Marko Smiljanic

CSc – 180

Spring 2019

Project 3 – Neural Networks and Deep Learning

I. Part 1 – Single Neural Network

Settings:

```
// NN parameters  ------------------------------------------------
#define NumINs      3     // number of inputs, not including bias node
#define NumOUTs     2      // number of outputs, not including bias node
#define Criteria    0.3334  // all training outputs must be within this for training to stop
#define TestCriteria 0.6667    // all testing outputs must be within this for generalization

#define LearningRate 0.3    // most books suggest 0.3 as a starting point
#define Momentum    0.95   // must be >=0 and <1
#define bias       -1     // output value of bias node (usually 1, sometimes -1 for sigmoid)
#define weightInit  1.00   // weights are initialized randomly with this max magnitude
#define MaxIterate  1000000 // maximum number of iterations before giving up training
#define ReportIntv  101   // 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   10     // 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 layer
#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   1
#define Activation5   1
#define Activation6   1
#define NumOfCols   6      // number of non-zero layers specified above, including the
input layer
#define NumOfRows   10     // largest layer - i.e., max number of rows in the network
```

```
// data files ---------------------------------------------
#define TrainFile    "BeamA.dat"  // file containing training data
#define TestFile     "BeamB.dat"  // file containing testing data
#define TrainCases   20          // number of training cases
#define TestCases    10          // number of test cases

// advanced settings ----------------------------------------
#define LeakyReluAmt 0.1
```

Summary:

The program learned the training set to 100% accuracy. The program reported the percentage of training cases that met criteria to be 1.0000. Training took very little time. It was less than five seconds to run through the whole training set and it require 540,000 iterations to complete. It generalized well. My settings allowed for a wide margin of error and the program was usually within 3% error when it came to the final test cases. In training the error was much lower, usually around .5 or below. I think the program did a good job of solving for the mean of the data set, but it got a lot more flummoxed when it came to the median. That was where I saw the most error in the program.

II. Part 2 – Multiple Neural Network

I did not have time to complete this portion of the assignment.

III. Part 3 – Image Classification

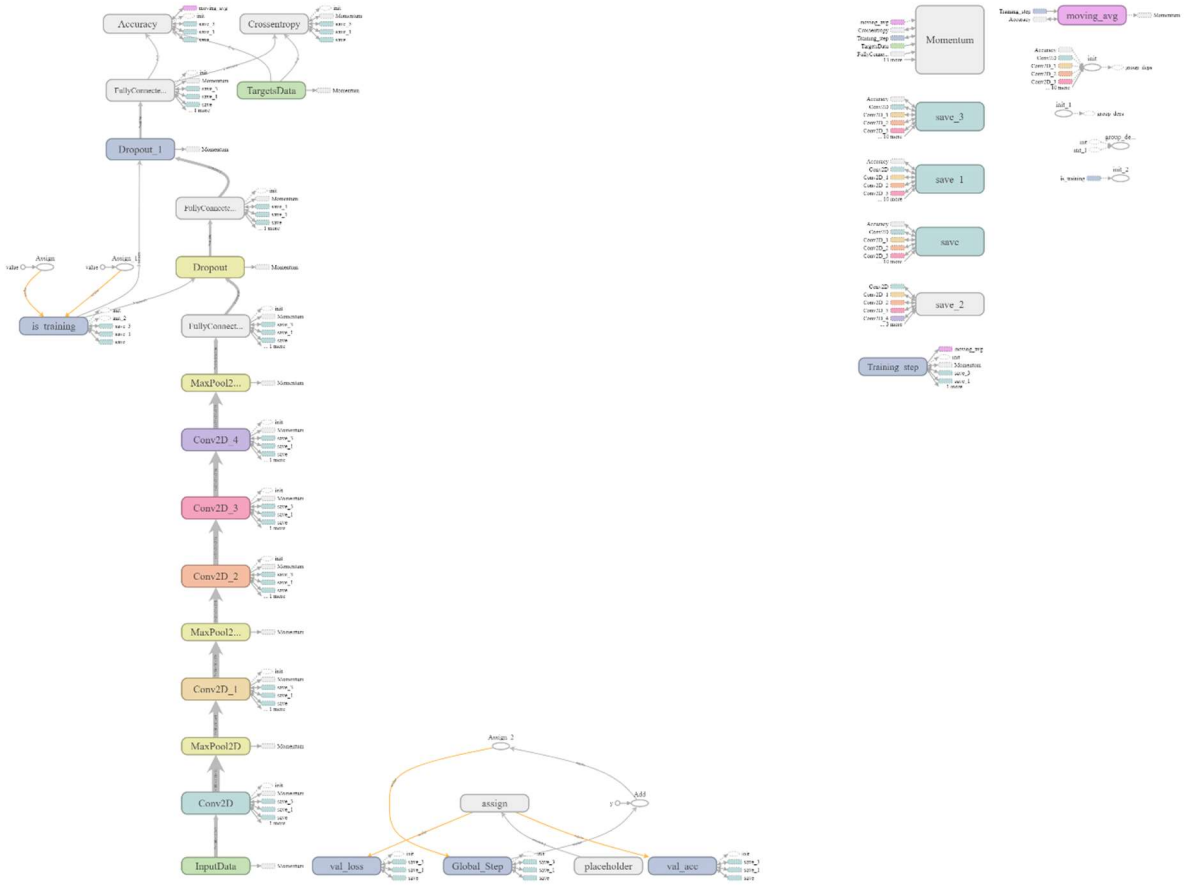
Image Classification:

My Image Classification Problem was to determine the difference between pictures of cats and pictures of hats. I wanted to see how the program would do with images that could be similar in color patterns but varied drastically in terms of shape on flow of lines on the images themselves.

I developed the training and testing sets by first searching for images of different types of cats on google. First, I searched for general cats, then white cats, and finally black cats. I used a similar pattern to find the images for hats, first searching for top hats, then baseball hats, and then hats in general.

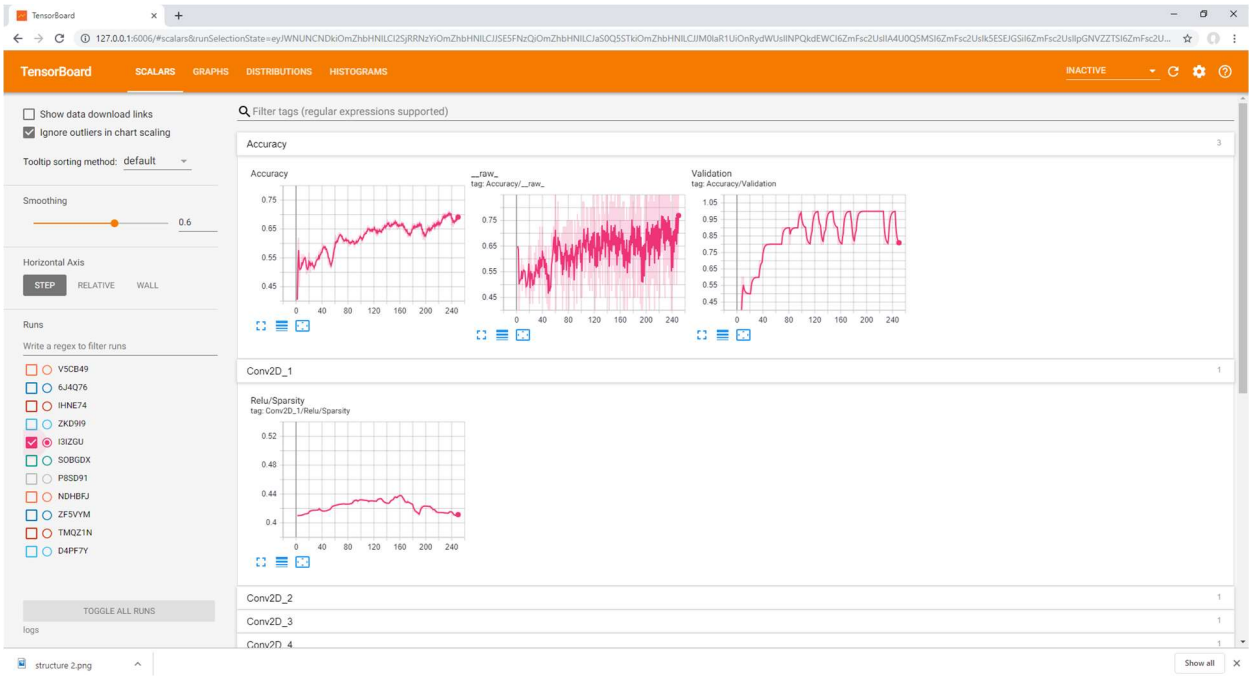
Architecture:

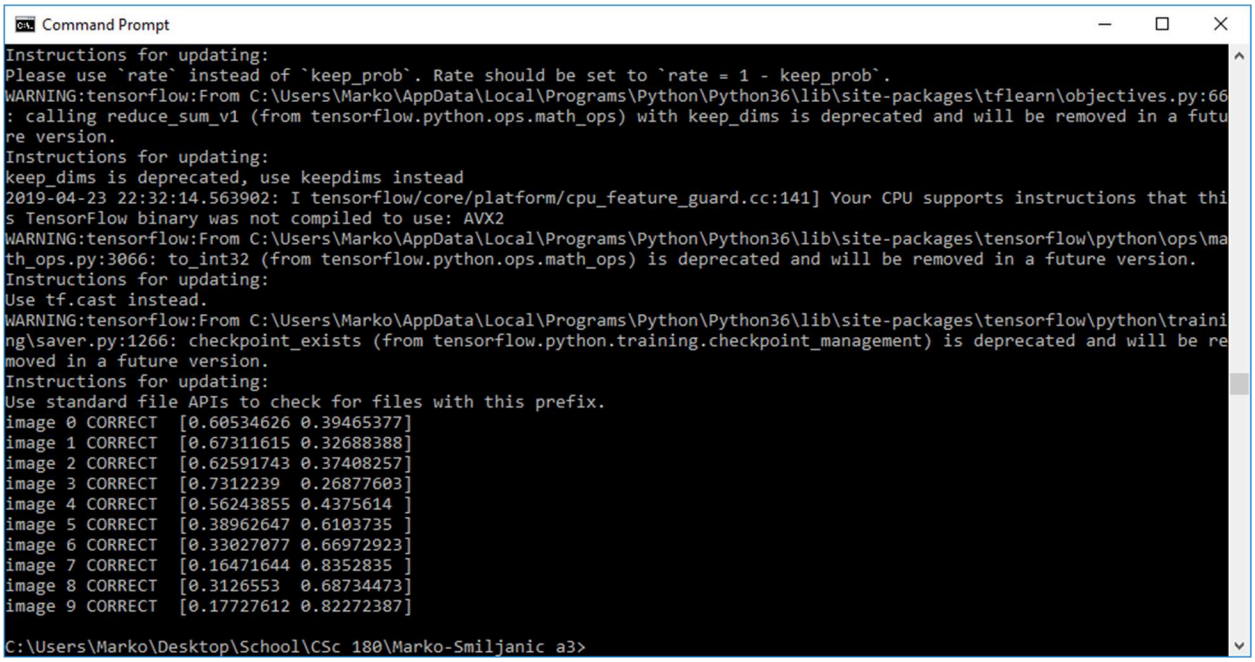
The architecture that ended up working best for me was a modification of the code that was provided by the professor. The dimensionality was conv(64x3), max pool(2), conv(128x3), max pool(2), conv(256x3), conv(256x3), conv(256x3), max pool(2) and then fully connected(4096, 4096, 2). This architecture is shown in the image below.



Summary:

The training took a very long time. I was ultimately unable to figure out how to turn on Cuda so I was limited in how long I could allow the program to run. I ran this architecture for 100 epochs, and each one took 12 seconds to complete. To create a comparison with the code provided by the professor, that architecture I ran for 250 epochs and it took roughly 4 seconds for each. This new architecture resulted in slightly more accurate, and more consistent guesses. I think the code generalized somewhat well. As with all architecture, it would have been more accurate had I run the training for longer. The accuracy ranged from 60 to 85 percent on when it came time to run the prediction.





It determined all images accurately, but it struggled more with the pictures of cats. I am not certain of this, but I feel this was caused by all the white space created from the resizing of the images, combined with some cat images having distinct backgrounds, such as someone's floor of their house.

Code:

```
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, 128, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 256, 3, strides=1, activation='relu')
network = conv_2d(network, 256, 3, strides=1, activation='relu')
network = conv_2d(network, 256, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)

# Fully Connected Layers -------------------------
network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 4096, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 2, activation='softmax')

# Final network
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
```

```
                learning_rate=0.001, metric=acc)
model = tflearn.DNN(network, tensorboard_verbose=3, tensorboard_dir="logs")
#model = tflearn.DNN(network)
```