SURF **SURF**

Second Progress Report

Name: Nishanth Bhaskara **July 7, 2016**

A) Current Goals

Currently, we are trying to further study CNNs, or convolutional neural networks, and test the full extent of their use in classification problems. However, instead of going down the standard road of image classification, we are trying to do something different. Our matrices have an extra dimension, time, as we are now working on classification of videos. The TensorFlow library is still being used extensively, but as CNNs for 4-D classification have not been used much in the past, several key functions have to be written by ourselves. Another problem we encountered with this is that for videos to be classified to specific categories/labels, they all have to be, to a certain degree, uniform. With that in mind, we chose to use fMRI clips. We wish to be able to diagnose a patient as autistic or not autistic based solely off of the software's analysis on the fMRI clips.

B) Current Work

Each fMRI clip is a $64x64x40x300$ clip. The first three dimensions are spatial, and the last is temporal. An immediate challenge that is presented here is that in a classic image classification problem, although the images **are** three dimensional, the third dimension was color channels and not depth. This would mean that we would have to reshape all of our clips into matrices of dimension $64x64x1x40x300$ instead for better results. There are several problems with this however. First of all, reshaping a matrix with near $50,000,000$ entries is not a task that a computer can do in a fast amount of time. Even with clever manipulations of list slicing and redeclarations of portions of the matrix in Python, this is still a lengthy process. Second, TensorFlow does not support convolution and pooling operations over input data that is 5 - D. This problem is not as troublesome though, as we can write custom functions as the work progresses. Another problem we encountered is the order of the dimensions of the data. In NIFTI format, the first three dimensions are spatial and the last is temporal. However, for a convolutional network, filters extend through the depth of the input (the last dimension) and are through a small spatial size. This means that we need our input to actually be in the form $300x64x64x40$. Here again we encounter the problem of reshaping a matrix with $50,000,000$ entries, which is no easy problem.

C) Code/Network Layout

The entire script for the classification is over 250 lines long, so it is hard to go through all the details in a succinct matter, but we can focus on the main convolution and pooling methods, as well as the overall layer of the network.

```
def conv3d(x, W):
  return tf.nn.conv3d(x, W, strides=[1, 1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool3d(x, ksize=[1, 2, 2, 1, 1],
                          strides=[1, 2, 2, 1, 1], padding='SAME')
```

Our convolution function is having a stride of 1 among every single dimension (the first dimension being the index of the input clip). Our pooling is doing $2x2$ pooling over the x and y dimensions of the clip and retaining information in all other dimensions.

```
W_conv1 = weight_variable([5, 5, 40, 300, 64])
b_conv1 = bias_variable([64])
x_image = tf.reshape(x, [-1,64,64,40,300])
h_conv1 = tf.nn.relu(conv3d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)


W_conv2 = weight_variable([5, 5, 1, 64, 128])
b_conv2 = bias_variable([128])

h_conv2 = tf.nn.relu(conv3d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([13 * 13 * 128, 4096])
b_fc1 = bias_variable([4096])

h_pool2_flat = tf.reshape(h_pool2, [-1, 13 * 13 * 128])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)



keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([4096, 2])
b_fc2 = bias_variable([2])
```

Our original filters have dimension $5x5x40x300$ and there are 64 output channels. We then have a pooling layer. The resultant volume is now $30x30x1x64$. The next filter is of size $5x5x1x64$ with 128 output layers, and after another pooling layer, our resultant volume is $13x13x1128$. Lastly we have a fully connected layer with 4096 neurons, which leads to a binary output (autism or no autism).

Technical Difficulties So far, I have no results from this. The theory is sound and I have checked over the code repeatedly, but the simple problem is that I need more computing power to achieve this feat. At first, my laptop was the only thing I had at my arsenal, but after running for 10 hours and not even making 20 percent progress. I was given an ASUS ROG G20, an extremely powerful gaming computer, to do further calculations, but to use it efficiently, I have to make use of the NVIDIA GPU as it alone is more powerful than all of the CPU cores. The problem is that there are very few supported OS's by the CUDA Toolkit, of which Windows is not a member. (The CUDA Toolkit is needed in conjunction with TensorFlow to run calculations on the GPU). I dual booted an Ubuntu system, and at first struggled to see why the OS kept crashing. It ended up being because I did not have the appropriate dirvers for the graphics chip, and was not installing during primary installation because I did not have the correct SSL certificate to join the Caltech network. Then, once I used an ethernet connection and downloaded the graphics drivers, the OS kept crashing, as it ended up being that Ubuntu in itself is not compatible with this specific NVIDIA card. Now, I am dual booting a Fedora OS instead (a different Linux distro) in hopes that this will be compatible with the graphics card. As soon as I can use the full functionality of the chip, I will be able to get results from the classification problem.

D) Conclusion

I am on track with the project as long as I am able to solve the technical difficulties with the computer, the fMRI classification should go just fine.