Tensorflow hands-on

•••

CVM - M5

Jonatan Poveda
Master in Computer Vision
UAB, CVC, M5, 2018

Tensorflow hands-on

This exercise has the focus to accomplish the following tasks:

- Build the Simple Example in Tensorflow with 3 fully-connected layers.
- Train MNIST on the simple network and evaluate the test set for classification. This is not a competition on best accuracy.
- Build the LeNet model and train MNIST on it. Give classification performance on the test set.
- Visualize the loss and accuracy with TensorBoard. Plot the graph of the network.
- (Extra) Define LeNet as a class in a separate file
- (Extra) Visualize any other metric or features.

Dataset

The dataset used is MNIST, is a well known and maybe the most used in machine learning, nowadays as a toy example. It is a handwritten digit database.

It is split in:

- Training Set: 55000 samples (79%)
- Validation Set: 5000 samples (7%)
- Test Set: 10000 samples (14%)

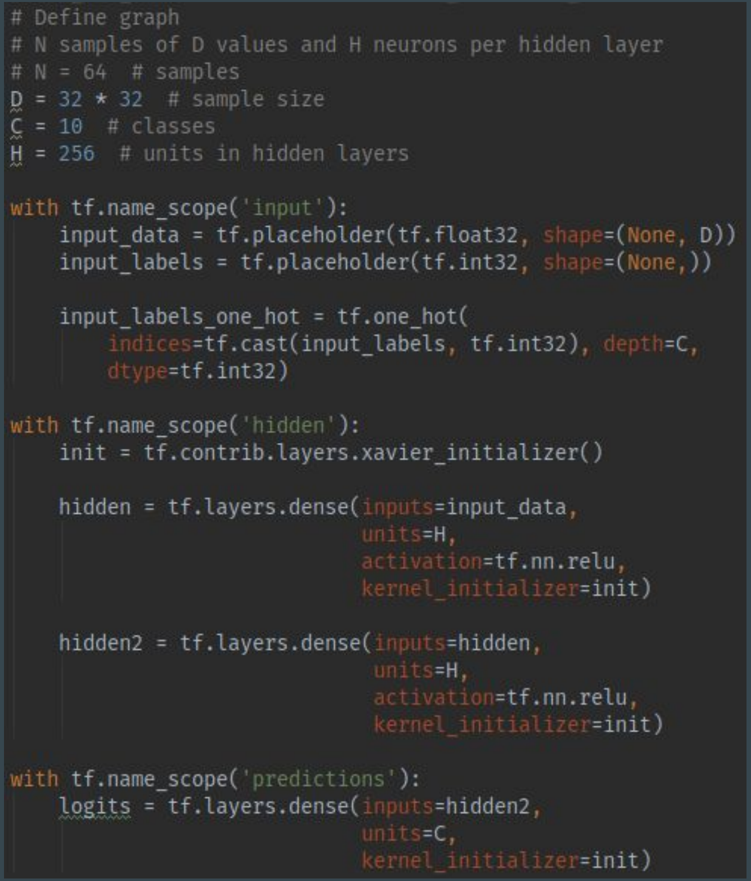
But only the training and test set are used

Simple Example with 3 FC-layers

In TF this simple model could be described as:

- 2 placeholders
  - input data: 1D images (vectorization of a 2D image)
  - input_labels: categorical labels, in our case ∈[‘1’, .. , ’9’]
- 3 FC layers (called Dense in TF)
  - 2 hidden layers (with an initializer)
  - a prediction layer (with an initializer)

In our case the images are of size 32x32 so a vector of 1024 values [D], there are 10 different classes [C] and each hidden layer has 256 neurons [N]

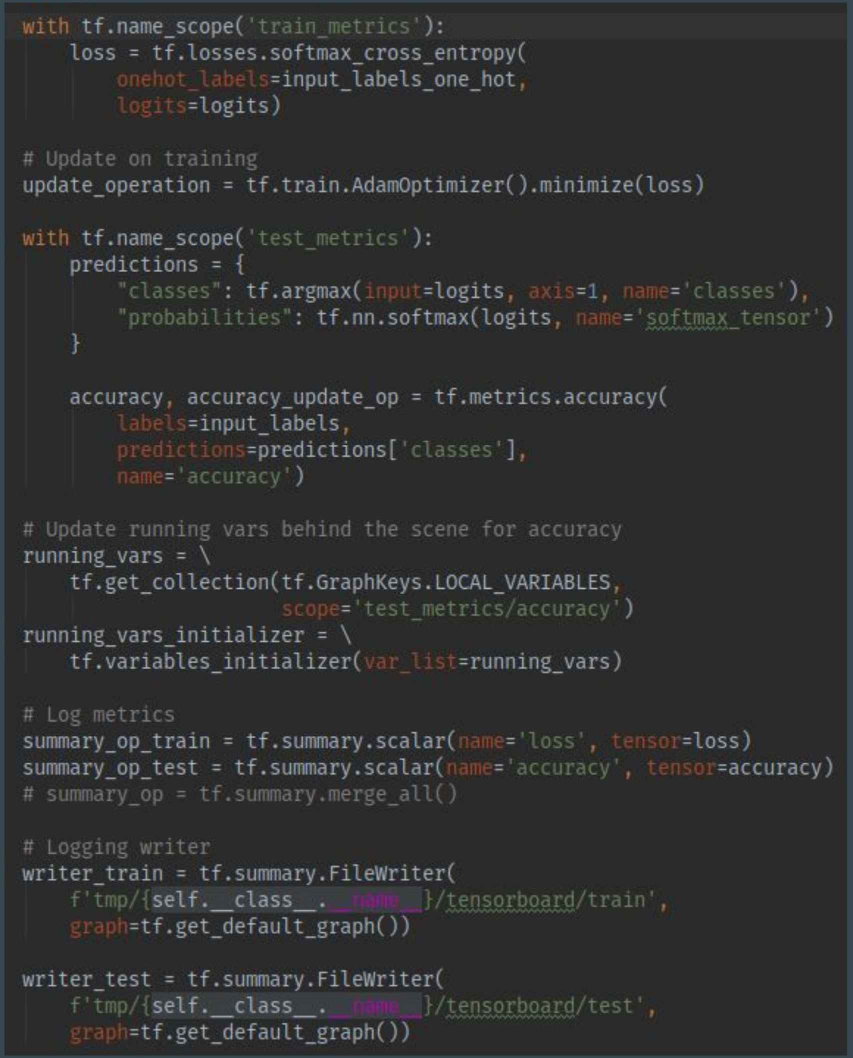


```
# Define graph
# N samples of D values and H neurons per hidden layer
# N = 64  # samples
D = 32 * 32  # sample size
C = 10  # classes
H = 256  # units in hidden layers

with tf.name_scope('input'):
    input_data = tf.placeholder(tf.float32, shape=(None, D))
    input_labels = tf.placeholder(tf.int32, shape=(None,))

    input_labels_one_hot = tf.one_hot(
        indices=tf.cast(input_labels, tf.int32), depth=C,
        dtype=tf.int32)

with tf.name_scope('hidden'):
    init = tf.contrib.layers.xavier_initializer()

    hidden = tf.layers.dense(inputs=input_data,
                             units=H,
                             activation=tf.nn.relu,
                             kernel_initializer=init)

    hidden2 = tf.layers.dense(inputs=hidden,
                              units=H,
                              activation=tf.nn.relu,
                              kernel_initializer=init)

with tf.name_scope('predictions'):
    logits = tf.layers.dense(inputs=hidden2,
                             units=C,
                             kernel_initializer=init)
```

Simple Example

We set:

- For training
  - a loss function: softmax_cross_entropy
  - an optimizer: AdamOptimizer
    - with the parameter to be minimized: loss
- For testing
  - a metric to assess our classifier: accuracy
  - an initializer for this metric
- For logging
  - summary operations
  - summary writers

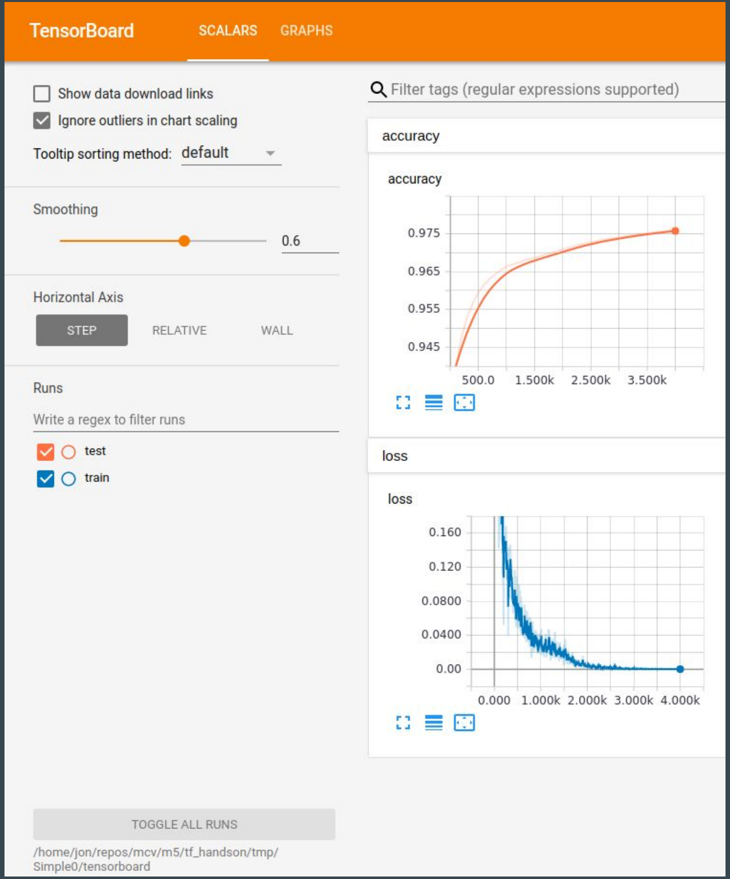


```
with tf.name_scope('train_metrics'):
    loss = tf.losses.softmax_cross_entropy(
        onehot_labels=input_labels_one_hot,
        logits=logits)

# Update on training
update_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.name_scope('test_metrics'):
    predictions = {
        "classes": tf.argmax(input=logits, axis=1, name='classes'),
        "probabilities": tf.nn.softmax(logits, name='softmax_tensor')
    }

    accuracy, accuracy_update_op = tf.metrics.accuracy(
        labels=input_labels,
        predictions=predictions['classes'],
        name='accuracy')

# Update running vars behind the scene for accuracy
running_vars = \
    tf.get_collection(tf.GraphKeys.LOCAL_VARIABLES,
                      scope='test_metrics/accuracy')
running_vars_initializer = \
    tf.variables_initializer(var_list=running_vars)

# Log metrics
summary_op_train = tf.summary.scalar(name='loss', tensor=loss)
summary_op_test = tf.summary.scalar(name='accuracy', tensor=accuracy)
# summary_op = tf.summary.merge_all()

# Logging writer
writer_train = tf.summary.FileWriter(
    f'tmp/{self.__class__.__name__}/tensorboard/train',
    graph=tf.get_default_graph())

writer_test = tf.summary.FileWriter(
    f'tmp/{self.__class__.__name__}/tensorboard/test',
    graph=tf.get_default_graph())
```

Simple Example Performance

We can track the performance of the training with TensorBoard (TB).

In our case, we separate the summaries in two, ‘train’ and ‘test’ so they are better found in TB.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

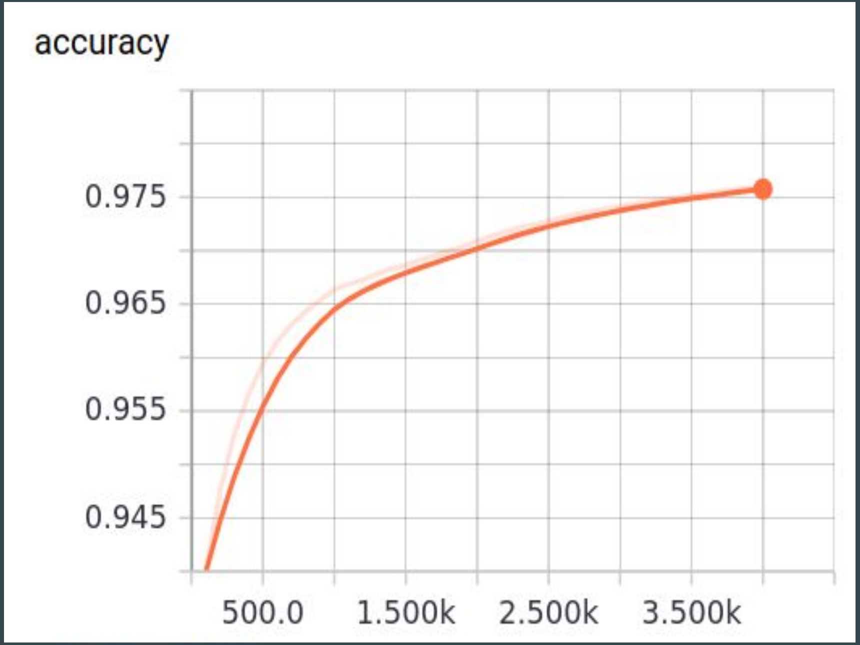
Smoothing 0.6

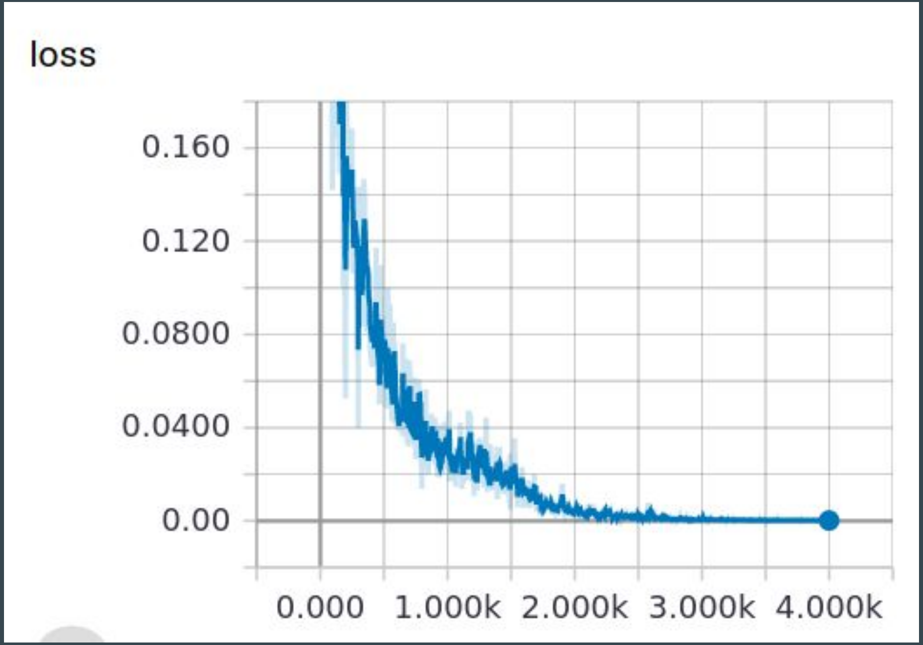
Horizontal Axis [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
☑ test
☑ train

/home/jon/repos/mcv/m5/tf_handson/tmp/Simple0/tensorboard

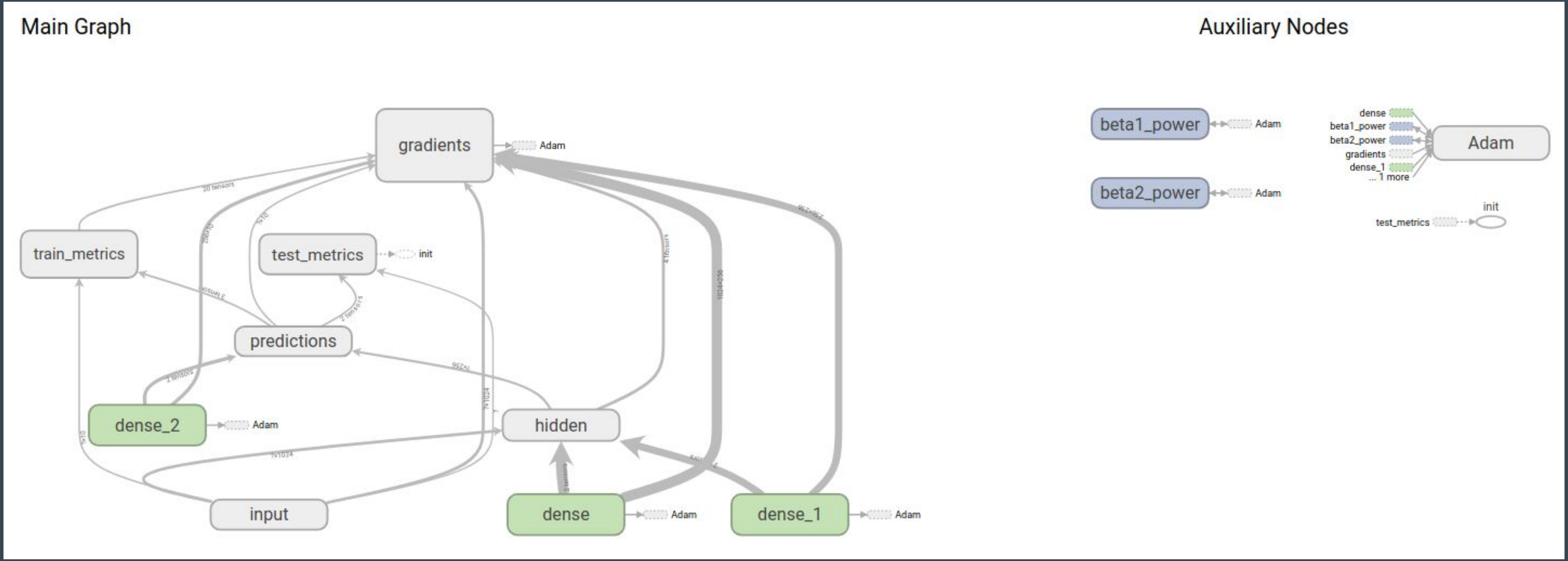
Simple Example Performance





Simple Example

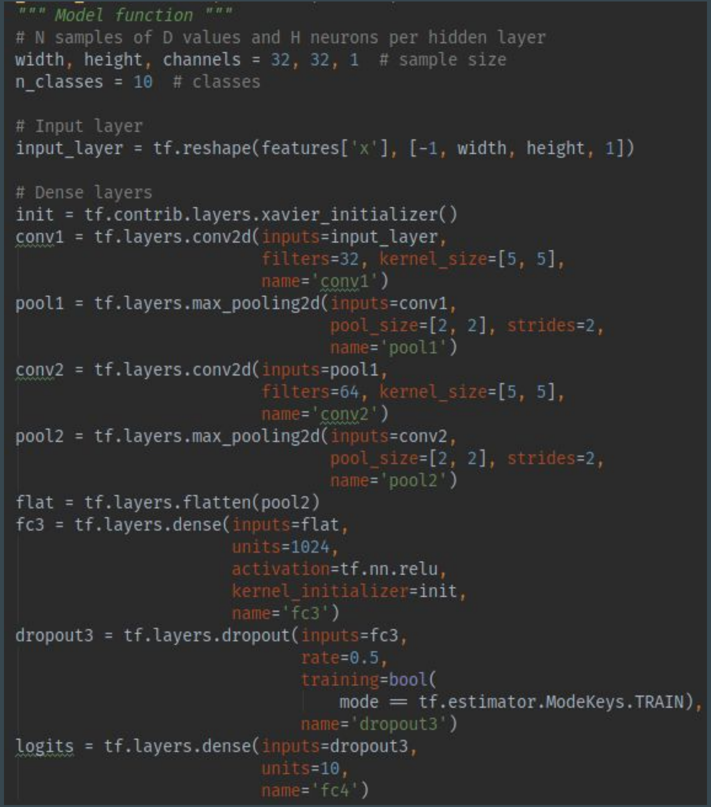
Model graph in Tensorboard



LeNet

In TF this model could be described as:

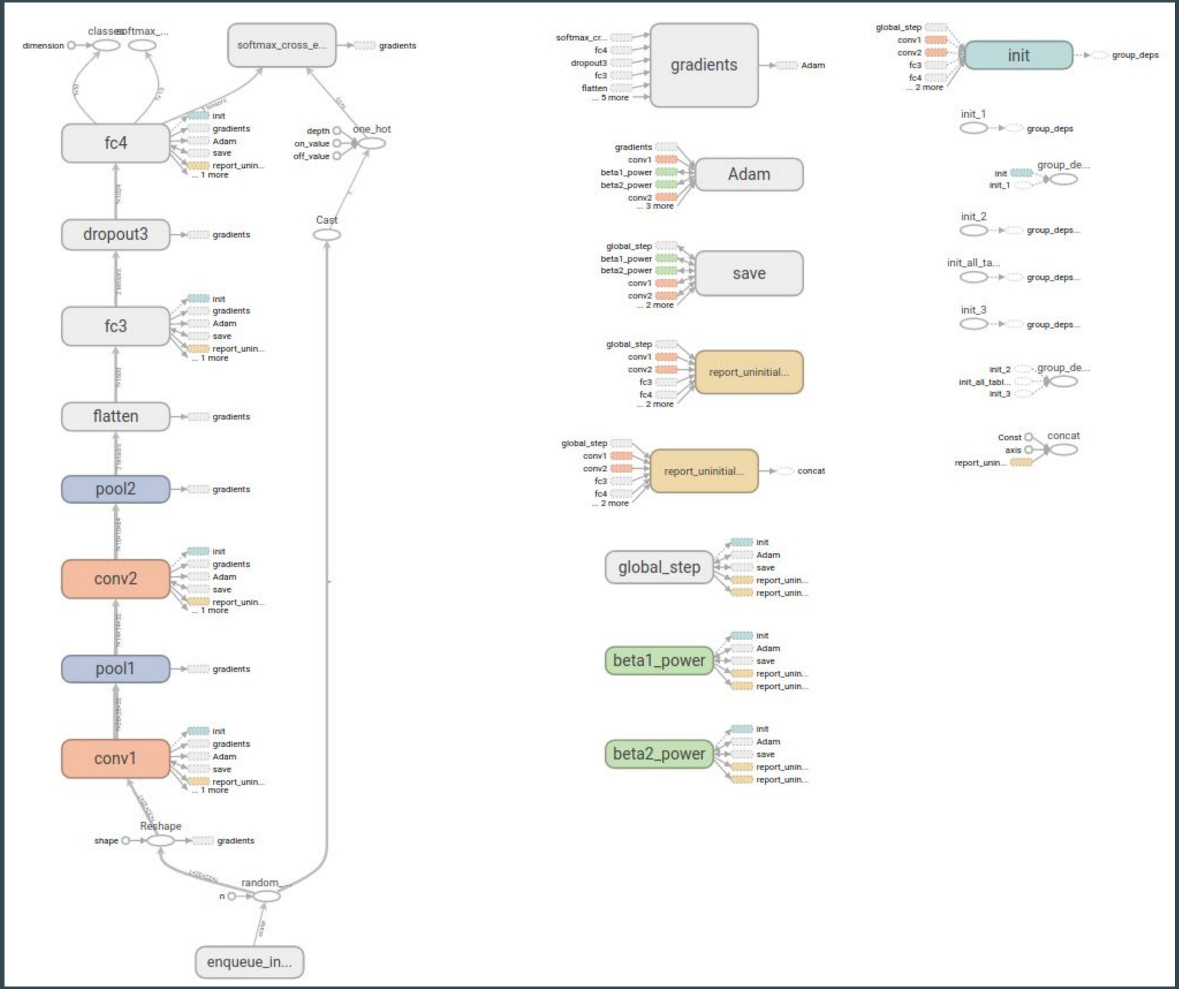
- input layers
- two blocks of 5x5 filter convolutional layers and a 2x2 pooling
- a FC layer with 1024 unit with a dropout of 0.5
- a prediction layer



```
""" Model function """
# N samples of D values and H neurons per hidden layer
width, height, channels = 32, 32, 1  # sample size
n_classes = 10  # classes

# Input layer
input_layer = tf.reshape(features['x'], [-1, width, height, 1])

# Dense layers
init = tf.contrib.layers.xavier_initializer()
conv1 = tf.layers.conv2d(inputs=input_layer,
                         filters=32, kernel_size=[5, 5],
                         name='conv1')
pool1 = tf.layers.max_pooling2d(inputs=conv1,
                                pool_size=[2, 2], strides=2,
                                name='pool1')
conv2 = tf.layers.conv2d(inputs=pool1,
                         filters=64, kernel_size=[5, 5],
                         name='conv2')
pool2 = tf.layers.max_pooling2d(inputs=conv2,
                                pool_size=[2, 2], strides=2,
                                name='pool2')
flat = tf.layers.flatten(pool2)
fc3 = tf.layers.dense(inputs=flat,
                      units=1024,
                      activation=tf.nn.relu,
                      kernel_initializer=init,
                      name='fc3')
dropout3 = tf.layers.dropout(inputs=fc3,
                             rate=0.5,
                             training=bool(
                                 mode == tf.estimator.ModeKeys.TRAIN),
                             name='dropout3')
logits = tf.layers.dense(inputs=dropout3,
                         units=10,
                         name='fc4')
```

LeNet

Model graph in Tensorboard



Simple vs LeNet Performance

After a training of 100 epochs with a batch size of 550 images:

	Accuracy (test)
Simple model	**0.9785**
LeNet	**0.9921**

Which is expected due to LeNet is designed specially for images, doing the best of the spatial correlation of images (conv. layers) and reducing less relevant information (pooling). Moreover, it has an explicit generalization (uses dropout technique)

The code is hosted in https://github.com/jonpoveda/tensorflow_handson