DOCUMENTATION

THIS IS AN EXPLANATION FOR THE IMPLENTATION AND RUNNING OF BOTH NEURAL NETWORKS.

BOTH MODELS REQUIRES TENSORFLOW.

1. TENSORFLOW INSTALATION
2. RE TRAINING (USING INCEPTION V3 FROM GOOGLE WITH TENSORFLOW)
3. CONVOLUTIONAL NEURAL NETWORK FROM SCRATH (USING TFLEARN LIBRARY FOR TENSORFLOW)

We put all the code that we show here and the model examples into a GitHub repository so everyone interested can download them.

https://github.com/fioritonicolas/cnn_machinelearning

TENSORFLOW INSTALATION

TensorFlow is an open source library for numerical computation, specializing in machine learning applications. In this inform we will show the steps of how to install and run TensorFlow on a single machine.

TensorFlow is develop in Python and run well in Python 2.7 or Python 3.3+.

To have python installed is the first step.

We are going to cover the installation with virtualenv. More information about virtual env in https://virtualenv.pypa.io. Basically virtual env are cleaning virtualizations from your python environment. So you can create them, and install in them all the program that you need without affecting your Global Python installation.

With Virtualenv the installation is as follows:

- Install pip and Virtualenv.
- Create a Virtualenv environment.
- Activate the Virtualenv environment and install TensorFlow in it.
- After the install you will activate the Virtualenv environment each time you want to use TensorFlow.

Install pip and Virtualenv:

```
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev python-virtualenv

# Mac OS X
$ sudo easy_install pip
$ sudo pip install --upgrade virtualenv
```

Create a virtual environment:

```
$ virtualenv --system-site-packages ~/tensorflow
```

Activate the virtual environment:

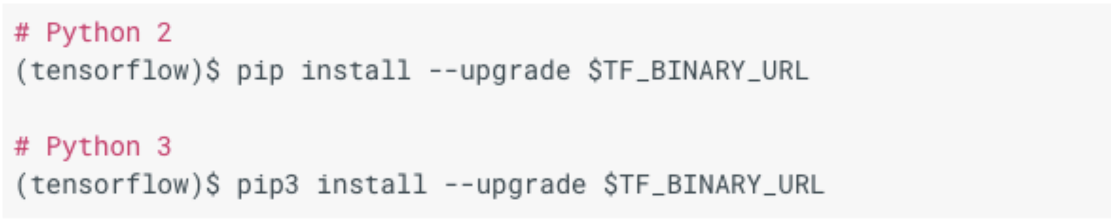
```
$ source ~/tensorflow/bin/activate  # If using bash
$ source ~/tensorflow/bin/activate.csh  # If using csh
(tensorflow)$  # Your prompt should change
```

Now we are going to do the pip installation but first we have to create an Envirment Variable with our OS version, so we run one of the correct from the list in this site https://www.tensorflow.org/get_started/os_setup#virtualenv_installation

In our case we use the mac one CPU only so:

```
# Mac OS X, CPU only, Python 2.7:
(tensorflow)$ export
TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/t
ensorflow-0.12.1-py2-none-any.whl
```

Finally install TensorFlow:

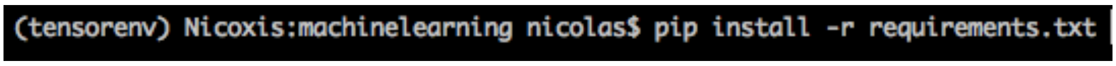


```
# Python 2
(tensorflow)$ pip install --upgrade $TF_BINARY_URL

# Python 3
(tensorflow)$ pip3 install --upgrade $TF_BINARY_URL
```

For the sake of the two models, we need also others libraries, the good thing is that using virtual env + pip just putting this lines:

```
funcsigs==1.0.2
h5py==2.6.0
mock==2.0.0
numpy==1.12.0
olefile==0.44
pbr==1.10.0
Pillow==4.0.0
protobuf==3.1.0
scipy==0.18.1
six==1.10.0
tensorflow==0.12.0rc1
tflearn==0.2.2
```

This lines represent all the other libraries used on this models, so we copy and paste them in a plain file, we name requirements.txt for convenience and then we run "pip install –r requirements.txt":



```
(tensorenv) Nicoxis:machinelearning nicolas$ pip install -r requirements.txt
```

That's all we need to set up the environment.

RETRAINING WITH INCEPTION V3

With a data set of five thousand images from men and women this Model reach and accuracy of 95.7%

This model it takes advantages from a pre trained model. This is a Convolutional Neural Network with a big architecture which has been trained with millions of images. So we don't need to start from scratch we just need to re-used this architecture. What we are going to do in this example take advantage of this model and used all the features and patterns detection already detected and start from there.

We are going to change the last layer so the fully connected network classified according our features and re trained we a new data set from there.

We just need to download from GitHub this code:

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining

And the file that we are going to use is the retrain.py. Having this file is pretty much everything. Now we just need to prepare our pictures is the follow way:

directory_contains_labeled_directorys

->label_directory_1

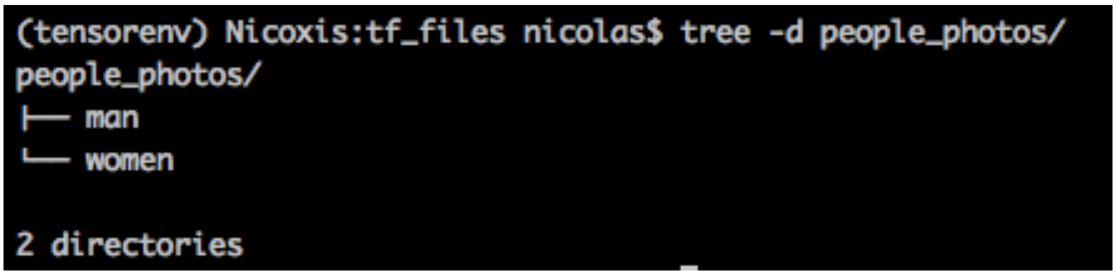
-> thousands of pictures in category labeled as his parent directory

->label_directory_2

-> thousands of pictures in category labeled as his parent directory

That is the structure we need to respect..

In our example.



```
(tensorenv) Nicoxis:tf_files nicolas$ tree -d people_photos/
people_photos/
├── man
└── women

2 directories
```

So for our gender classifier we just need the directory man with all the man pictures and directory woman with all woman pictures.

NOTE: there are not requirements for the name of the photos.

Once we have this directory proper set up, we just need to start the training to do that we just run the follow command:

```
python retrain.py \
--bottleneck_dir=./tf_files/bottlenecks \
--how_many_training_steps 800 \
--model_dir=./tf_files/inception \
--output_graph=./tf_files/retrained_graph.pb \
--output_labels=./tf_files/retrained_labels.txt \
--image_dir ./tf_files/people_photos
```

So lets explain a the parameters:

--bottlenecks_dir:we need to specify a directory to store the bottle necks this will save a processing stage of every image as a txt. Really handy if we want to add more images latter on and keep training our model

--how_many_training_steps: it explain by itself

--model_dir;this is the directory where we have or we want to download the inception model if we already have it. The algorithm will use it if not it will download and then use it.

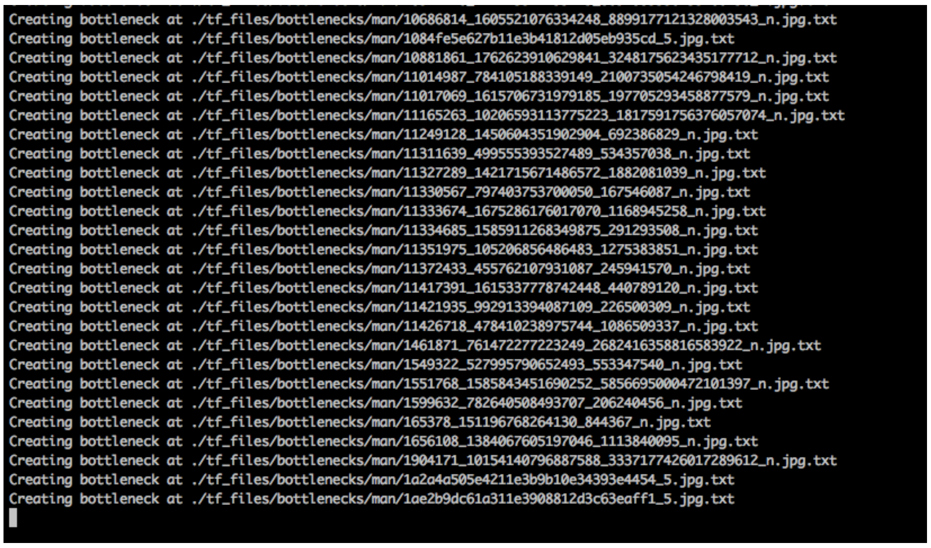
--output_graph; where to store our trainen model

--output_label=this is a txt we a resume of the label extracted from the directorys

--image_dir=the directory where we put the photos.

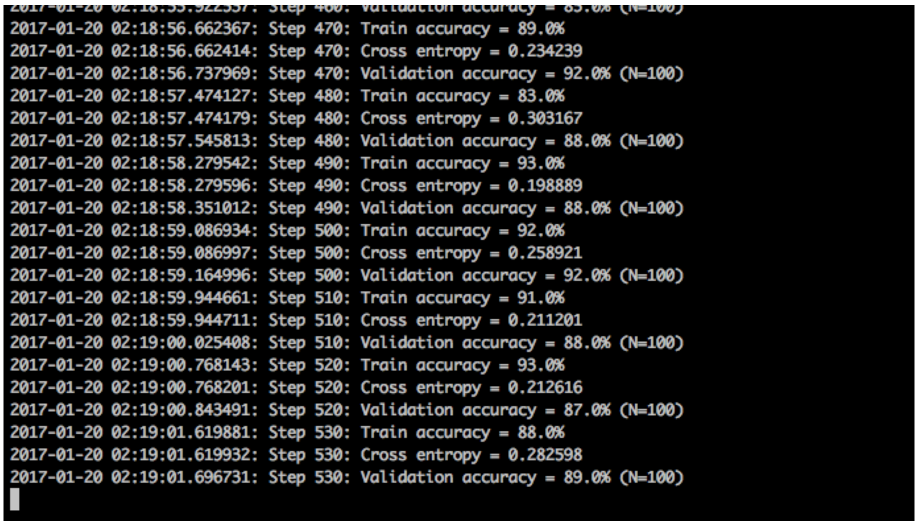
With al the parameters set up correctly, next step is run it.

First time it will calculates every bottleneck for every images and then it will start the training



```
Creating bottleneck at ./tf_files/bottlenecks/man/10686814_1605521076334248_8899177121328003543_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1084fe5e627b11e3b41812d05eb935cd_5.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/10881861_1762623910629841_3248175623435177712_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11014987_784105188339149_2100735054246798419_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11017069_1615706731979185_1977052934588775579_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11165263_10206593113775223_1817591756376057074_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11249128_1450604351902904_692386829_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11311639_499555393527489_534357038_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11327289_1421715671486572_1882081039_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11330567_797403753700050_167546087_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11333674_1675286176017070_1168945258_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11334685_1585911268349875_291293508_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11351975_105206856486483_1275383851_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11372433_455762107931087_245941570_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11417391_1615337778742448_440789120_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11421935_992913394087109_226500309_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/11426718_478410238975744_1086509337_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1461871_761472277223249_2682416358816583922_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1549322_527995790652493_553347540_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1551768_1585843451690252_5856695000472101397_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1599632_782640508493707_206240456_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/165378_151196768264130_844367_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1656108_1384067605197046_1113840095_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1904171_10154140796887588_3337177426017289612_n.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1a2a4a505e4211e3b9b10e34393e4454_5.jpg.txt
Creating bottleneck at ./tf_files/bottlenecks/man/1ae2b9dc61a311e3908812d3c63eaff1_5.jpg.txt
```

Training:



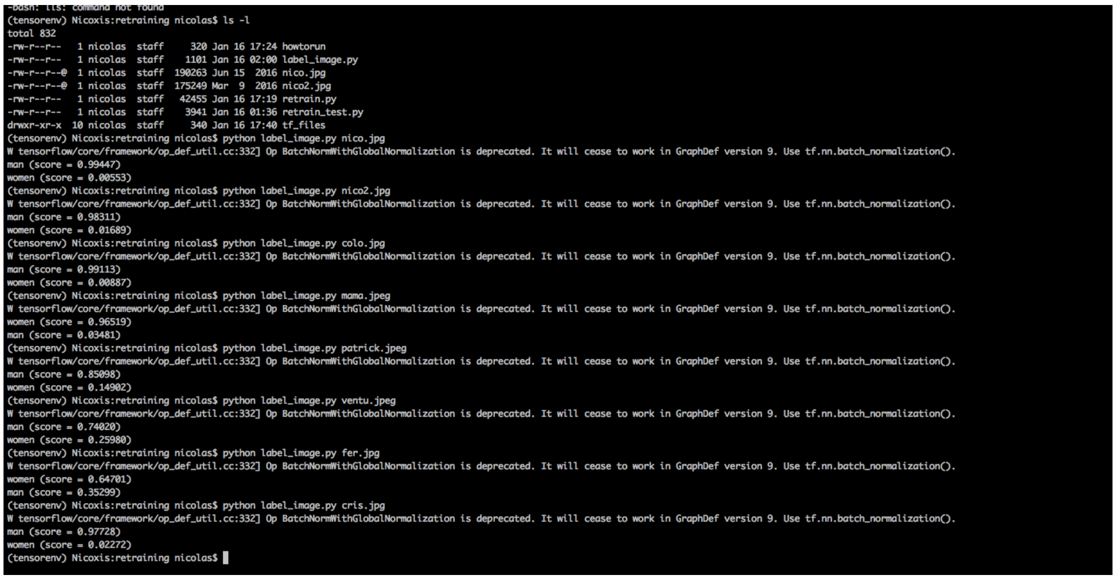
```
2017-01-20 02:18:55.922337: Step 460: Validation accuracy = 85.0% (N=100)
2017-01-20 02:18:56.662367: Step 470: Train accuracy = 89.0%
2017-01-20 02:18:56.662414: Step 470: Cross entropy = 0.234239
2017-01-20 02:18:56.737969: Step 470: Validation accuracy = 92.0% (N=100)
2017-01-20 02:18:57.474127: Step 480: Train accuracy = 83.0%
2017-01-20 02:18:57.474179: Step 480: Cross entropy = 0.303167
2017-01-20 02:18:57.545813: Step 480: Validation accuracy = 88.0% (N=100)
2017-01-20 02:18:58.279542: Step 490: Train accuracy = 93.0%
2017-01-20 02:18:58.279596: Step 490: Cross entropy = 0.198889
2017-01-20 02:18:58.351012: Step 490: Validation accuracy = 88.0% (N=100)
2017-01-20 02:18:59.086934: Step 500: Train accuracy = 92.0%
2017-01-20 02:18:59.086997: Step 500: Cross entropy = 0.258921
2017-01-20 02:18:59.164996: Step 500: Validation accuracy = 92.0% (N=100)
2017-01-20 02:18:59.944661: Step 510: Train accuracy = 91.0%
2017-01-20 02:18:59.944711: Step 510: Cross entropy = 0.211201
2017-01-20 02:19:00.025408: Step 510: Validation accuracy = 88.0% (N=100)
2017-01-20 02:19:00.768143: Step 520: Train accuracy = 93.0%
2017-01-20 02:19:00.768201: Step 520: Cross entropy = 0.212616
2017-01-20 02:19:00.843491: Step 520: Validation accuracy = 87.0% (N=100)
2017-01-20 02:19:01.619881: Step 530: Train accuracy = 88.0%
2017-01-20 02:19:01.619932: Step 530: Cross entropy = 0.282598
2017-01-20 02:19:01.696731: Step 530: Validation accuracy = 89.0% (N=100)
```

Once we have our model trained next step is to run a script which used this model receive an image and do the classification.

The code to run the label image was taken from a google code lab and adapted a little we upload it on the GitHub project as /retraingin/label_image.py

https://github.com/fioritonicolas/cnn_machinelearning/blob/master/retraining/label_image.py

Once we have this file we check that is pointing where to our model, and next step is run it. It takes one parameter and is the image to be processed.



```
-bash: lls: command not found
(tensorenv) Nicoxis:retraining nicolas$ ls -l
total 832
-rw-r--r--   1 nicolas  staff     320 Jan 16 17:24 howtorun
-rw-r--r--   1 nicolas  staff    1101 Jan 16 02:00 label_image.py
-rw-r--r--@  1 nicolas  staff  190263 Jun 15  2016 nico.jpg
-rw-r--r--@  1 nicolas  staff  175249 Mar  9  2016 nico2.jpg
-rw-r--r--   1 nicolas  staff   42455 Jan 16 17:19 retrain.py
-rw-r--r--   1 nicolas  staff    3941 Jan 16 01:36 retrain_test.py
drwxr-xr-x  10 nicolas  staff     340 Jan 16 17:40 tf_files
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py nico.jpg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.99447)
women (score = 0.00553)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py nico2.jpg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.98311)
women (score = 0.01689)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py colo.jpg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.99113)
women (score = 0.00887)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py mama.jpeg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
women (score = 0.96519)
man (score = 0.03481)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py patrick.jpeg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.85098)
women (score = 0.14902)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py ventu.jpeg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.74020)
women (score = 0.25980)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py fer.jpg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
women (score = 0.64701)
man (score = 0.35299)
(tensorenv) Nicoxis:retraining nicolas$ python label_image.py cris.jpg
W tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is deprecated. It will cease to work in GraphDef version 9. Use tf.nn.batch_normalization().
man (score = 0.97728)
women (score = 0.02272)
(tensorenv) Nicoxis:retraining nicolas$
```

CONVOLUTIONAL NEURAL NETWORK FROM SCRATH (USING TFLEARN LIBRARY FOR TENSORFLOW)

With a data set of five thousand images from men and women this Model reach and accuracy of 89%. Not that accurate as the re-trained example but not bad to be a model from the scratch.

In this example we are going to use TFLearn which is an abstraction of tensor flow, which help us with al the session and instance creation so we can take more care of the architecture of our networks.

If you follow step one you already have TFLearn in because of the requirements from pip was written so it's already installed.

The firs different thing in compare with the other is that we have to write code by ourselves. And before getting into the network we need the preparation for the dataset.

This could be quite messy so we created a python file which imitates the way of work from the re-training example and we can put all or our data into directory and subdirectories and it will create H5 compressed datasets already labelled.

So first step is to run this file which we named prepare_dataset.py, is here

https://github.com/fioritonicolas/cnn_machinelearning/blob/master/cnntflearn/prepare_dataset.py

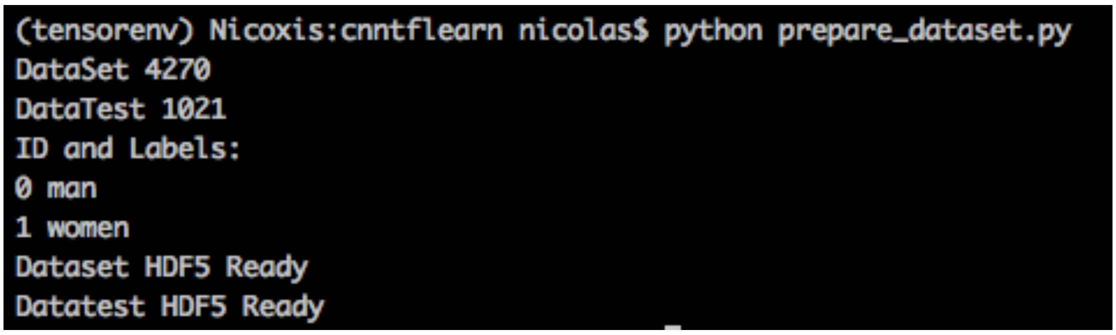
Here we need to set some parameters:

path: Is a string representing the path to the parent directory which is storing the subdirectories with the photos.

percentaje_of_data_to_test: this a percentage of the data inside our folder that we want to put as dataset (0 - 1)

We need to make to have a dataset folder create at the same level of the script because is going to create our prepared dataset there.

So we run it:


```
(tensorenv) Nicoxis:cnntflearn nicolas$ python prepare_dataset.py
DataSet 4270
DataTest 1021
ID and Labels:
0 man
1 women
Dataset HDF5 Ready
Datatest HDF5 Ready
```

It will prompt us here and also store in dataset folder, the dataset to use and a label.txt file to remind which number is which label, In our case 0 man, 1 woman

Now we are ready to run our classifier:

https://github.com/fioritonicolas/cnn_machinelearning/blob/master/cnntflearn/convnet_tflearn.py

Here we put some lines from the code:

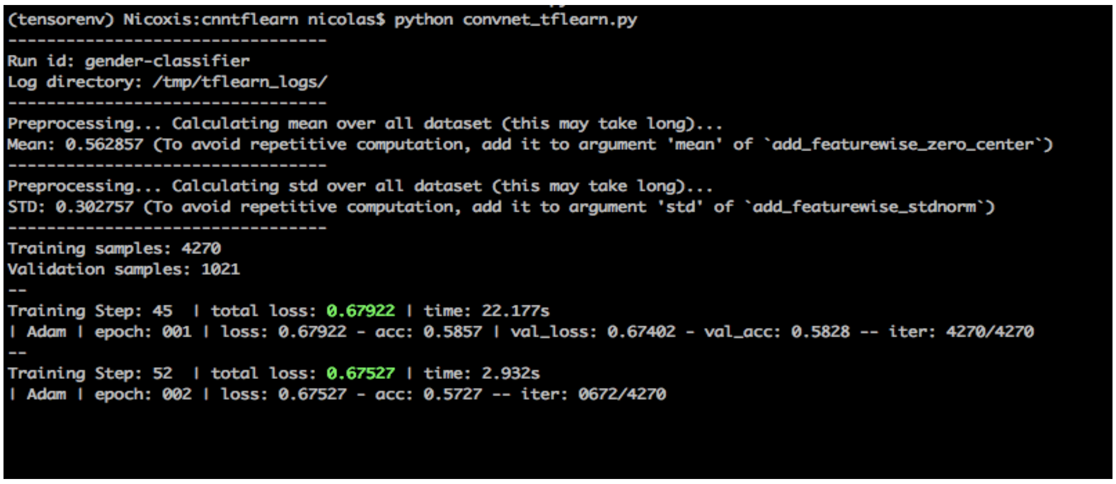
```
# Input is a 32x32 image with 3 color channels (red, green and blue)
network = input_data(shape=[None, 32, 32, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug)

# Step 1: Convolution
network = conv_2d(network, 32, 3, activation='relu')

# Step 2: Max pooling
network = max_pool_2d(network, 2)

# Step 3: Convolution again
network = conv_2d(network, 64, 3, activation='relu')

# Step 4: Convolution yet again
network = conv_2d(network, 64, 3, activation='relu')

# Step 5: Max pooling again
network = max_pool_2d(network, 2)

# Step 6: Fully-connected 512 node neural network
network = fully_connected(network, 512, activation='relu')

# Step 7: Dropout - throw away some data randomly during training to
prevent over-fitting
network = dropout(network, 0.5)

# Step 8: Fully-connected neural network with two outputs (0=isn't a
bird, 1=is a bird) to make the final prediction
network = fully_connected(network, 2, activation='softmax')

# Tell tflearn how we want to train the network
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.001)
```

```
# Wrap the network in a model object
model = tflearn.DNN(network, tensorboard_verbose=0,
checkpoint_path='gender-classifier.tfl.ckpt')
```

Basically is building the network architecture with all the convolutional steps and the max pooling.

One important parameter is snapshot_epoch inside the model.fit method. If it is set to True it will create a model every epoch, this can be handy when we already reach a nice accuracy so we can stop the training. But also if we stop the training we need to used that last snapshot in our case the name of that model will be something like gender-classifier.tfl.ckpt-NNNN where N the number of training steps reached.

To run this file, we do python convent_tfleanr.py



```
(tensorenv) Nicoxis:cnntflearn nicolas$ python convnet_tflearn.py
---------------------------------
Run id: gender-classifier
Log directory: /tmp/tflearn_logs/
---------------------------------
Preprocessing... Calculating mean over all dataset (this may take long)...
Mean: 0.562857 (To avoid repetitive computation, add it to argument 'mean' of `add_featurewise_zero_center`)
---------------------------------
Preprocessing... Calculating std over all dataset (this may take long)...
STD: 0.302757 (To avoid repetitive computation, add it to argument 'std' of `add_featurewise_stdnorm`)
---------------------------------
Training samples: 4270
Validation samples: 1021
--
Training Step: 45  | total loss: 0.67922 | time: 22.177s
| Adam | epoch: 001 | loss: 0.67922 - acc: 0.5857 | val_loss: 0.67402 - val_acc: 0.5828 -- iter: 4270/4270
--
Training Step: 52  | total loss: 0.67527 | time: 2.932s
| Adam | epoch: 002 | loss: 0.67527 - acc: 0.5727 -- iter: 0672/4270
```

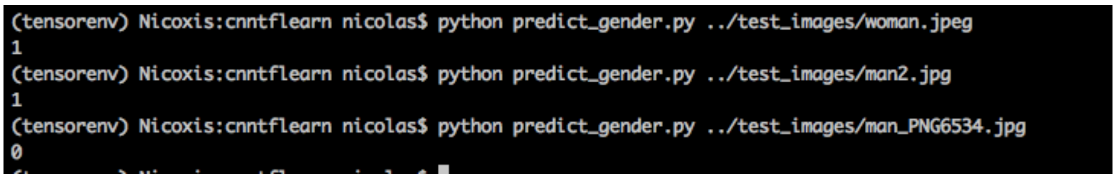
When we are finish with the learning we can used that model to do the prediction.

We need to pay attention to the parem

The file to run the model is also in GitHub:

https://github.com/fioritonicolas/cnn_machinelearning/blob/master/cnntflearn/predict_gender.py

Here is were we need to pay attention to the name of the model in case we stop the training. It also take an image as a parameter so:
python predict_gender.py my_pic_to_classify.jpg will do the classification.



```
(tensorenv) Nicoxis:cnntflearn nicolas$ python predict_gender.py ../test_images/woman.jpeg
1
(tensorenv) Nicoxis:cnntflearn nicolas$ python predict_gender.py ../test_images/man2.jpg
1
(tensorenv) Nicoxis:cnntflearn nicolas$ python predict_gender.py ../test_images/man_PNG6534.jpg
0
```

We see here it will prompt 1 if is a woman and 0 if is a man, and with some pictures like man2.jpg will make mistake because is not that accurate.