Task Report

Ali Nehrani

1. Introduction

In this work I prepared two different deep learning codes to classify image data set with two classes. This is a binary classification problem in which the amount of data in two classes are equivalent. Binary classification is a common problem in the area of machine learning and there are challenging versions of them e.g. problems with harshly unbalanced data an detc. The common measure to analyse accuracy of the algorithms in binary classification is to use confusion matrix and ROC curve.

In the first code I developed a simple Convolutional Neural Network (CNN) which is state of the art in processing images.
In the second code I use pretrained vgg-16 code and fine-tuned it to make it proper for this binary classification context.

2. Image preparation

In order to make learning process more efficient I did some preprocessing over the images and transformed all of the image data into tfrecords which is a data format developed by google to make data feeding efficient in tensorflow. These preprocessings are very simple,

- Resize the images into 224*244*3
- Normalize the images
- Subtracting the overall average of all images to reduce background effects

In order to feed data to the networks in training and validation, I developed two structures; use tensorflow's queue runner and developing my own data feeding.
The first case is very efficient in which it is possible to use different threads and there is no data transferring between python and tensorflow. All data remained in tensorflow format (as tensors) and data is feeded directly to the network without using placeholders.
In my own data feeding (whish i used in pre-trained vgg16) I use pythons yield method and extracted the data as numpy arrays. I put placeholders on network and feeded them during the processes.
Trained parameters are saved as metafile during training and I load the meta (saves graph and all parameters) file to test the network.

Important results as well as the graph is written as tensorboard logs so it is possible to run tensorboard and see all of the results an analyse the training process.

In the following I described some details of the idea.

2. My network

I applied CNNs to classify the images into two classes (sushi and sandwich). The structure of my simple network consists of 3 convolutional layer followed by maxpolling and 2 dense layer. Which is very shallow to get good performance and I just put it to show the process.
I noticed that one class (sandwich) has far more variant the other class. One of the efficient techniques in this case is to apply autoencoders. Variational autoencoders are reported with very good performance in analysing the images. I developed a simple autoencoder to handle this issue.
I feed the data to auto encoder and network gets the encoded images in order to reduce that unbalanced variation. Auto encoders can also be used to make more data in deep learning and they can improve the performance significantly by producing noisy and variant to handle overfitting and underfitting the network.

Further developments

Developing complex network so that they can produce accurate results are not an easy task. One needs deep knowledge and lots of testing over the results and of course good resources (image and computation power). What I did is to show how we can develop CNNs to classify images.
It is possible to make highly efficient networks e.g. with batch normalizations, clever dropouts and etc. Also applying RNNs can lead to better results.

3. Transfer learning with Vgg16

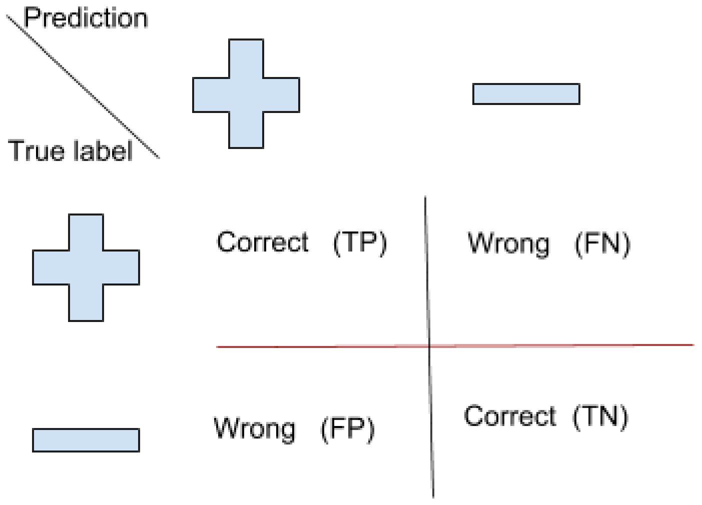
In applied point of view, now a days we have access to high efficient networks which are developed and trained for image classification purposes. We can use transfer learning, fine-tuning and also ensemble learning techniques to get use of those networks.
Among VGG is a heavy network which implemented and trained for classifying 1000 image classes. There are many others.
In this task I used vgg16 which belongs to family of VGG and fine tuned two dense layers of it to make it proper for two classes.

For further developing, we can get far better results by relearning last convolution layers also.
As the images that we are trying to classify are not among pre-trained but it is not very different from the feeding images.
Because of the similarity of our images to the trained images, I did not re-trained upper conv layers (if the images become very different we should retrain first layers as they extracting general features).
They last conv layers extracting detailed features and thus in our case it would be more efficient if we train the last two conv layers.
Another development is to transfer learning with other networks which were trained with similar images. There are other more efficients and high performance networks out there.

4. Analysis of the results

Analysing in binary classification is based on confusion matrix mainly and ROC curve in the case of unbalanced data.
We have two classes, let's call them positive and negative, therefore the results will have these conditions:

- correct prediction (positive predicted positive and negative predicted negative),
- the image is in positive class, we predicted negative,
- image is in negative classe, we predicted positive



Accuracy measures are developed based on these informations and by combining them we can get different measure each one is proper for specific problem and we can decide upon the problem. I'm not going to deep into this.
As is is show, at the beginning there is not difficult to use the measures but deciding where which one requires statistics and math knowledge.

I divided data into 3 parts: 80% for training, 10% for validation (to decide about hyperparameters) and 10% for testing (I use this data for representation instead but if I put accuracy measure there it can use for analysing) .
The measure that I used is the accuracy measure

$$\frac{TP + TN}{TP + FN + FP + TN}$$

Error measure is the complement of this but shows the error percentage

$$\frac{FP + FN}{TP + FN + FP + TN}$$

I directly reported the results in the validation set during training to see the performance on the validation set. For the test set which I developed in both codes, i did not included any measurement. But I visualized the result kind of representing the performance of the network.

Tensorflow provided matrix accuracy method which does the same job, for comparison I used this build in method.

The main accuracy measure that frequently can be used in binary classification is so called precision ration which is computed with the following formula:

$$precision = \frac{TP}{TP + FP}$$

The other measure is recall ration which is computed as

$$recall = \frac{TP}{TP + FN}$$

Precision and recall usually reported as 2d graph called P/R curves. Another frequently used metric is F1 metric which is indeed combines precision and recall in a single number:

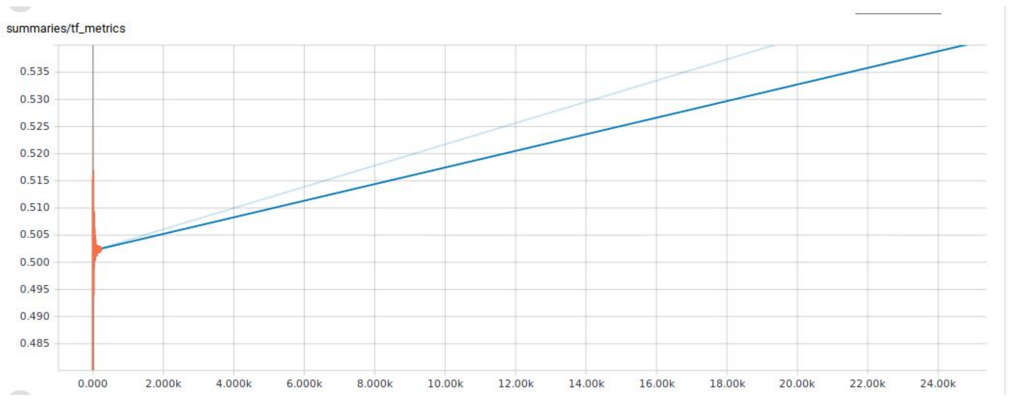
$$F1 = \frac{2 * precision * recall}{precision + recall}$$

Analysing based on results:
To see the process and get better understanding on what are going in the training process, I recorded results with tensorboard. Tensorboard writes the summaries in log files (I pointed it to write in log files placed in both algorithm’s folder).
I saw that the performance for my net is around 50% there are many reasons for this low accuracy; the hyperparameters (I tried very few due to time and limited resources), mount of data, shallow network structure and etc.
The best idea is to have look at the summaries. In the following i show some results and discussed on them.

Notting that the accuracy is measured over very low amount of data (batch with size 10) because of the limited memory that I have. This will not represent the true accuracy but it will be kind if sampling measurement of the accuracy which is a good approximation. This will help me to see whether the algorithms learns or not.

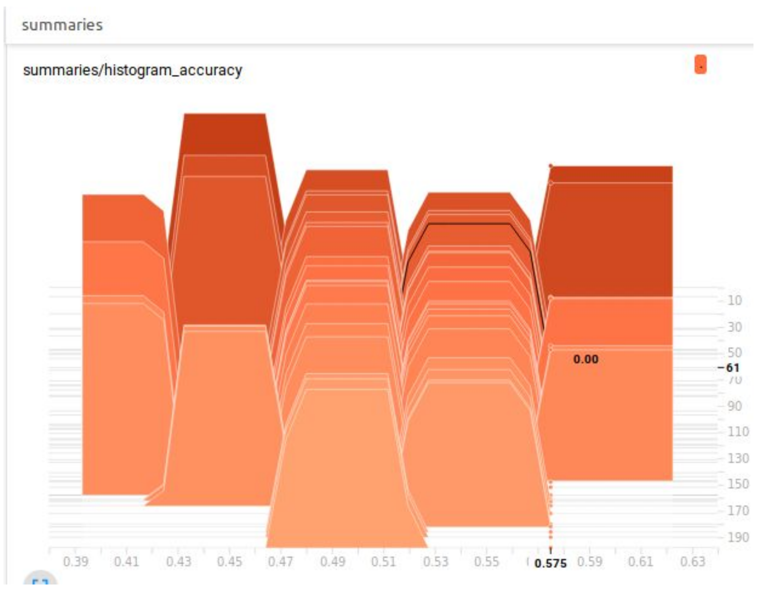
- **mynet**

The accuracy summary shows that during training accuracy is increased. however this increasing in the accuracy is not significant.



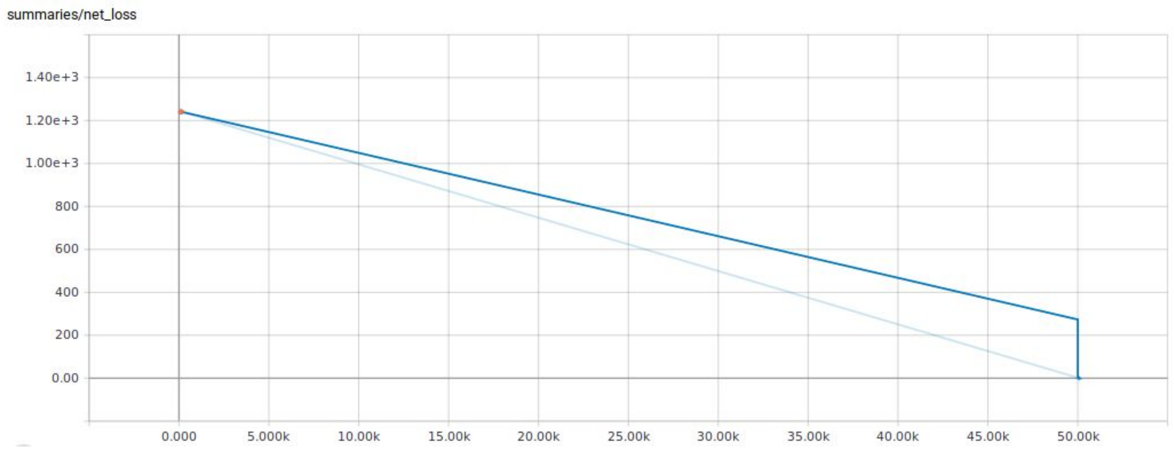
Accuracy summary for mynet

In the following I represent the histogram of the accuracy. As I told, the frequency of the accuracy is significantly in 0.47 - 0.53 range. We should notice that the accuracy is around 0.3 at the beginning and these are results for 34k of batch iterations that’s why we can not clearly see the accuracy improvement. In general this soft improvement is satisfying for this condition.

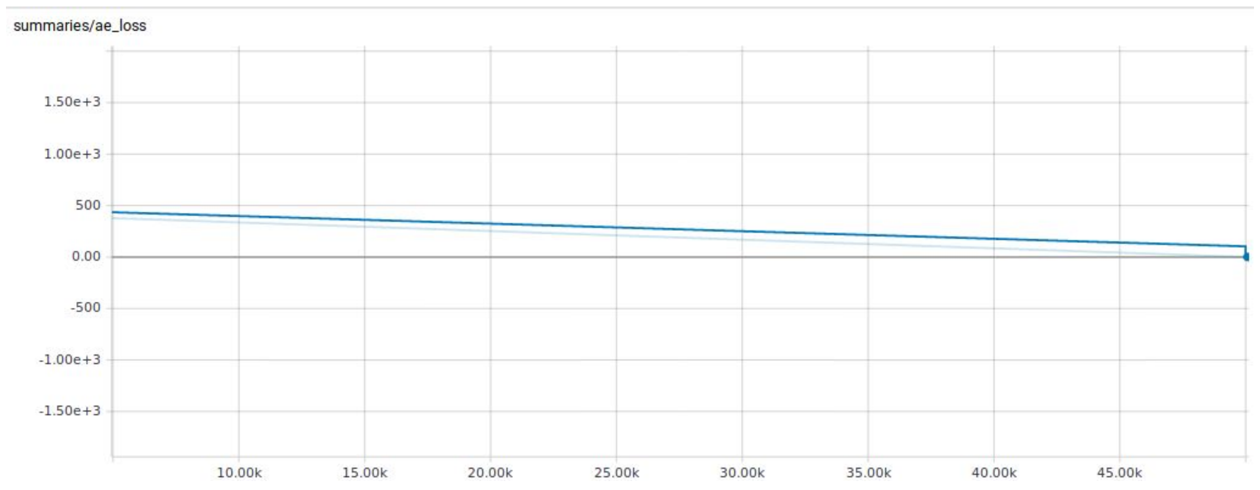


Accuracy histogram for mynet

I represent the loss function for autoencoder and the total loss here. These loss functions show that in general the algorithm learns during the time. The learning-rate that I selected is 0.9 with adam optimization which modifies the learning rate itself. As a result the learning rate is high at the beginning and it reduces dow which is normal.



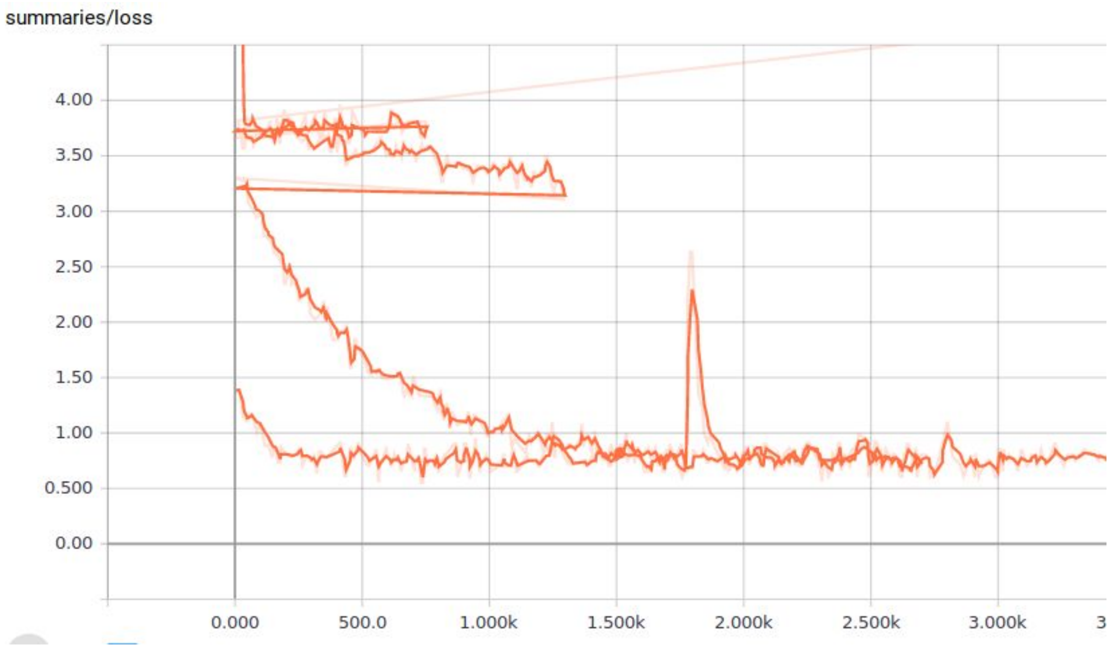
Loss summary for autoencoder in mynet



Total loss for mynet

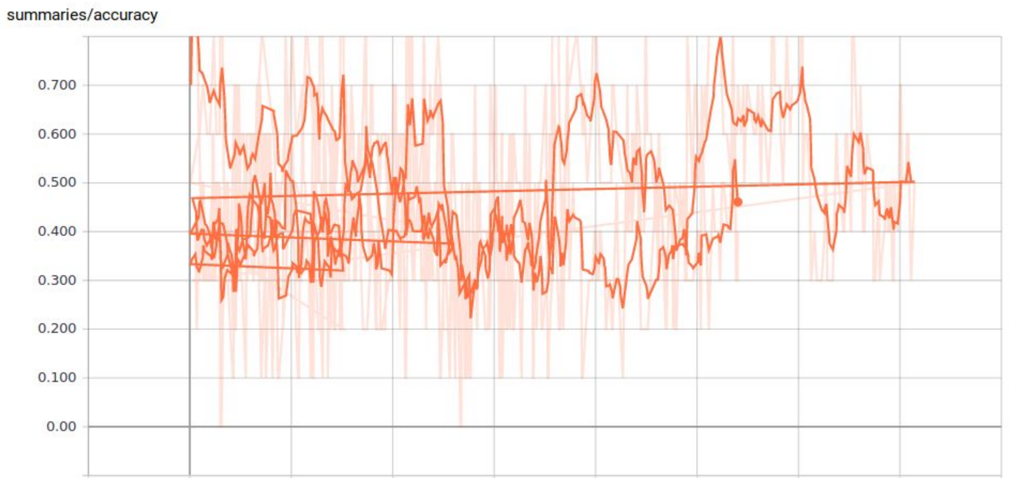
- **Pre-trained vgg16**

In the case of pretrained vgg16, loss function shows that this transfer learning works and it started to learn. Because of the structure of the loss function, the norm of it is low in general.
The repeating patterns in the graph are because I loaded the meta file and started the training process again. As the loss norm started from where it was left we can conclude that this saving and loading process also works fine.



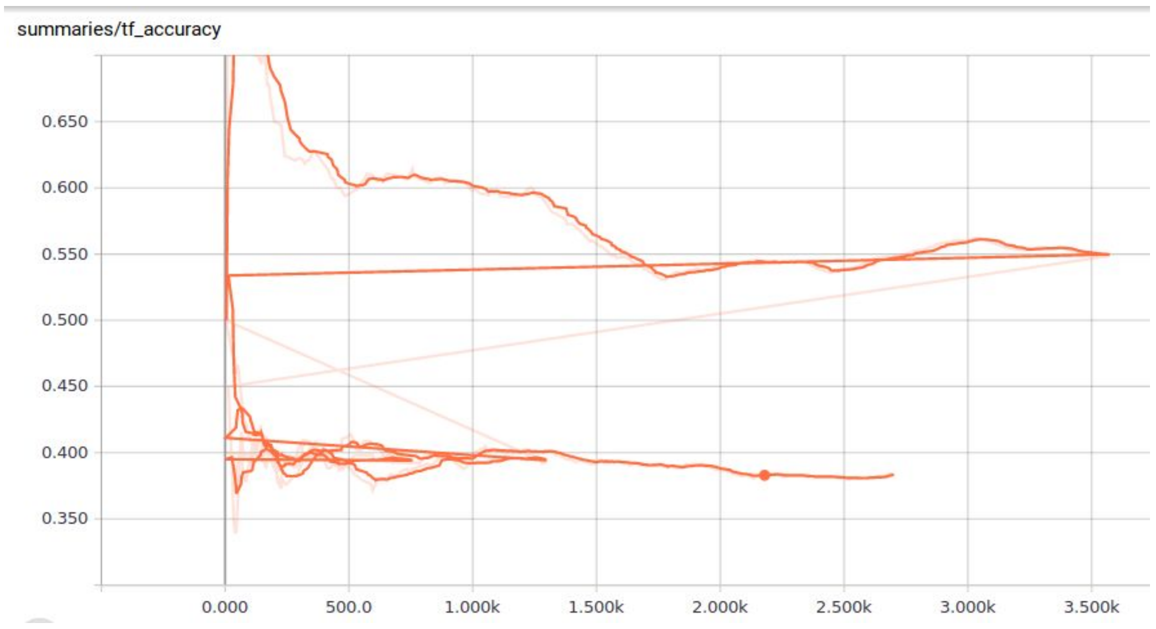
Loss for pre-trained vgg16

The accuracy measures showing that the tuning process could not increased the accuracy enough. The fluctuation is because I select samples randomly to measure the accuracy.



Accuracy for pre-trained vgg16

Tensorflow accuracy measure perfectly shows that during trained actually the accuracy is improved. That means the learning process works fine.



Accuracy with tf metrics for pre-trained vgg16

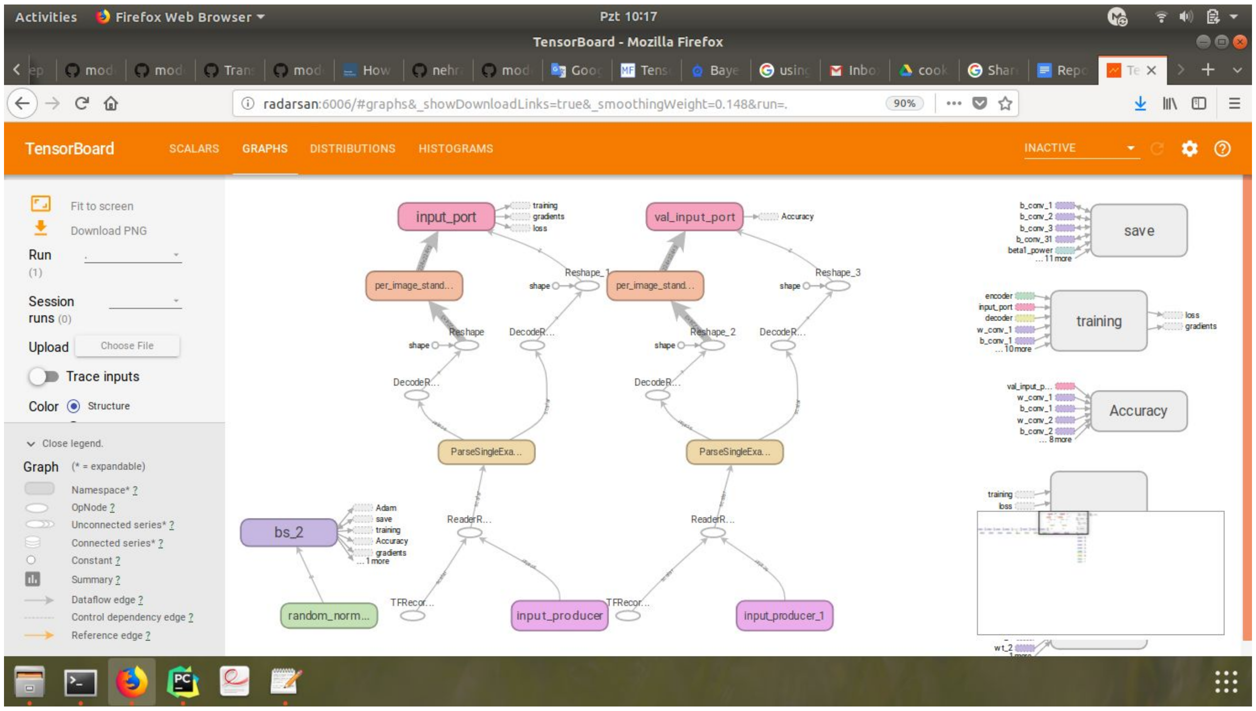
Tensorboard summaries as well as detailed graph can be accessed by running the command

```
>>     teorboard --logdir=./
```

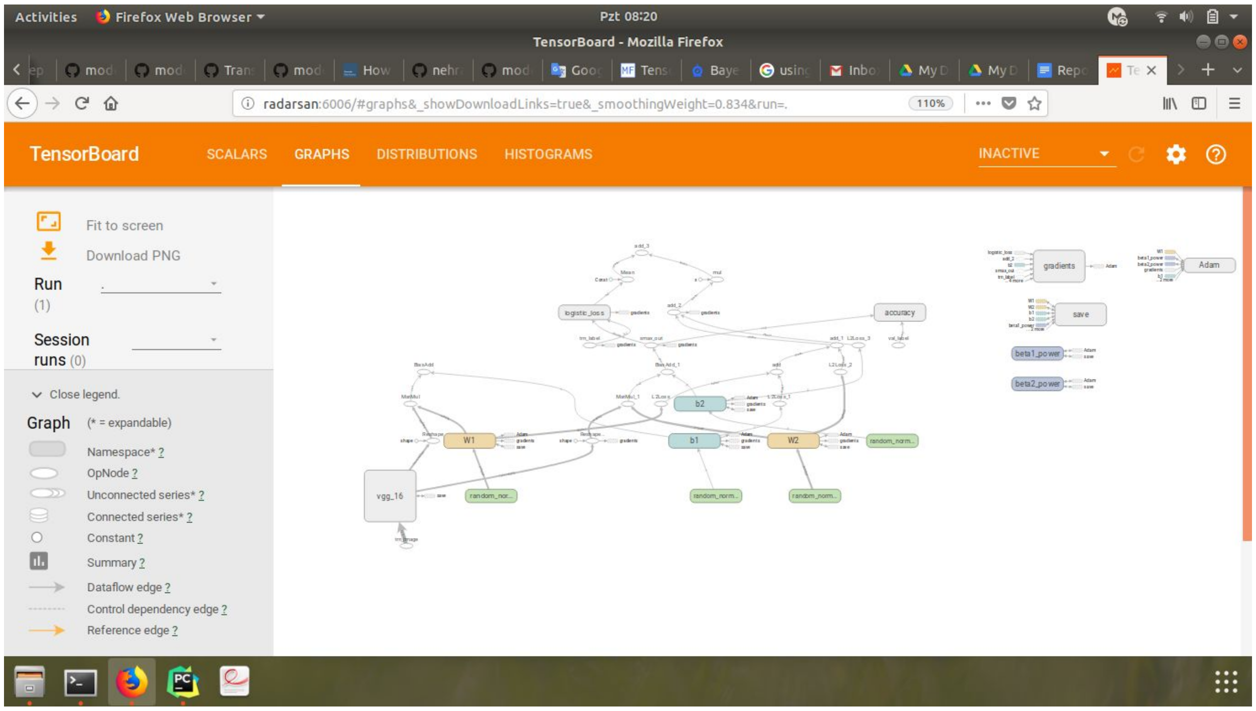
Then a link will appear, you can click on it and see the summary. For my case it is:

```
>> TensorBoard 1.8.0 at http://Radarsan:6006 (Press CTRL+C to quit)
```

The graphs structure for both networks are included here



Graph of mynet



Graph of fine-tuned vgg16

5. The provided files

In this task I applied tensorflow as deep learning framework and I did all of the process with it. In preprocessing images and writing them as tfrecords I use opencv and for presenting results on images I applied Matplolib and finally numpy. Therefore my codes does not require lots of packages to be installed.
If you already have them no other things is required and just running the codes will work.
Two different codes (approaches) are developed.
I called them "mynet" and "tuned_vgg16" (though it is actually not fine tuning). There are three folders:
data includes

- tfrecords; three files: train, validation and test data,
- a python file for transforming jpg images to tfrecords developed by myself

mynet includes:

- Checkpoint folder ("chkpnt") for storing checkpoints and metafile of trained network
- Log file ("log") to store tensorboard log files
- "Model_util.py" includes main network, auto encoder and other required parts of training and validation
- "Mynet_training.py": python file for training and validation of the network
- "Mynet_testing.py": python file for testing the network

Tuned_vgg16 includes:

- Log folders for tensorboard logs ('vgglog')
- "Chkpnt" to store checkpoints and meta file
- "Finetunning_vgg16.py" the main file which includes all required codes for training and testing developed by me
- "Vgg16" tensorflow implementation of VGG network from tensorflow slim

How to run the codes:

I developed two python codes for this purpose:

1. **"mynet_main.py"** to run mynet code
   The structure is very simple and easy to understand. For required parameters I use tensorflow log parser (tf.app.flags) so all parameters can be accessed by a general flag. I put short description of all parameters there.
   It is important to not that by the first parameters you can specify whether to train or directly go to test

2. **"vggnet_main.py"** to run vgg16 fine-tuned network
   The structure is the same as mynet case.

For simplicity I made the structure of these two codes very similar. I provided two separate functions for training and testing which can be selected by the parameters.
I developed the testing function so that it loads the network and shows some tests for some images (I chosen 10). I did not put accuracy measure here though it is very is to do that.
As a python file you are free to select the way of running the codes, e.g. with python shell etc.
Example usage of the provided files from python shell for mynet for training:

```
./mynet_main \
--logtostderr \
--train=True \
--test=False\
--showing_step=1
```

And for test just set test "True"

Example usage of the provided files from python shell for pretrained vgg16 for training:

```
./vggnet_main \
--logtostderr \
--train=True \
--test=False\
--showing_step=1
```

You can open the files "`mynet_main.py`" and "`vggnet_main.py`" with an editor and see all flags there.

6. Installing requirements

First of all I want you to run the codes on ubuntu which makes everything easier to handle. In order to install requirements, as there are few files, I used simple txt file to handle this.
The codes used tensorflow, numpy, opencv, matplotlib thats all!
This simply can be done as following:
On the place (or virtualenv) that you want to install, open a terminal and put "requirements.txt" into current directory then type:

```
>> python3 -m pip install -r requirements.txt
```

But these packages require some prerequisites. To install requirements I prepared prerequisites which can be installed with the following command from the python shell

```
>> apt-get install $(cat prerequisites.txt)
```

You may need root access which is with sudo

Final words
Developing high performing networks requires hours of works with proper resources, several times of running and testing different ideas to see which one works, checking everything, and making everything perfect. For this task I used my old laptop and it took long time for simple runs. Therefore I couldn't properly train the proposed networks.
Because of limited memory I chosen small batches for train and validation. This short batch for validation will not give good signal about the accuracy.

Thank you,
Ali