Birla Institute of Technology and Science

A Report on

Detection of Windmills from Satellite Images using Deep Neural Networks

BY

ASHUTOSH PUROHIT 2015ABPS501P

At

Regional Remote Sensing Centre - West, Jodhpur

Birla Institute of Technology and Science

Birla Institute of Technology and Science, Pilani (RAJASTHAN)

Station: Regional Remote Sensing Centre - West **Centre**: Jodhpur

From: 22nd May 2017 **To:** 15th July 2017

Date of submission: 12th July 2017

Title of the project:

Detection of Windmills from Satellite Images using Deep Neural Networks

ID No.	Name of student	Discipline
2015ABPS501P	ASHUTOSH PUROHIT	MANUFACTURING ENG.

Name(s) of expert(s)

Dr Rakesh Paliwal

Mr. Gaurav Kumar

Name of the PS faculty

Mrs. Anubha Dadhich

Key Words:

Deep Learning, Neural Networks, Satellite Images, Object Detection

Project area: Deep Learning

Signature of student

Date:

Abstract

The Project is Titled "Detection of Windmills from Satellite images using Deep Neural Networks".
The Project can be divided into four major steps excluding the learning of prerequisites.

Step Zero (The Learning Curve):
For application of this Computer Vision problem. We did Coursera course on Machine Learning and watched lectures from Stanford's Deep Computer Vision class,CS231n.

Step One (Collection and annotation of data) :
We got the Coordinates of Various windmills around the country (1988 windmills) from our Mentor Shri Gaurav out of which we scrapped satellite image of 1200 of them from internet using a script that we wrote. Initially we thought to Auto annotate them using the grayscale image and thresholding the lighter parts of the Image but it did not work out because of dark features of land. So Later We also worked on a script to manually Annotate them. We decide to annotate 60 images out of which we already annotated 200 images.

Step Two (The Algorithm) :
We decided to implement an Algorithm named "Faster R-CNN (Regional Convolutional Neural Network)" for this problem. We looked among different deep learning Frameworks to work on and after a lot of discussion between Tensorflow and Pytorch we decided to go for the former. However an optimised object detection API was released and we decided to use that. We went through all the documentation and are currently preparing the dataset in accordance to

the input requirements. We are also working on Pipelining Via Protobuffs. The Pipeline specifies the architecture of neural network and various parameters of the API. We also decided to try out Transfer learning on another API named TensorBox which used GoogLeNet.

Step Three (Training of the network) :
The Tensorbox was trained on ISRO Provided machinery whereas the Google API was trained on a GPU owned by us.

Step Four (Making the Interface for trained network) :
After Training the algorithms we ran the model on an IPython notebook.

Acknowledgements

I would like to use this opportunity to express my gratitude to everyone who supported me throughout the course of this project. I am thankful for their aspiring
guidance, invaluably constructive criticism and friendly advice during the project work.
I am sincerely grateful to them for sharing their truthful and illuminating views on the number of issues related to the project.

I would like to thank Dr. SS Rao, The General Manager of RRSC-W for providing me opportunities to work in such an esteemed research institute. I would like to extend our heartfelt gratitude to Dr. Rakesh Paliwal and Mr. Gaurav Kumar for giving me this great opportunity of working under them on such an interesting topic.

I would especially like to thank our PS instructor Mrs. Anubha Dadhich, for her constant support, words of wisdom and timely suggestions.

INDEX

S.NO	TITLE	PAGE NO.
1	The Problem Statement	8
2	Need for the Project	9
3	THEORY ➢ Neural Networks ➢ Deep learning ➢ Object detection ➢ Faster R-CNN	10
4	Creation of Dataset	16
5	Setting up the machinery	17
6	The frameworks used- ➢ Google Tensorflow Object Detection API ➢ Setting up and Training ➢ Tensorbox introduction ➢ Setting up ➢ Training	18
7	Results	28
8	Further applications and scope	32
9	Appendix	33
10	References	44

The Problem Statement

Object detection is one of the most classic problems in the field of computer vision. The most robust way to tackle this is via using deep neural networks.
Our project is titled" Detection of windmills from satellite images using deep neural networks".
The satellite image of windmills has huge peculiar shadows making them easy to spot and identify. Hence in our project we have to work out how to detect them automatically and localize the object in the image of given resolution by drawing bounding boxes around the area of high probability of windmill.

Fig1.1 *A sample prediction of windmills during training*

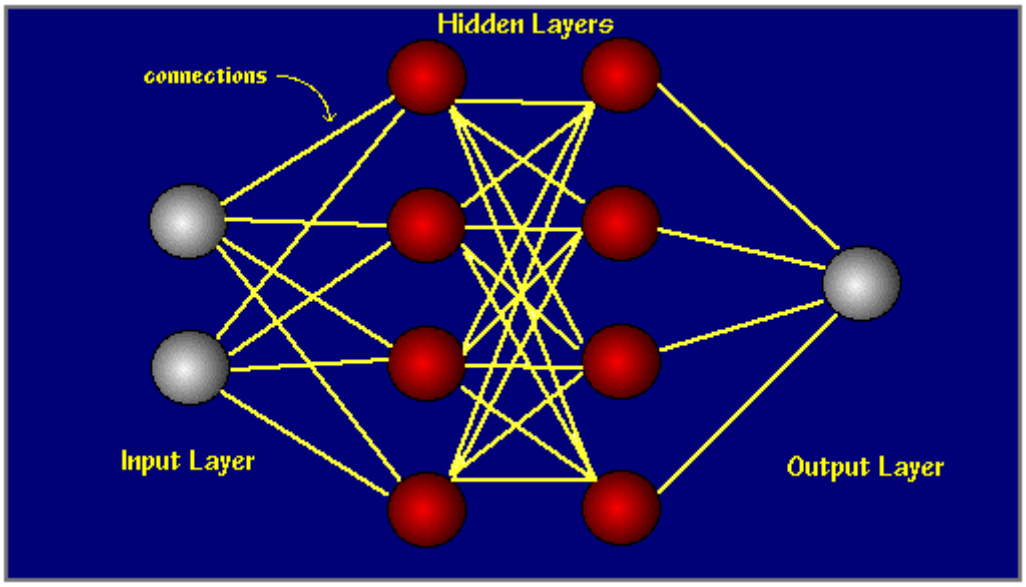
NEED FOR THE PROJECT

Regional Remote Sensing Centres (RRSCs) support various remote sensing tasks specific to their regions as well as at the national level. RRSCs are carrying out application projects encompassing all the fields of natural resources. Since windmills are a potential source of renewable energy and upcoming especially in Rajasthan and Gujarat ,this project holds an immense value to the centre. Designing a classifier which identifies the windmills and counts the number of windmills in Rajasthan is of great use in estimating power generated by the windmills.Also we can look at the distribution of the windmills in rajasthan and obtain their respective coordinates. Also using this an example we can train classifiers to detect other objects in satellite images such as solar panels ,aircrafts etc.

Theory

Neural Networks

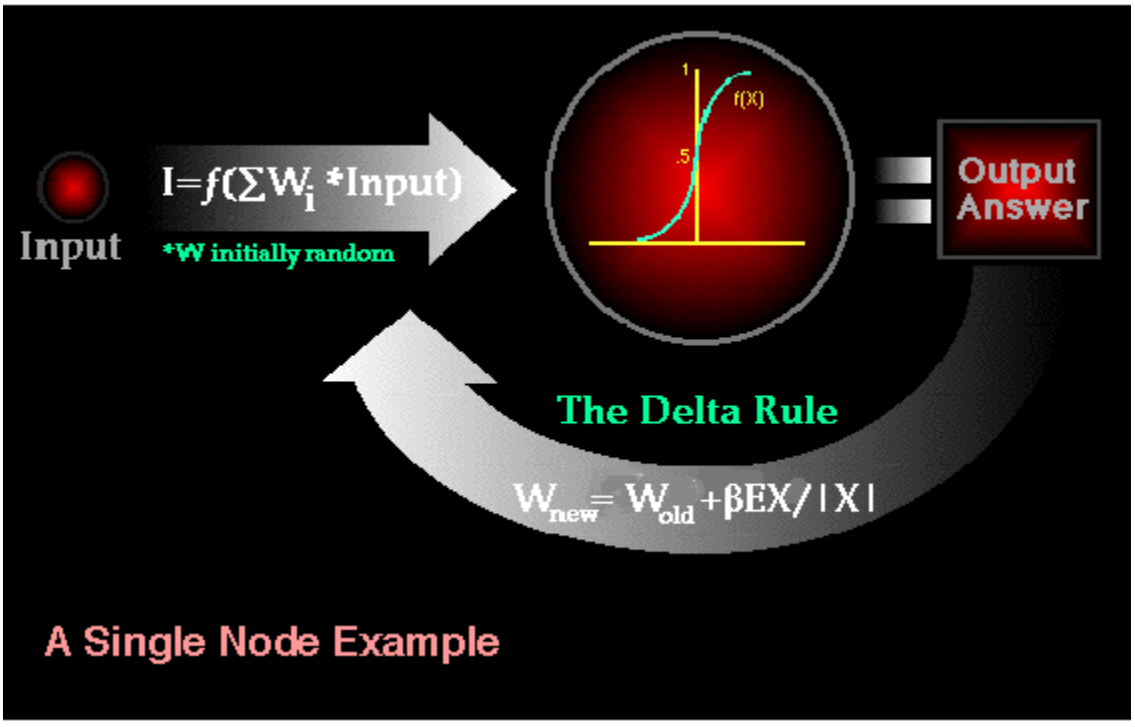
Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.



Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts; a child learns to recognize dogs from examples of dogs.

Although there are many different kinds of learning rules used by neural networks, we will take example of only one; the delta rule. The delta rule is often utilized by the most common class of ANNs called 'backpropagational neural networks' (BPNNs). Backpropagation is an abbreviation for the backwards propagation of error.

With the delta rule, as with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. More graphically, the process looks something like this:



Note also, that within each hidden layer node is a sigmoidal activation function which polarizes network activity and helps it to stabilize.
Back propagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. The error surface itself is a hyperparaboloid but is seldom 'smooth' as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minum' which is not the best overall solution. How the delta rule finds the correct answer

Since the nature of the error space cannot be known a prior neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

It is also possible to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "grand mothered" in neural network jargon. In real-world applications this situation is not very useful since one would need a separate grand mothered network for each new kind of input.

We use a variation of artificial neural network known as convolutional neural networks.

Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

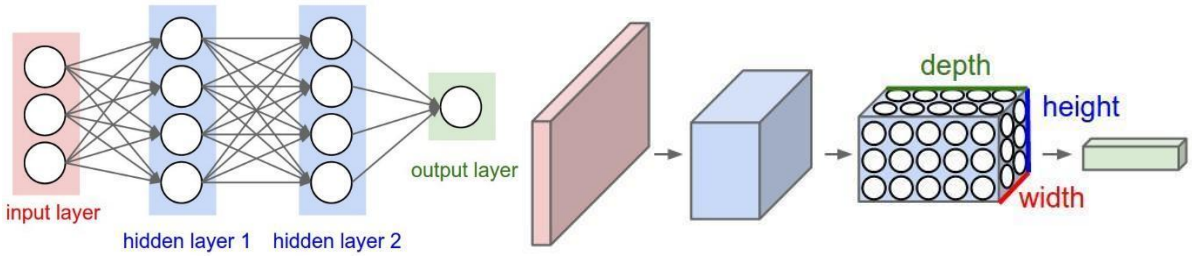
Architecture Overview: Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is

called the “output layer” and in classification settings it represents the class scores.

Regular Neural Nets don’t scale well to full images. In CIFAR-10 (an example dataset with 10 classes), images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to over fitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

Here is a visualization:



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Object detection

One of real world usage of machine learning is object detection: find object location on the image. Facebook finds faces on photos; Google self-driving cars detect pedestrians, cars and traffic signs.

The system should get an image as an input, and provide a location (x, y, width, height of a rectangle) of windmills on the image. It is not much useful to send the whole image to a neural network and expect to get an object location, as a result. Probably it is possible to train such neural network, but it will take tremendous computational power and huge training dataset.

Sliding window approach can be used instead. A neural network is trained for classification of fixed size images, for an example it can just classify if the image is an eye or not. The system gets part of the image and classifies it. Then it gets

another part of the image, slightly sliding its position, and so on. When all the image is covered, it starts over with sliding window of different size. When the process finishes, there are results of classification for every part of the image, therefore a location of all found objects is known.

Faster RCNN

Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions (Region Proposal network), and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection. For depth in details in is recommended that you go through the paper here:
https://arxiv.org/pdf/1506.01497.pdf

Creation Of Dataset

To train our neural network, we needed a comprehensive dataset consisting of satellite images with better resolution of windmills along with their shadows. This dataset should be such that it contains all the types of windmills found in Rajasthan. Our emphasis was on covering the shadow part of the windmills which could be easily identified in the dry arid region. We needed a minimum of 3000 images for proper training. The larger the dataset the better our results should be.

A previous project had been done in RRSC-W which manually looked for windmills. The KML (KML is a file format used to display geographic data in an Earth browser such as Google Earth) files of that project were available. It contained 1918 coordinates of windmills in Rajasthan. To extract the coordinates

of these 1918 coordinates, we made a python script which looked for specific tags in the file and then wrote those coordinates into excel worksheet.
Since ISRO portal Bhuvan couldn't provide us with high resolution images we decided to use data scraping techniques to load Google satellite images and download onto our hard disks. For this we created a python script which when given a longitude ,latitude and zoom level would download a high resolution satellite image of 1024*1024 pixel with the windmill coordinates in the center of the image. It did so by converting the image into tiles and then stitching the tiles together to get the whole image. After a sample download we then wrote another script that would extract the coordinates from a excel file and batch download the images according to their coordinates.

Setting up the basic machinery

- Install Ubuntu 16.04 LTS
- Install Anaconda2 (As we require Python 2.7 if Anaconda 3 is installed we can create and environment with Python 2.7)
- Install CUDA 8.0
- Download CuDNN library and move its content in respective folders of CUDA installation
  Open and Change the .bashrc file by adding at its end

```
Export
LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64"
Export CUDA_HOME=/usr/local/cuda
```

- Restart terminal

- Install tensorflow with GPU using

$ pip install tensorflow-gpu

- Use pip to install the following :
    - runcython>=0.2.5
    - opencv-python>=3.2

Frameworks Used

Tensorflow Object detection API

About the API

The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models**.** It is highly customizable with a config file to define various parameters of a network.In out problem we Use Faster R-CNN algorithm applied to Inception Resnet V2 network Architecture with Atrous convolutions.

Setup and Training

(The following has been written as complement to official documentation, go through it first) :

- Do the Basic machinary installation
- Download the API from its GitHub Repository https://github.com/tensorflow/models/
- Follow the setup guide at https://github.com/tensorflow/models/blob/master/object_detection/g3doc/installation.md

- Build you dataset using the instructions provided :
  (I would recommend to build you dataset as a clone to oxford-IIT Pets dataset http://www.robots.ox.ac.uk/~vgg/data/pets/)
  1. Annotate all the images and save annotation in annotation format like one in appendix
  2. Make sure your dataset is in jpg
  3. Make trainval.txt like given in appendix. This contains all the images you want in training and the validation set. They are randomly distributed in 50:50
  4. You will need to make a label map as .pbtxt file like one in appendix
  5. The folder hierarchy for making the dataset is like:

+object_detection
+annotations
+xmls(contains all the annotations in xml format)
-trainval.txt
+data
-windmill_label_map.pbtxt
+images (contains all the images)

  6. Convert your dataset into tfRecord format using the script (we used create_pet_tf_record.py which is in appendix) You can modify it. Read https://github.com/tensorflow/models/blob/master/object_detection/g3doc/preparing_inputs.md for running the script.
  7. You will get two .record files, one for training and another for validation.

NOTE: While writing this Report a new Tutorial has been released in documentation specifically about using your own dataset.

(https://github.com/tensorflow/models/blob/master/object_detection/g3doc/using_your_own_dataset.md) It is highly recommended that you use that.

- Now that you have created the dataset you must prepare the procedure for training
- Build the .config training pipeline in accordance to the guide provided in the documentation, This proto file will specify all the nitty-grity details of your network. You can look into sample configs in samples folder (https://github.com/tensorflow/models/blob/master/object_detection/g3doc/configuring_jobs.md)
- Now put the folder hierarchy as:

```
+object_detection
        +data
            -windmill_label_map.pbtxt
            -train.record
            -val.record
        +models
            +model
                +eval
                +train
                 -config.proto
```

- Begin training the network with instruction provided in documentation https://github.com/tensorflow/models/blob/master/object_detection/g3doc/running_locally.md . It is advised that you use two GPU's one to run training on and second to run evaluation on.
- You might run into an error where the Loss tensor becomes NaN, you will need to start training again, it will resume from a check point

- Export your checkpoint into a graph as explained in the documentation https://github.com/tensorflow/models/blob/master/object_detection/g3doc/exporting_models.md
- Run it by putting your test images in test images folder and renaming them as image1, image2 and so on and modifying the sample iPython Notebook, by changing the parameters such as imported graph path, number of images to be run on etc. https://github.com/tensorflow/models/blob/master/object_detection/object_detection_tutorial.ipynb <- Modify this script.

Sources of error:

- In creation of dataset:
  - You might have given wrong filename in xml file.
  - Your Pathways might be wrong
  - Make sure xml is valid
- In training
  - Initialize the .proto as .config file properly with all the paths

Tensorbox

About Tensorbox

TensorBox is a simple framework for training neural networks to detect objects in images. Training requires a json file containing a list of images and the bounding boxes in each image. The basic model implements the simple and robust GoogLeNet-OverFeat algorithm with attention.

About the architecture - GoogLE OVERFEAT

It presents an integrated framework for using Convolutional Networks for classification, localization and detection. It presents a slight modification to previously existing AlexNet by showing how a multiscale and sliding window approach can be efficiently implemented within a ConvNet. It also introduces a novel deep learning approach to localization by learning to predict object boundaries. Bounding boxes are then accumulated rather than suppressed in order to increase detection confidence. It shows that different tasks can be learned simultaneously using a single shared network. This integrated framework is the winner of the localization task of the ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC 2013) and obtained very competitive results for the detection and classifications tasks. In post-competition work, it establishes a new state of the art for the detection task. Finally, we release a feature extractor from our best model called OverFeat.

Understanding the network

The network uses JSON objects as input for the images. These objects contain the annotated coordinates and the path of the image in a dictionary (python) format. Hence all images are to be first annotated and referenced by JSON objects and then input to the network.

Cloning the Framwork

Open Terminal

Run : *$ git clone http://github.com/russell91/tensorbox*

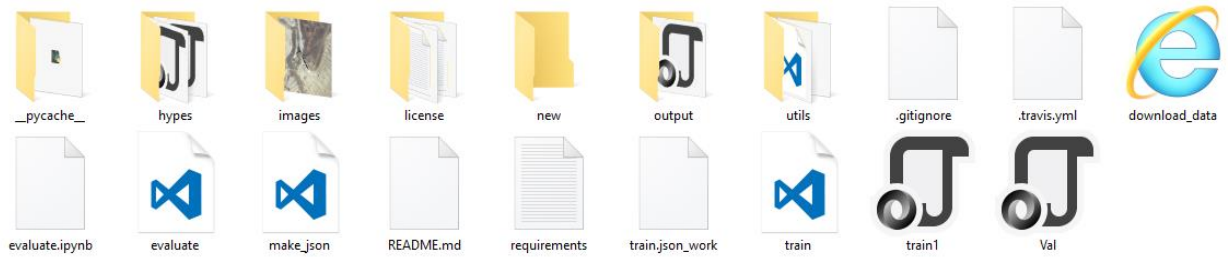


Fig2: *Inside the Tensorbox Folder*

Creating JSON objects

- Create a folder in which the images that are to be used in training are stored
- Run : *python make_json.py folder_name Dest_file.json*
  - Where folder_name and Dest_file is to be replaced with your required folder/file and path.
- Annotate each image
  - You can annotate multiple features within a single image!
  - While annotating we selected the windmill as well as its shadow. This was done because in arid regions the shadow of the windmill is more prominant than the windmill itself. However to let the network extract the relation between the windmill and its shadow we

selected the windmill as well as its shadow. This increases the accuracy of the network

Fig3 *Annotating an image*

Training The Network

To train the network in accordance with your own dataset follow the below commands:

- *cd /path/to/tensorbox/utils && make cd ..*
    - This would shift control to the utils directory and compile the stitch_wrapper files present there
- Go to hypes/overfeat_rezoom.json and change the training as well as the validation file to what you require.
- Return to Tensorbox directory
- Run: *python train.py –hypes/overfeat_rezoom.json –gpu 0 –logdir output*

- There would be some warnings while running this command, these can be ignored as they won't affect the output

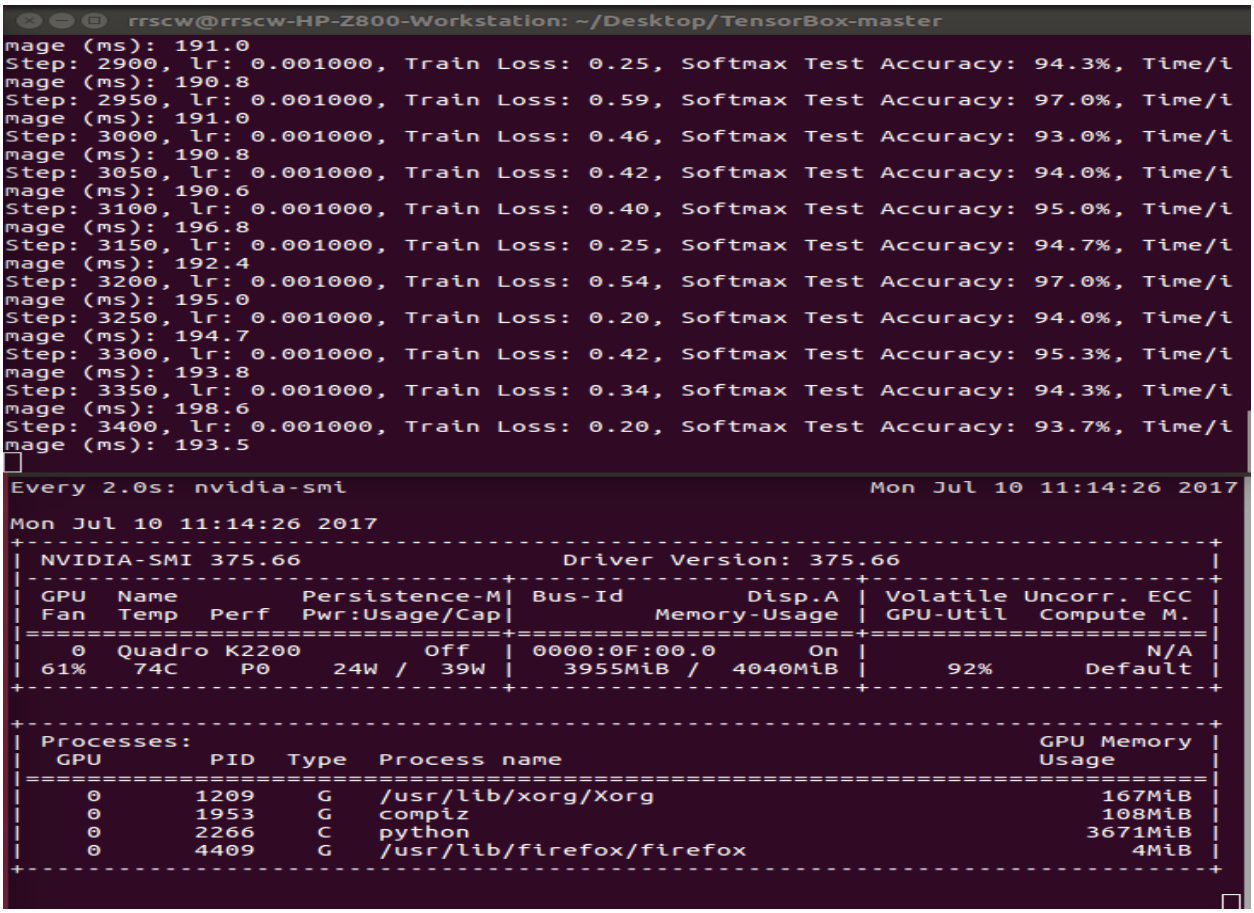


Fig 4 *Training and GPU usage*

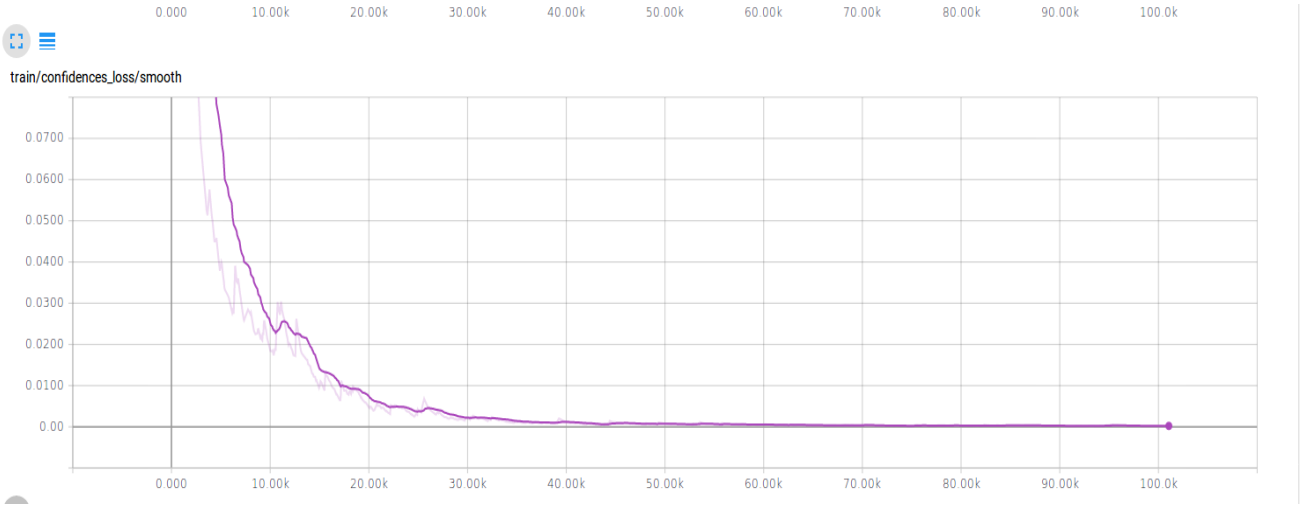


Fig5 *ConfidenceLoss vs Epochs*

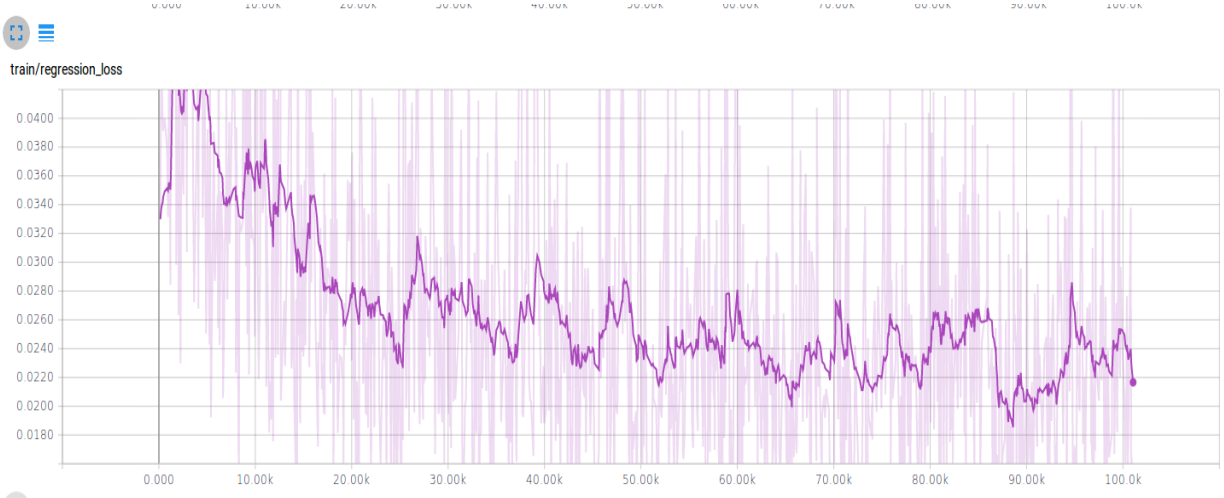


Fig6 *RegressionLoss vs Epochs*

Notes:

- Make sure to be running on python2 and a linux OS
- Training would take time, be patient
- Train the network till atleast 100000 steps
- Checkpoints would be made after every 10000 steps, and can be found in the output directory
- There is no end to training, the program would run indefinitely, and will have to be terminated when it is felt that the losses have converged to a near 0 value

Evaluation

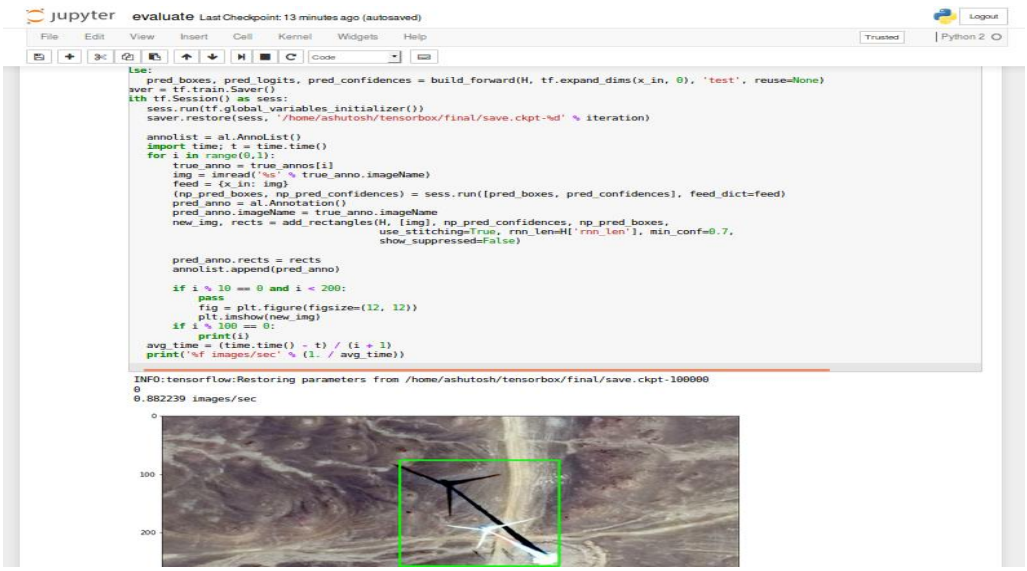
Evaluation and Testing the Network

Evaluation is an important component of a neural network. It allows the developer to understand how well has the network been trained and to what extent has it been trained. Testing is done on images that have not been used in the dataset. Evaluation in TensorBox involves the use of an ipython notebook. In the following steps I have used Jupyter Notebook.

Steps to Evaluate / Test

- Create test/evaluation images
    - Put the images, you want to test the network on, in a folder.
    - Run the make_json.py and create a json object for the test images. Make sure to replace the Dest_file.json with another name
- Open terminal , navigate to the tensorbox directory and run $jupyter notebook

- Run the evaluate.pynb file in the window that opens
- Replace the true_json file with your new json object
- Within the for loop remove the img= imread statement and replace it with
- img = imread('/%s' % true_anno.imageName)
- change the range of the for loop according to requirements



- click on *kernel -> restart and run all*

Fig7 *Jupyter Notebook*

Results

TensorFlow Object detection API:

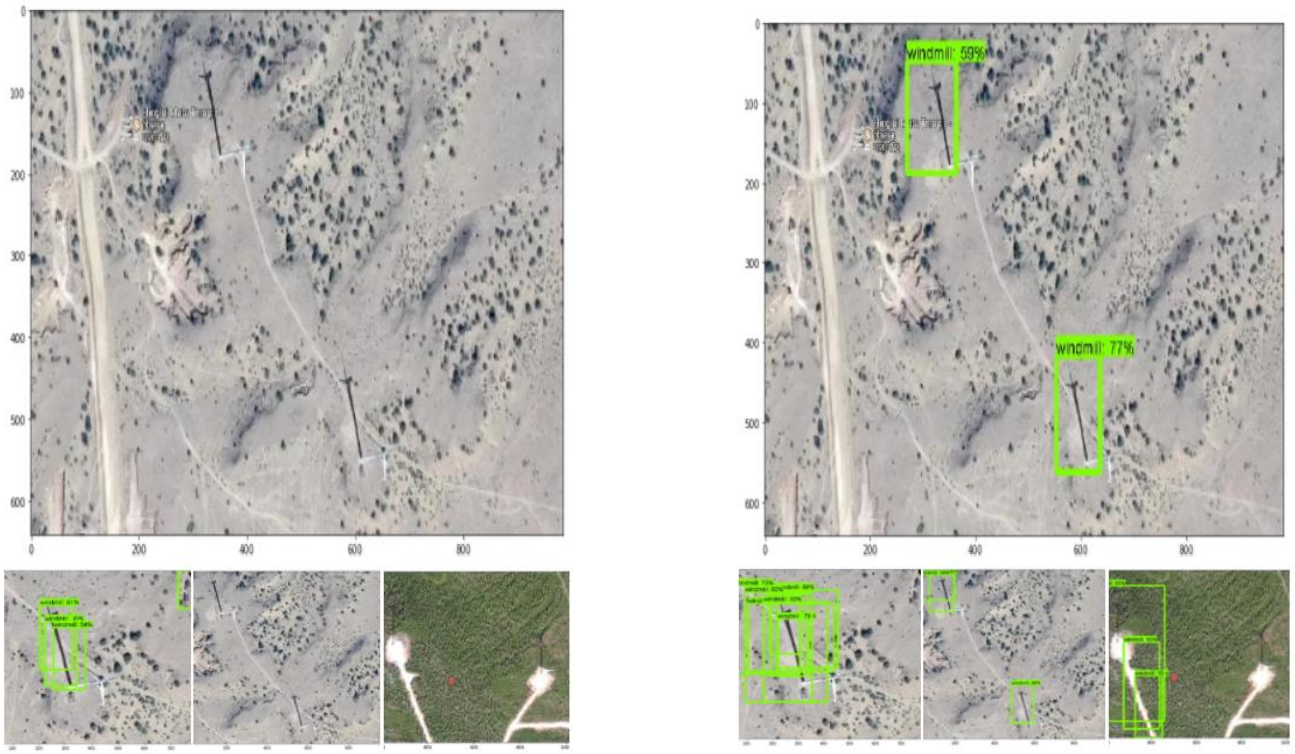


Fig8 Comparison between network output at 8000 iterations (left) and 18000 iterations (right)

As seen from Figure 8 the algorithm starts to detect the windmills and improves its accuracy over time.

Because of the lack of computing resources and networks high memory requirements we were partially able to train The Faster RCNN however the evolution of the network can be observed as in figure 8. (The evolution and

observations have been summarized in this blog post: https://thesilentmonksretreat.wordpress.com/2017/07/09/live-blog-training-an-object-detection-faster-rcnn-from-the-scratch/). The training was later focused on Tensorbox because of its robust nature and low computing requirements.

The Loss Graphs are as follows:

Fig9 *Loss Graphs for Tensorflow API*

TensorBox:

Since the model uses transfer learning and smaller architecture, it was much quicker as compared to Google API. The time taken per step has decreased considerably.On killing the training after 100000 iterations(7 hours), we found accurate results on images tested upon some of which have been shown below. It achieved accuracy of about 99% on the validation set which had been chosen. However in certain images we found random localizations which shows the need for larger dataset.Also notice how even with just a part of the shadow visible in the image it can detect the windmill with the actual windmill in the satellite image. It works succesful even in urban settings as shown in figure 11.Multiple windmills is also easily detected.

Another interesting result which we found out is that it gave accurately the results for windmills in grassy regions.This is confounding since we had trained our neural network only on dry arid regions of Rajasthan. This thus shows how successful it was in achieving the desired results and adds another dimension to the project in detecting windmills not just in the arid regions but for all regions.

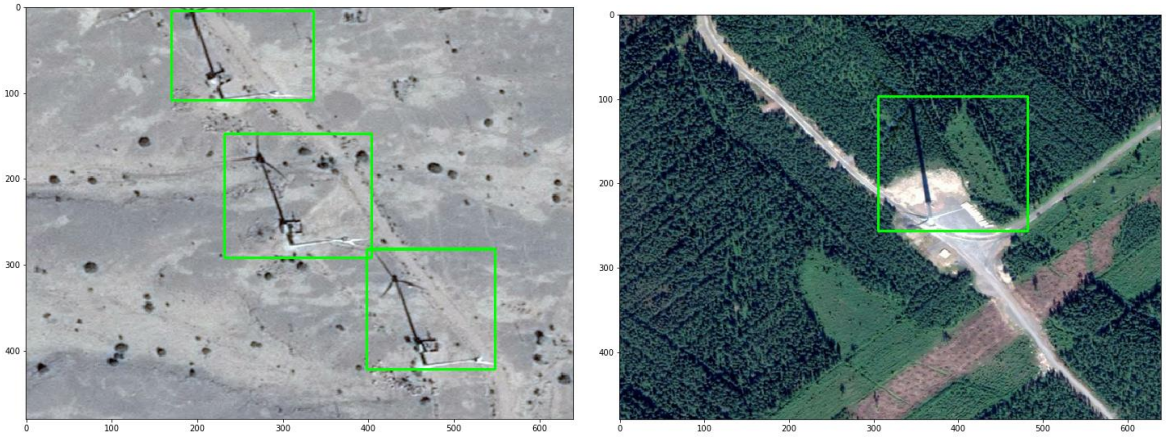


Fig10 *Network rendering output on random images*

Fig11 *Network rendering output on random images*

Further Applications

The outreach of this project is immense and has no bounds. With countries moving towards an era of computers solving every problem, this project could contribute towards solving quite a few of them. Increasing the data set to include a much greater variety of windmills would help in increasing its predicting power for a windmill placed in a very different background than what it has been trained for. Also we could assess a large area to scan for windmills by segmenting the image into smaller pieces and feeding them one by one into the network according to the limitations set by the GPU memory. This is of much practical significance as we can now assess exactly the number of windmills in our country and then make predictions on the energy produced by them as well as other analysis.This segmentation can be done using the sliding window approach wherein each segment of the big image isn't exclusive on its own but contains an area that is common with its adjacent segments. This will help detect better windmills that may be cut during the segmentation.

Training the network to detect different military vessels, a country can scan another country to assess their military power by gathering data of all openly placed vehicles.

This network if trained to detect rooftops can then detect different rooftops and suggest, taking data of the sun’s position , the best area to put a solar panel on the rooftop to gain maximal energy. We could also train the network to detect solar panels and calculate their current efficiency and predict areas where such panels can be placed where energy output is high.

References

- CSF2231n Stanford course
  - http://cs231n.stanford.edu/
  - http://cs231n.github.io/
- TensorBox
  - https://github.com/TensorBox/TensorBox
- Tensorflow API
  - https://github.com/tensorflow/models/tree/master/object_detection