COMS 4995 Deep Learning Assignment 2

Pulkit Jain (pj2313)
Greg Kocher (gk2500)
Ratheet Pandya (rp2707)

Part 1

- Architecture (batch_size omitted):
    - 32 x 32 x 3 input
    - 32 x 32 x 3 x 64 first convolutional layer: 64 5 x 5 x 3 filters fed into a ReLU
    - 8 x 8 x 64 normalized max pooling layer (2 x 2 filter, stride 2)
    - 32 x 32 x 3 x 64 second convolutional layer: 64 5 x 5 x 3 filters fed into a ReLU
    - 8 x 8 x 64 normalized max pooling layer (2 x 2 filter, stride 2)
    - 512 unit dense fully-connected layer (with dropout regularization using a 0.4 keep probability)
    - 10 unit dense output layer (one for each output class)
- Training Accuracy: 0.88
- Test Accuracy: 0.70

Part 2

- Architecture and observations: (see below)
- Training Accuracy: 0.84
- Test Accuracy: 0.7166
- Duration of training: 5000 iterations (80 minutes)
- Size of training data: 45000

Our architecture for Part 2 contains convolution layers, max pooling layers, normalization layers, and dense layers as before.

- It also includes dropout layers (.e.g with dropout probability .6)
- Changing the optimizer from SGD to ADAM was extremely helpful. WIthin a few hundred steps, the training accuracy already jumps up to 40-50%, compared to taking several times longer with regular SGD.
- We used a standard best practice for weight initialization [Xavier-Glorot Gaussian]
- We did standard scaling on the input images [0 mean, unit variance]

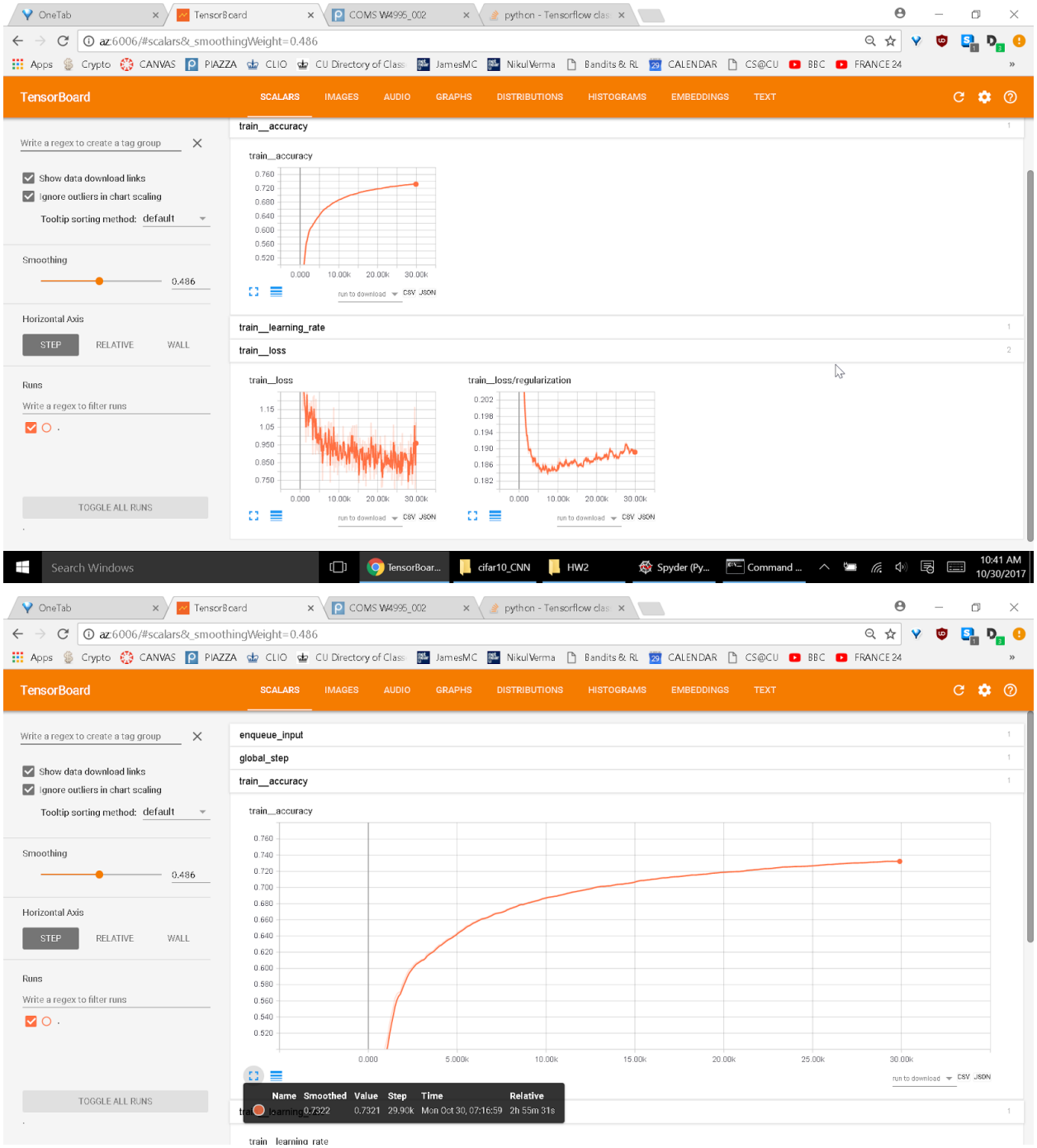
- In one of our other models, we also used L2 norm regularization on the weights [weights of the conv layers and dense layers]
- Replacing ReLU with ELU (exponential linear units) by using “activation=tf.nn.elu” in the conv and dense layers did not seem to have a noticeable impact
- Using alternative forms of normalization: our current implementation uses Local Response Normalization, but we also tested Batchnorm, which had basically the same performance
- Changing the architecture to include atrous convolution layers [dilated convolutions] was explored but it’s hard to say if it had much effect in either direction. Theoretically, dilated convolutions could be useful here and can be set e.g. as follows: tf.layers.conv2d(...,dilation_rate=(2,2)), but for this particular small CPU architecture it didn’t have much impact.

In particular, the current implementation of the architecture is a modular set of conv-pool-norm blocks, with dropout and dense layers:

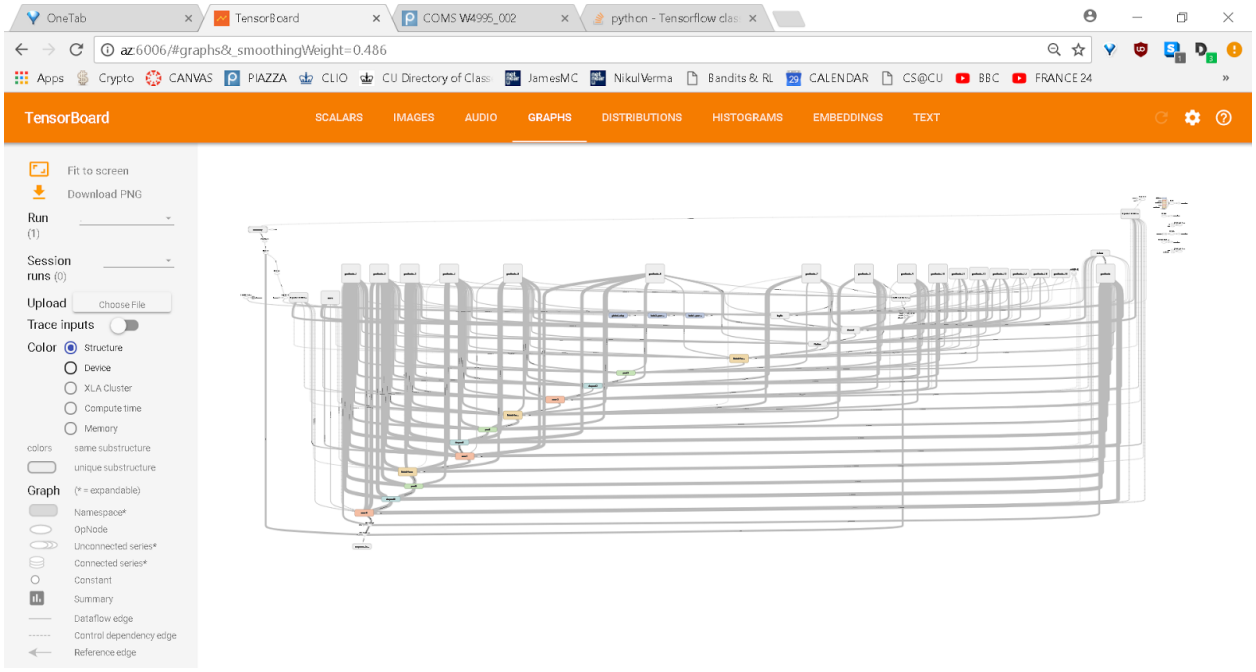
- 32 x 32 x 3 input
-
- Conv1: 64 filters, all (5 x 5 x 3), ReLU activation
- Max pooling (3 x 3 kernel, stride 2)
- Local Response Normalization
-
- Conv2: 64 filters, all (5 x 5 x 3), ReLU activation
- Local Response Normalization
- Max pooling (3 x 3 kernel, stride 2)
-
- FullyConnected1: 512 units
- Dropout: 0.4 keep probability
- FullyConnected2: Logits: 10 unit output layer (one for each output class)

Some of our Tensorboard plots and weight/gradient histograms are shown below for one of our architectures:

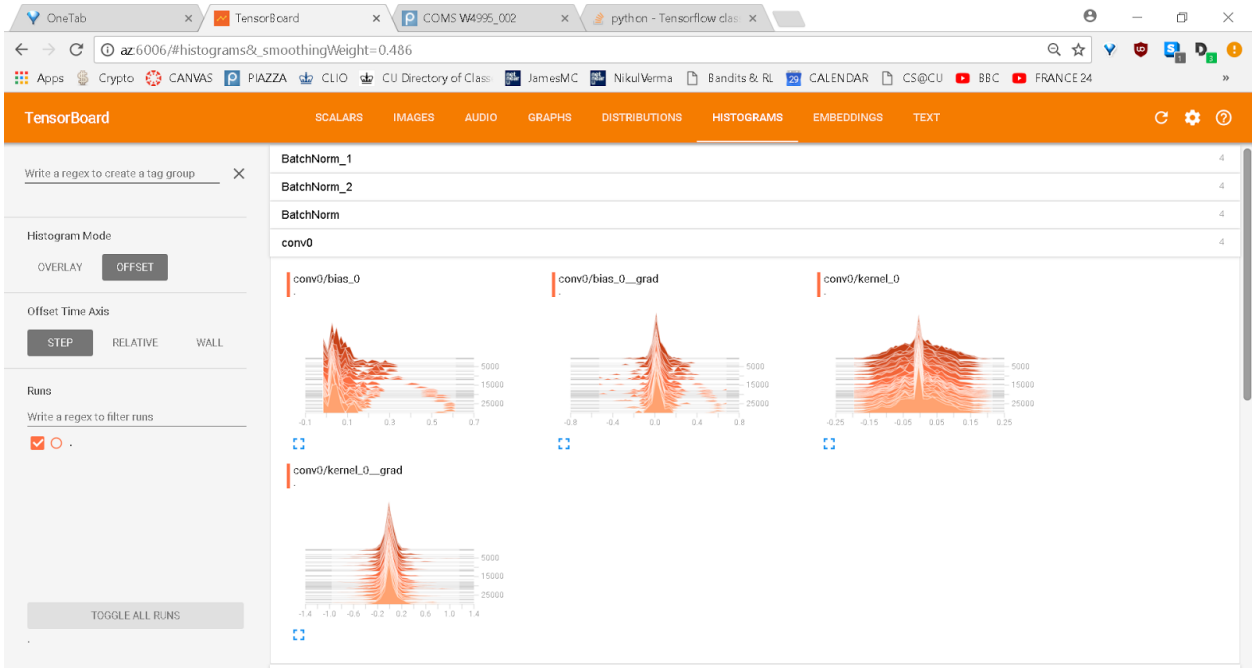
Accuracy and loss [loss is split into total loss and L2 regularization loss]:



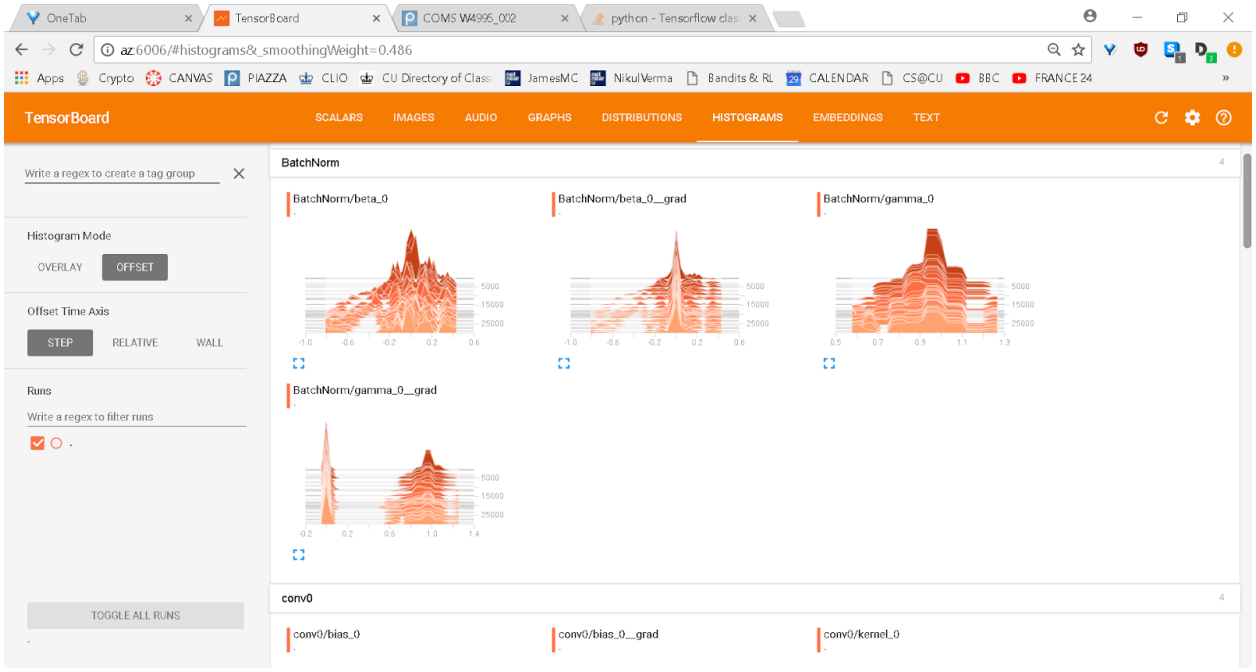
Graph structure: [architecture also put in list form later]



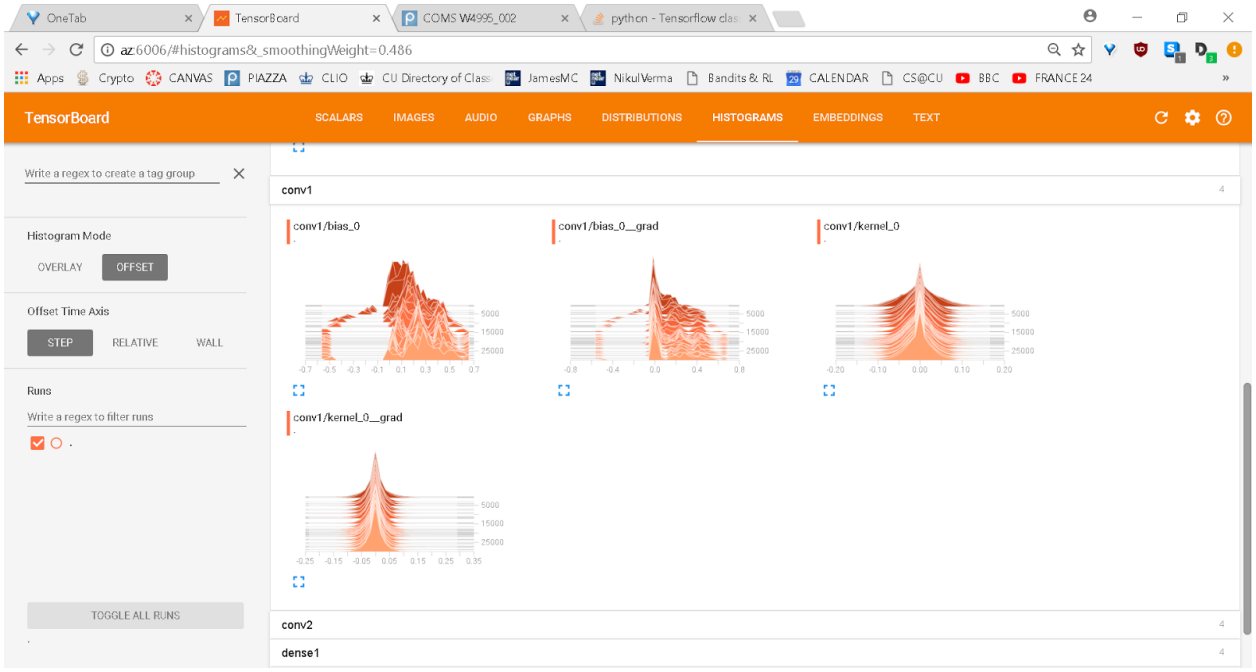
1st Conv layer [conv0]: weights and gradients:

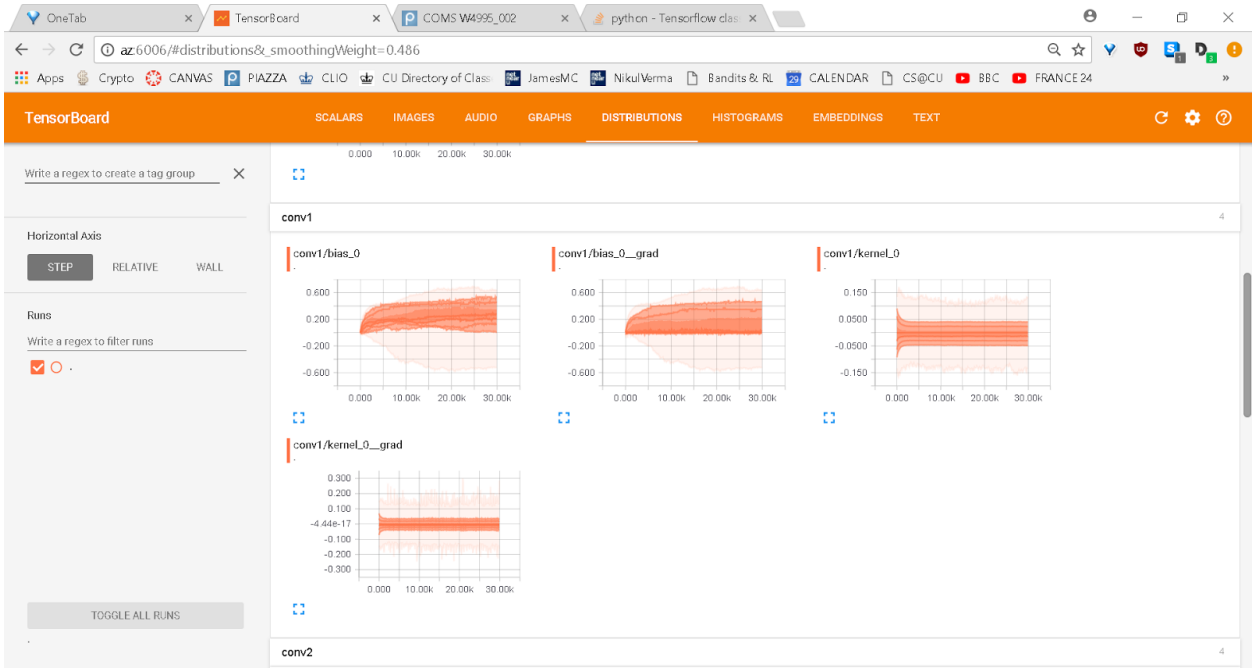


1st norm layer [as example of norm layers, other norm layers not shown]:

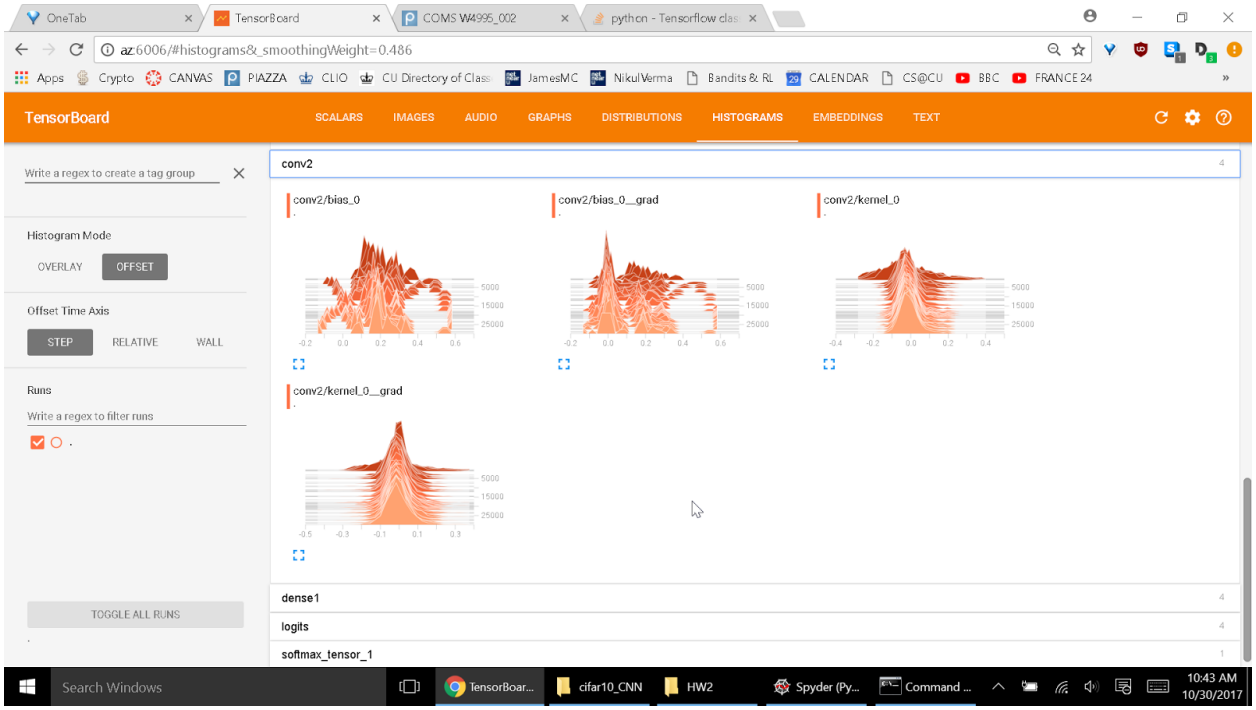


2nd conv layer [conv1]

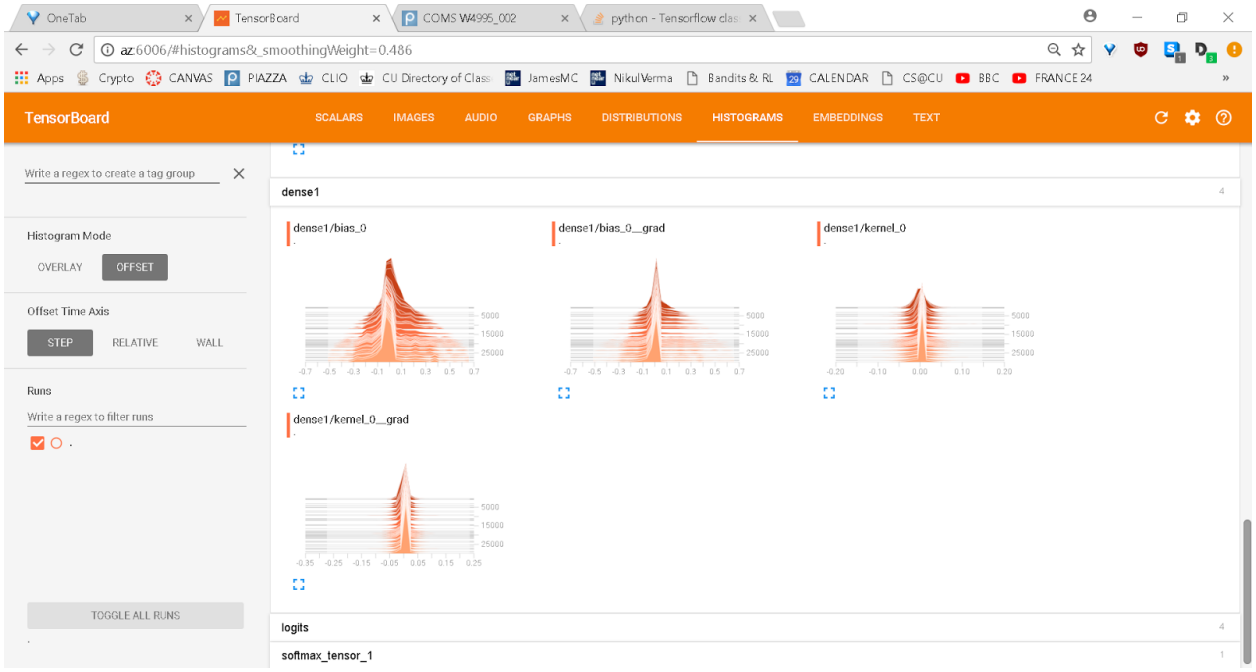




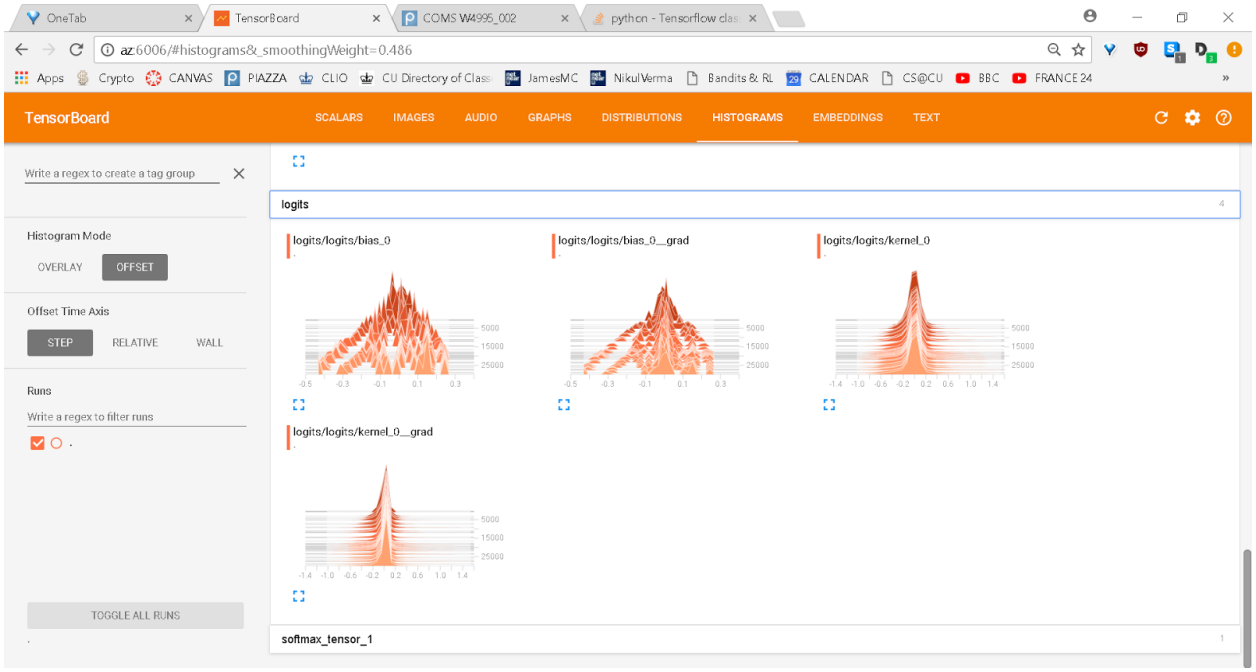
3rd conv layer [conv2]



Dense layer 1 weights and gradients:



Logits layer [final dense layer] weights and gradients:



The above figures are from a model that had this architecture:

Conv0 - 64filters with [3,3] kernel size
ReLU activations
Dropout - drop probability .25
MaxPooling - kernel size [2,2] with stride 2
BatchNorm

Conv1 - 32 filters with [3,3] kernel size
ReLU activations
Dropout - drop probability .25
MaxPooling - kernel size [2,2] with stride 2
BatchNorm

Conv2 - 16 filters with [1,1] kernel size
ReLU activations
Dropout - drop probability .25
MaxPooling - kernel size [2,2] with stride 2
BatchNorm

Pool to flat (just reshaping)
Dense (Fully connected) - 512 nodes
Dense - logits out layer - 10 nodes (1 per class)
Softmax operation

**Also note: this is not the exact model that made the predictions represented in ans1 or ans2, this was another model we trained earlier but still had good representative Tensorboard screenshots. You can see more of those model outputs in the ipynb notebooks where info is printed to the console.

References

- Tensorflow MNIST Deep tutorial: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/mnist_deep.py