Douglas Wong
Tanveer Bariana

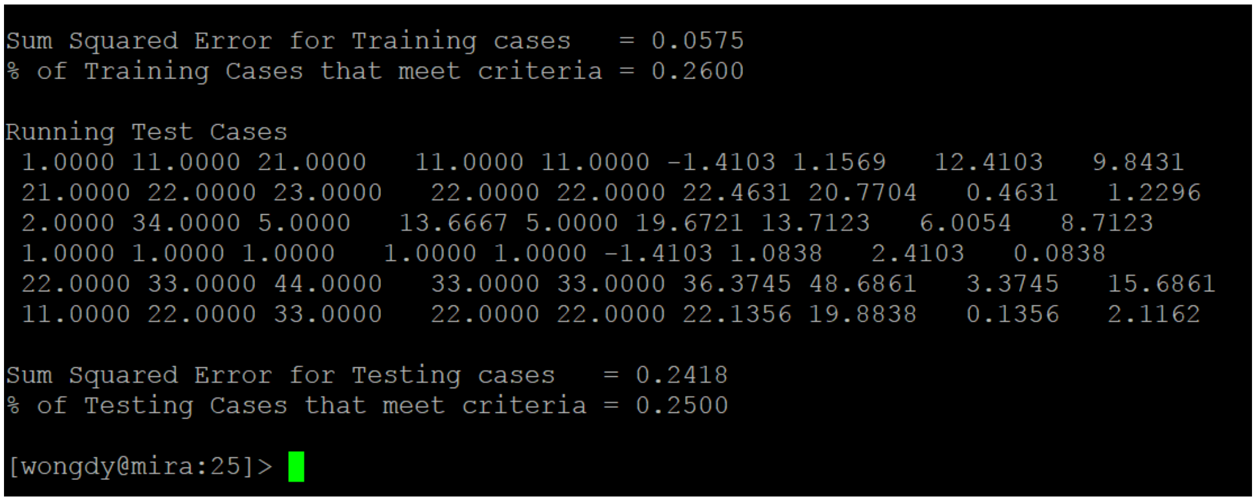
Project 3: deep-douggo

Part 1:

* the settings that you used for the neural network configuration

```
// NN parameters  ------------------------------------------------
#define NumINs       3       // number of inputs, not including bias node
#define NumOUTs      2       // number of outputs, not including bias node
#define Criteria     0.3      // all training outputs must be within this for training to stop
#define TestCriteria 0.5               // all testing outputs must be within this for generalization

#define LearningRate 0.75     // most books suggest 0.3 as a starting point
#define Momentum     0.3    // must be >=0 and <1
#define bias         -1.0   // output value of bias node (usually 1, sometimes -1 for sigmoid)
#define weightInit   0.5     // weights are initialized randomly with this max magnitude
#define MaxIterate   10000000 // maximum number of iterations before giving up training
#define ReportIntv   100001    // print report every time this many training cases done

// network topology ----------------------------------------------
#define NumNodes1    4       // col 1 - must equal NumINs+1 (extra node is bias node)
#define NumNodes2    8       // col 2 - hidden layer 1, etc.
#define NumNodes3    5       // output layer is last non-zero layer, and must equal NumOUTs
#define NumNodes4    4       // note - last node in input and hidden layers will be used as bias
#define NumNodes5    3       // note - there is no bias node in the output later
#define NumNodes6    2
#define Activation1    0     // use activation=0 for input (first) layer and for unused laters
#define Activation2    4     // Specify desired activation function for hidden and output layers
#define Activation3    1     // 1=sig, 2=tanh, 3=relu, 4=leakyRelu, 5=linear
#define Activation4    4
#define Activation5    1
#define Activation6    1
#define NumOfCols    6       // number of non-zero layers specified above, including the input layer
#define NumOfRows    8       // largest layer - i.e., max number of rows in the network

// data files ----------------------------------------------
#define TrainFile    "data1.dat"  // file containing training data
#define TestFile     "data2.dat"  // file containing testing data
#define TrainCases   25           // number of training cases
#define TestCases    6            // number of test cases
```

* a summary of results:



```
Sum Squared Error for Training cases    = 0.0575
% of Training Cases that meet criteria = 0.2600

Running Test Cases
 1.0000 11.0000 21.0000   11.0000 11.0000 -1.4103 1.1569   12.4103   9.8431
 21.0000 22.0000 23.0000   22.0000 22.0000 22.4631 20.7704   0.4631   1.2296
 2.0000 34.0000 5.0000   13.6667 5.0000 19.6721 13.7123   6.0054   8.7123
 1.0000 1.0000 1.0000   1.0000 1.0000 -1.4103 1.0838   2.4103   0.0838
 22.0000 33.0000 44.0000   33.0000 33.0000 36.3745 48.6861   3.3745   15.6861
 11.0000 22.0000 33.0000   22.0000 22.0000 22.1356 19.8838   0.1356   2.1162

Sum Squared Error for Testing cases    = 0.2418
% of Testing Cases that meet criteria = 0.2500

[wongdy@mira:25]>
```

Training cases:

Sum Squared Error for Training cases = 0.0575
% of Training Cases that meet criteria = 0.2600

Testing cases:
Sum Squared Error for Testing cases = 0.2418
% of Testing Cases that meet criteria = 0.2500

The training cases precision that we set is at 0.3
And the takes ~1mins to train since the MaxIterate is at 10000000
We have another version where the precision is at 0.01, and the max iterate is at 1 billion, and it takes 20mins and the result is 0 since they couldn't met the 0.01 precision. Since it didn't able to meet the criteria, we didn't use this setting at all. We also add a 0.3 momentum, and we experienced that adding the momentum only helped on the training set, it decrease 1.0000 to 0.2600, which we think is more accurate because if we have 1.0 in the training, why wouldn't have a high testing percentage, thus we add 0.3 momentum into the parameters.

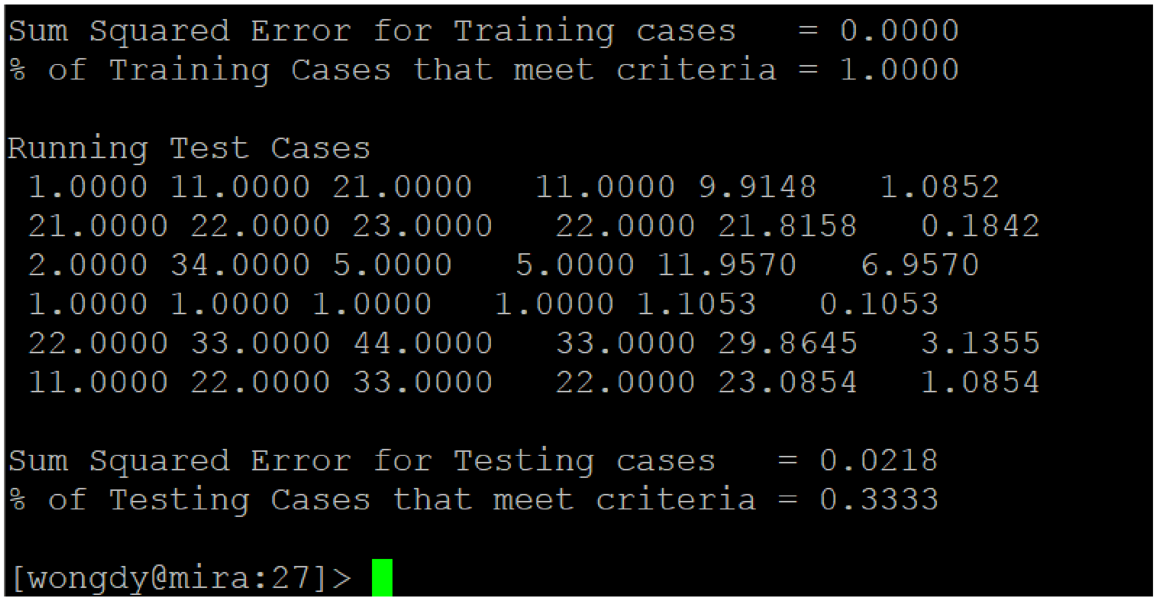
***Conclusion**
In conclusion of part 1, we would say that it really hard to get the testing and the training success rate at 1 or to get it close to 0.8 because the are 2 output where it will take a longer iterator to train which means more time and also it will require a lot more data set, we have tried it once, but because to takes too long and the admin killed our process on athena. We then switch our directions, instead of having a large data set and longer iterator period to train, we change the layers and the numNodes especially the hidden layer, and the bias such that we can have 4 hidden and output layers. Through the process of tweaking the values, and this is the most stable and best numbers we can get.

Part 2:

```
// NN parameters  ------------------------------------------------
#define NumINs       3       // number of inputs, not including bias node
#define NumOUTs      1       // number of outputs, not including bias node
#define Criteria     0.1      // all training outputs must be within this for training to stop
#define TestCriteria 0.5    // all testing outputs must be within this for generalization

#define LearningRate 0.75     // most books suggest 0.3 as a starting point
#define Momentum     0.3    // must be >=0 and <1
#define bias         -1.0   // output value of bias node (usually 1, sometimes -1 for sigmoid)
#define weightInit   0.5     // weights are initialized randomly with this max magnitude
#define MaxIterate   10000000 // maximum number of iterations before giving up training
#define ReportIntv   100001    // print report every time this many training cases done

// network topology ----------------------------------------------
#define NumNodes1    4       // col 1 - must equal NumINs+1 (extra node is bias node)
#define NumNodes2    8       // col 2 - hidden layer 1, etc.
#define NumNodes3    5      // output layer is last non-zero layer, and must equal NumOUTs
#define NumNodes4    4       // note - last node in input and hidden layers will be used as bias
#define NumNodes5    2       // note - there is no bias node in the output later
#define NumNodes6    1
#define Activation1    0     // use activation=0 for input (first) layer and for unused laters
#define Activation2    4     // Specify desired activation function for hidden and output layers
#define Activation3    1     // 1=sig, 2=tanh, 3=relu, 4=leakyRelu, 5=linear
#define Activation4    4
#define Activation5    1
#define Activation6    1
#define NumOfCols    6       // number of non-zero layers specified above, including the input layer
#define NumOfRows    8       // largest layer - i.e., max number of rows in the network

// data files -------------------------------------------------|
#define TrainFile    "data5.dat"  // file containing training data
#define TestFile     "data6.dat"  // file containing testing data
#define TrainCases   25           // number of training cases
#define TestCases    6            // number of test cases
```

*** a summary of results:**

3 input, 1 output(median)



```
Sum Squared Error for Training cases    = 0.0000
% of Training Cases that meet criteria = 1.0000

Running Test Cases
 1.0000 11.0000 21.0000   11.0000 9.9148   1.0852
 21.0000 22.0000 23.0000   22.0000 21.8158   0.1842
 2.0000 34.0000 5.0000   5.0000 11.9570   6.9570
 1.0000 1.0000 1.0000   1.0000 1.1053   0.1053
 22.0000 33.0000 44.0000   33.0000 29.8645   3.1355
 11.0000 22.0000 33.0000   22.0000 23.0854   1.0854

Sum Squared Error for Testing cases    = 0.0218
% of Testing Cases that meet criteria = 0.3333

[wongdy@mira:27]>
```

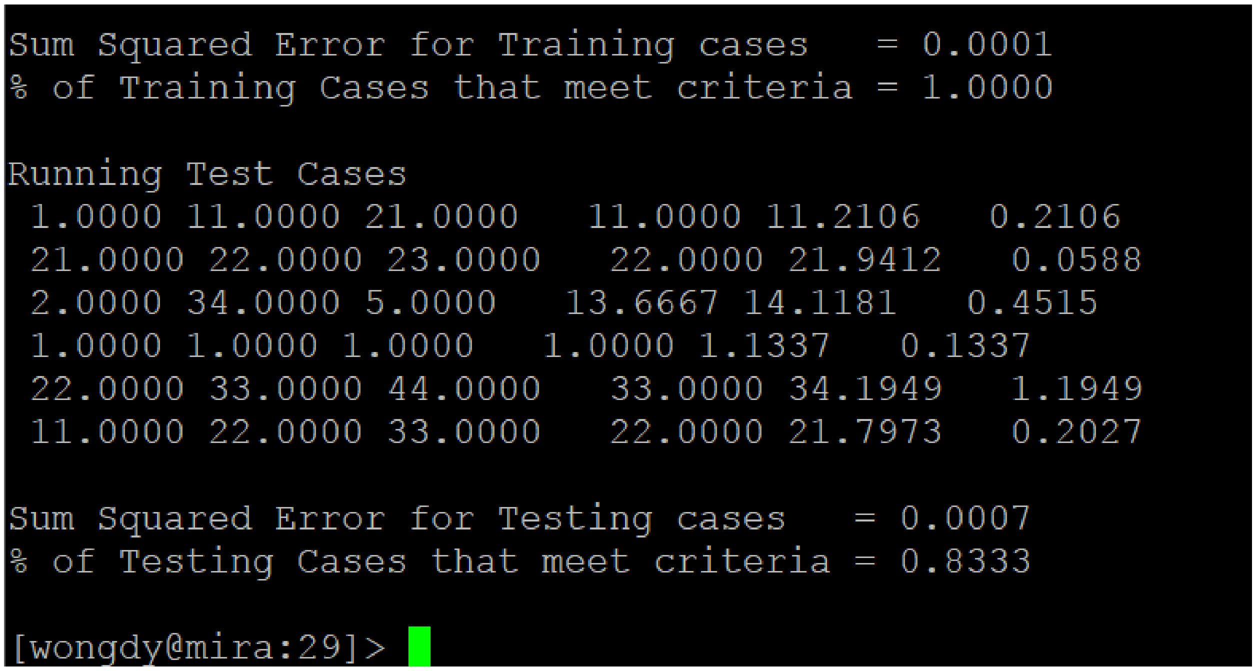
Training cases:

Sum Squared Error for Training cases   = 0.0000
% of Training Cases that meet criteria = 1.0000

Testing cases:

Sum Squared Error for Testing cases   = 0.0358
% of Testing Cases that meet criteria = 0.3333

Same setting and for 3 input 1 output(Average)



```
Sum Squared Error for Training cases   = 0.0001
% of Training Cases that meet criteria = 1.0000

Running Test Cases
 1.0000 11.0000 21.0000   11.0000 11.2106   0.2106
 21.0000 22.0000 23.0000   22.0000 21.9412   0.0588
 2.0000 34.0000 5.0000   13.6667 14.1181   0.4515
 1.0000 1.0000 1.0000   1.0000 1.1337   0.1337
 22.0000 33.0000 44.0000   33.0000 34.1949   1.1949
 11.0000 22.0000 33.0000   22.0000 21.7973   0.2027

Sum Squared Error for Testing cases   = 0.0007
% of Testing Cases that meet criteria = 0.8333

[wongdy@mira:29]>
```

Training cases:

Sum Squared Error for Training cases   = 0.0001
% of Training Cases that meet criteria = 1.0000

Testing cases:

Sum Squared Error for Testing cases   = 0.0007
% of Testing Cases that meet criteria = 0.8333

Both average and median did learn from the training sets at 0.1 precision,
It only takes 10 seconds for them to run the training sections, since we only set it to 1000000.
Both generalize as expected, except the average have higher % in testing cases than the median at 0.5 precision.

***Conclusion**

In conclusion, We think that it is acceptable when compared to the example that Dr. Gordon showed in class using 2 input and 1 output(sum) because in our settings, we have a tighter criteria and test criteria, the numbers should be lower than expected. Part 2 have better results than part 1 because part 2 only have 1 output, such that it's easier to training and generalize. Since 2 results from part 2 are different, we think that the code flavored in finding the average than the median since they both have the same number of training sets, input and output.

Part 3:

Image Classification Problem

There once lived two men who looked very alike. These men were very different but would constantly be mistaken for the other. For the longest time they never saw each other and did not know much of who the other person was. All of that changed in Fall 2017. These men finally met and everything made sense. Both men felt pity for everyone else who could not be as handsome as them and from this pity came an idea. Why don't they make the lives of the rest of the populace easier by making it easier for them to tell these two gods amongst men apart. So they came together, using the latest technology they knew of to make a program that could tell them apart. Finally there is an easy way to tell Douglas Wong and Tanveer Bariana apart.

This was no easy task that utilizes 72 pictures like the ones below to develop.

How we developed the training and testing data sets

We developed the data set by taking pictures of ourselves over a series of days to give it some variety of data to train on. Once we acquired all the raw data we cropped the pictures to just a bust shot to prevent any noise from our environment affecting the training. Then we resized the pictures down to 100 x 100 pixels to speed up the process of training and set a white fill to prevent the pictures from becoming distorted. Then once we had cleaner data we split it up 67% for training, 23% for testing and ran the trainer over 250 epochs.

Summary of the CNN architecture that worked best, and why we utilized it

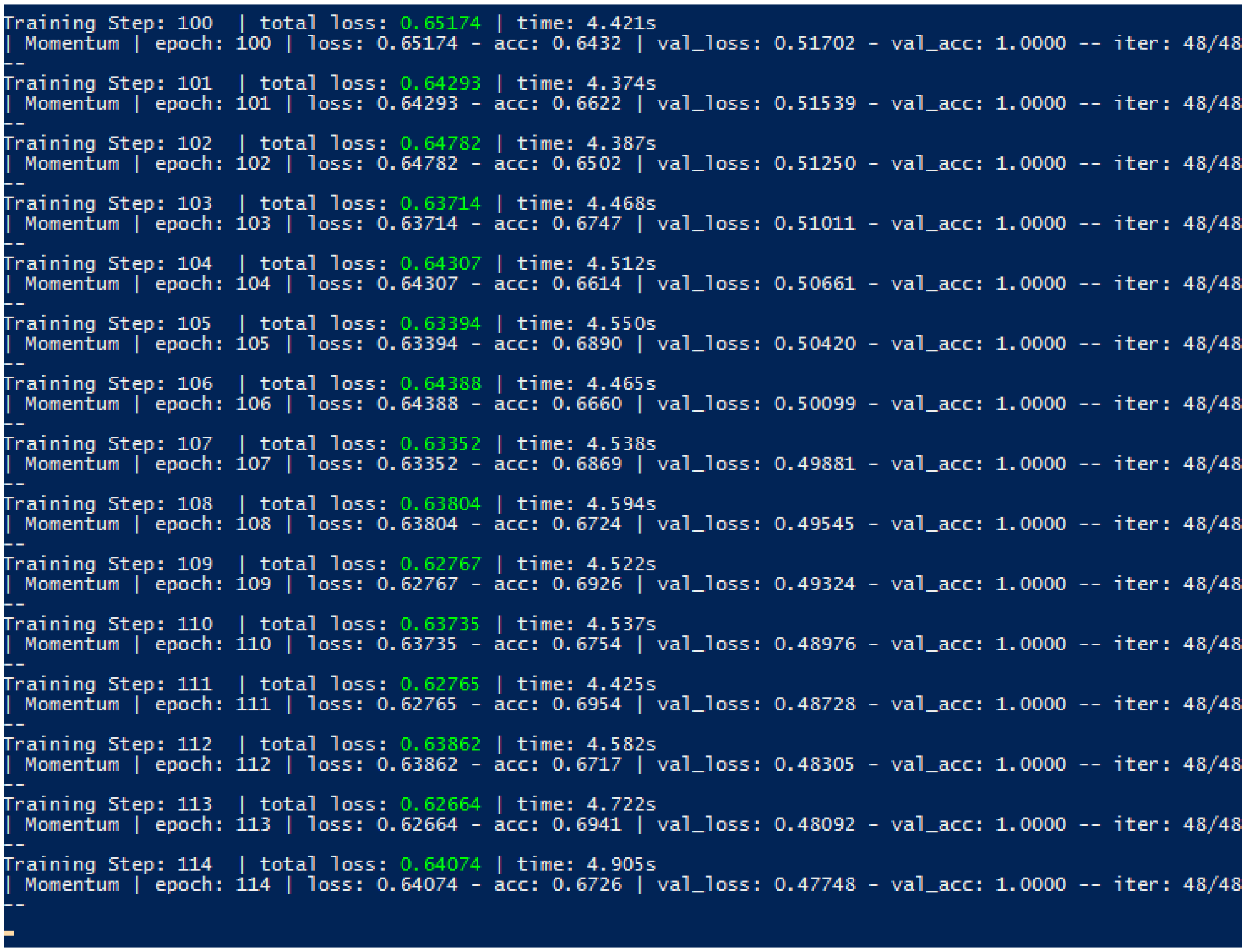
We implemented Alex-net style of CNN architectures and stayed with it because it worked. Tanveer experimented with the CNN architecture that came with the TFLearn examples but ultimately decided that Alex-net served the purposes of this project sufficiently. Experiments with the TFLearn example architecture were fueled entirely by a desire to understand the api enough to teach it.

Summary of our results

So it can accurately guess the picture all the way down to 32 epochs but significant learning is still at 250. For this training run took 16 minutes. It is a 75% confident and successfully guesses all of them so it generalized pretty well. The net learned the problem well as it has a 100% success rate with a 75% level of confidence for each guess. In the intrest of accuracy we tested it again running over 1000 epochs and noticed no significant improvement.

TensorBoard explanation of results.

We were unable to get TensorBoard working but watching the training program run one can notice the the loss inversely corresponds to the accuracy. This is expected given the equation for confidence was loss- acc. The console output shows the confidence moving forward only slightly faster than it is moving back, resulting in converging at a 75% level of confidence.



```
Training Step: 100  | total loss: 0.65174 | time: 4.421s
| Momentum | epoch: 100 | loss: 0.65174 - acc: 0.6432 | val_loss: 0.51702 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 101  | total loss: 0.64293 | time: 4.374s
| Momentum | epoch: 101 | loss: 0.64293 - acc: 0.6622 | val_loss: 0.51539 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 102  | total loss: 0.64782 | time: 4.387s
| Momentum | epoch: 102 | loss: 0.64782 - acc: 0.6502 | val_loss: 0.51250 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 103  | total loss: 0.63714 | time: 4.468s
| Momentum | epoch: 103 | loss: 0.63714 - acc: 0.6747 | val_loss: 0.51011 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 104  | total loss: 0.64307 | time: 4.512s
| Momentum | epoch: 104 | loss: 0.64307 - acc: 0.6614 | val_loss: 0.50661 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 105  | total loss: 0.63394 | time: 4.550s
| Momentum | epoch: 105 | loss: 0.63394 - acc: 0.6890 | val_loss: 0.50420 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 106  | total loss: 0.64388 | time: 4.465s
| Momentum | epoch: 106 | loss: 0.64388 - acc: 0.6660 | val_loss: 0.50099 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 107  | total loss: 0.63352 | time: 4.538s
| Momentum | epoch: 107 | loss: 0.63352 - acc: 0.6869 | val_loss: 0.49881 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 108  | total loss: 0.63804 | time: 4.594s
| Momentum | epoch: 108 | loss: 0.63804 - acc: 0.6724 | val_loss: 0.49545 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 109  | total loss: 0.62767 | time: 4.522s
| Momentum | epoch: 109 | loss: 0.62767 - acc: 0.6926 | val_loss: 0.49324 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 110  | total loss: 0.63735 | time: 4.537s
| Momentum | epoch: 110 | loss: 0.63735 - acc: 0.6754 | val_loss: 0.48976 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 111  | total loss: 0.62765 | time: 4.425s
| Momentum | epoch: 111 | loss: 0.62765 - acc: 0.6954 | val_loss: 0.48728 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 112  | total loss: 0.63862 | time: 4.582s
| Momentum | epoch: 112 | loss: 0.63862 - acc: 0.6717 | val_loss: 0.48305 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 113  | total loss: 0.62664 | time: 4.722s
| Momentum | epoch: 113 | loss: 0.62664 - acc: 0.6941 | val_loss: 0.48092 - val_acc: 1.0000 -- iter: 48/48
--
Training Step: 114  | total loss: 0.64074 | time: 4.905s
| Momentum | epoch: 114 | loss: 0.64074 - acc: 0.6726 | val_loss: 0.47748 - val_acc: 1.0000 -- iter: 48/48
--
```

The TFLearn script code that corresponds to our best solution.

So here is the best training program we had---------------------------------

```
import neuralNet as net
import numpy as np
from tflearn.data_utils import image_preloader
model = net.model
X , Y = image_preloader(target_path='./train',
      image_shape=(100, 100), mode='folder', grayscale=False,
      categorical_labels=True, normalize=True)
X = np.reshape(X, (-1, 100, 100, 3))
W, Z = image_preloader(target_path='./test',
      image_shape=(100, 100), mode='folder', grayscale=False,
      categorical_labels=True, normalize=True)
W = np.reshape(W, (-1, 100, 100, 3))
model.fit(X, Y, n_epoch=250, validation_set=(W,Z),
show_metric=True)
```

```
model.save('./ZtrainedNet/final-model.tfl')
```

Here is our best net--

```
import tflearn
from tflearn.layers.core import input_data, dropout,
fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy
acc = Accuracy()
network = input_data(shape=[None, 100, 100, 3])
# Conv layers ------------------------------------
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
# Fully Connected Layers --------------------------
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 2, activation='softmax')
network = regression(network, optimizer='momentum',
     loss='categorical_crossentropy',
     learning_rate=0.001, metric=acc)
model = tflearn.DNN(network,tensorboard_verbose=3,
tensorboard_dir="logs")
```