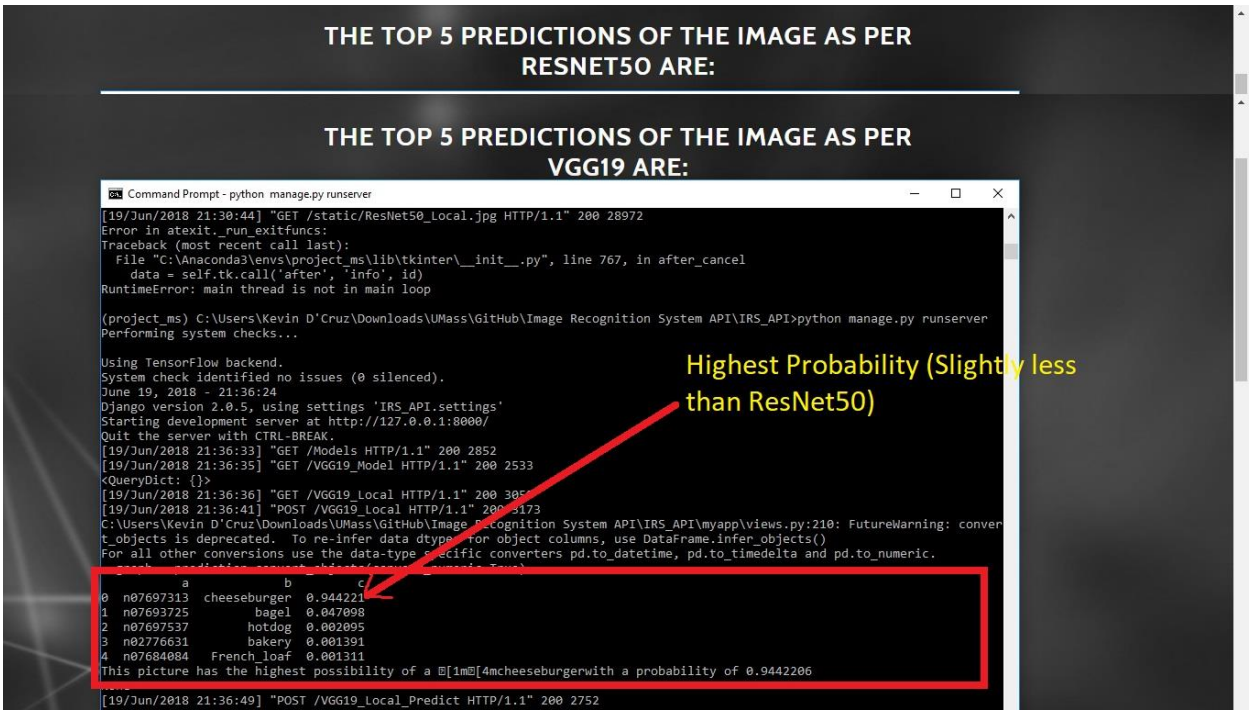
Screenshot 6: Image Selected, followed by hitting the Predict Button)



Screenshot 7: Bar Graph, showing the Probabilities of the top 5 Predictions

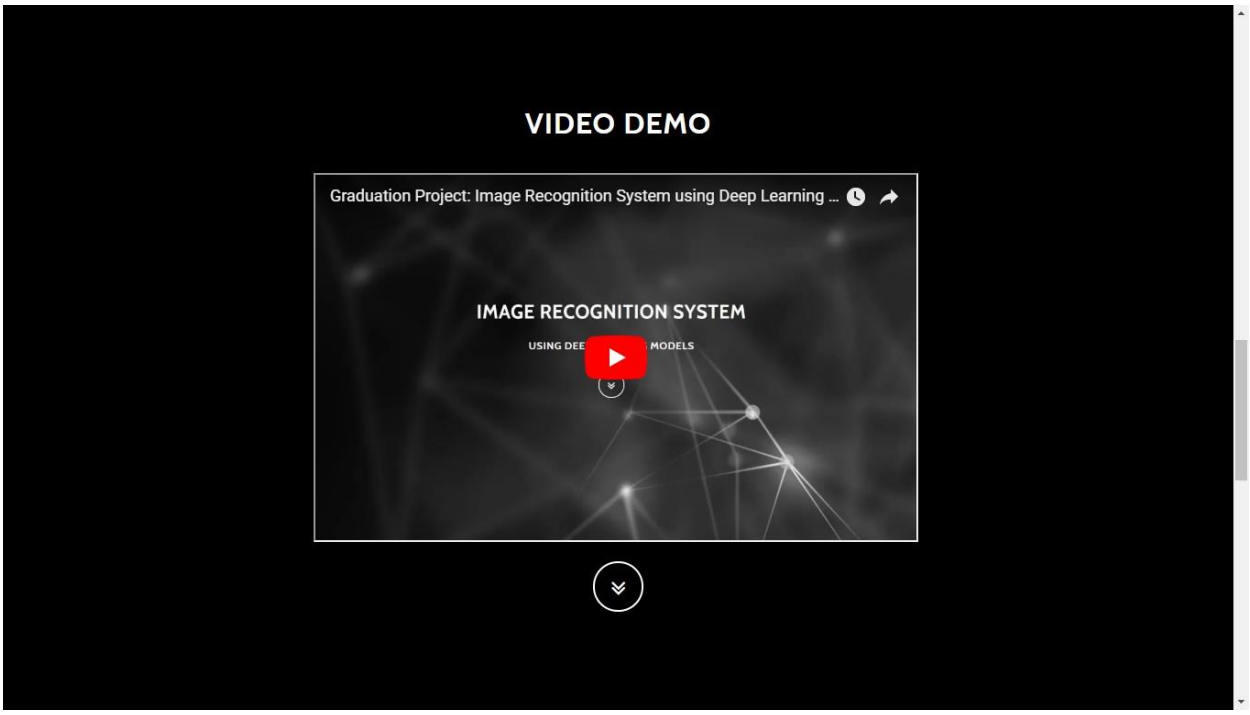


Screenshot 8: The Top 5 Probability Values, from the CMD



Screenshot 9: The Top 5 Probabilities of the same image, using VGG19 Model

Comparing Screenshots 8 and 9, the Probability value of a Cheeseburger with ResNet (0.982960) is more than that of VGG19 (0.944221)



Screenshot 12: Video Demo of the Project Walkthrough is also attached

This video will show what the project is all about, If the user doesn't want to install and run the API on his/her local machine.

Conclusion and Future Scope:

This project helped me have a concrete understanding of the underlying Deep Learning models, how convolutional neural networks work and a basic understanding of what it takes to built one such network from scratch. Building the API helped me learn Django which I never used before in my projects. I plan on extending this project further, by trying to build a CNN from scratch and integrate it with these pre-trained models and compare the results in the API itself.

Code Snippets (for ResNet50 Model):

```
#------------Model Loading---------------------------------
model = ResNet50(weights='imagenet')
target_size = (224, 224)
```

```
#Prediction function local image
def predict(model, img, target_size, top_n=5):

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  preds = model.predict(x)
  return decode_predictions(preds, top=top_n)[0]
```

```
#Prediction function local image
def predict(model, img, target_size, top_n=5):

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  preds = model.predict(x)
  return decode_predictions(preds, top=top_n)[0]
```

```
#Plotting: Local Image
prediction = predict(model, img, target_size)
prediction=pd.DataFrame(np.array(prediction).reshape(5,3), columns = list(
"abc"))
print("This picture has the highest possibility of a "+'\033[1m' '\033[4m'
+prediction.b[0])
graph=prediction.convert_objects(convert_numeric=True)


plt.bar(graph.b, graph.c, align='center', color='gray', edgecolor='black',
width=0.4)
plt.rcParams['figure.figsize'] = 13,6
plt.xlabel("Predicted Outcomes", color='blue')
plt.ylabel("Output Probabilities", color='blue')
plt.show()
```

References:

1. Very Deep Convolution Networks for Large Scale Image Recognition: Karen Simonyan & Andrew Zisserman: https://arxiv.org/pdf/1409.1556.pdf

2. Deep Residual Learning for Image Recognition: https://arxiv.org/pdf/1512.03385.pdf

3. Rethinking the Inception Architecture for Computer Vision: https://arxiv.org/pdf/1512.00567.pdf

4. Densely Connected Convolution Network: https://arxiv.org/pdf/1608.06993.pdf

5. Vincent Dumoulin, et al, "A guide to convolution arithmetic for deep learning", 2015: https://arxiv.org/pdf/1603.07285v1.pdf

6. Feature Extraction using Convolution: Stanford:

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

7. CS231n: Convolutional Neural Networks for Visual Recognition:

http://cs231n.github.io/convolutional-networks/

8. Deep Learning Book: http://www.deeplearningbook.org/

9. Django Documentation: http://www.deeplearningbook.org/

10. Keras Documentation: https://keras.io/

11. TensorFlow Documentation: https://www.tensorflow.org/api_docs/

12. ImageNet: http://www.image-net.org/

13. A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in ISVC, pages 867-877, 2015

http://scs.ryerson.ca/~aharley/vis/harley_vis_isvc15.pdf

14. Understanding Convolutional Neural Networks for NLP

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

15. http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/

16. My GitHub Link: www.github.com/KevinDCruz