Introduction to tensorflow
Why do you need a deep learning framework?

Speed:
- Fast implementations of matrix multiply, convolutions and backpropagation
- Cuda implementations that are simple to use

Automatic differentiations:
- Finished implementations of the most common gradients

Reuse:
- Reuse other people’s models
- Evaluate other models correctly

Updates:
- Updates your implementation to new hardware

The more code you write yourself, the more errors
Why Tensorflow?

- The right level of abstraction
  - Good for research
  - Good for production
- No extra work to run on different devices
- A lot of functionality
- Can be run on small embedded devices and huge clusters
- Resource availability
- A lot of examples
- Pretrained models
- Tensorboard/visualization
- Can be used with several languages
Disadvantages

- A lot of functionalities
  - Many of which you will never need or use, clutter up the API
- Different frameworks within the framework
  - Interoperates only partially
- Static graph building
  - Some implementations takes extra effort
What does it look like?

```python
In [9]: import tensorflow as tf
    sess = tf.Session()
    a = tf.zeros((2,2)); b = tf.ones((2,2)); c = tf.constant([3., 5.])
    a.get_shape()

Out[9]: TensorShape([Dimension(2), Dimension(2)])

In [10]: sum_b = tf.reduce_sum(b)
    sess.run(sum_b)

Out[10]: 4.0

In [14]: mul_b_c = tf.matmul(b, tf.reshape(c, [-1, 1]))
    sess.run(mul_b_c)

Out[14]: array([[ 8.],
                [ 8.]], dtype=float32)
```
Most “standard” operations from matlab or numpy

- tf.diag(diagonal, name=None)
- tf.diag_part(input, name=None)
- tf.trace(x, name=None)
- tf.transpose(a, perm=None, name=transpose)
- tf.matrix_diag(diagonal, name=None)
- tf.matrix_diag_part(input, name=None)
- tf.matrix_band_part(input, num_lower, num_upper, name=None)
- tf.matrix_set_diag(input, diagonal, name=None)
- tf.matrix_transpose(a, name=matrix_transpose)
- tf.matmul(a, b, transpose_a=False, transpose_b=False, a_is_sparse=False, b_is_sparse=False, name=None)
- tf.batch_matmul(x, y, adj_x=None, adj_y=None, name=None)
- tf.log(x, name=None)
- tf.ceil(x, name=None)
- tf.floor(x, name=None)
- tf.maximum(x, y, name=None)
- tf.minimum(x, y, name=None)
- tf.cos(x, name=None)
- tf.sin(x, name=None)
- tf.rint(x, name=None)
- tf.tan(x, name=None)
- tf.acos(x, name=None)
- tf.asin(x, name=None)
- tf.atan(x, name=None)
- tf.lgamma(x, name=None)
Overview
Overview

Estimators
- Easy to use
- Harder to make
- Easier to reuse components etc.
Estimator

# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.
classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    # Two hidden layers of 10 nodes each.
    hidden_units=[10, 10],
    # The model must choose between 3 classes.
    n_classes=3)

# Train the Model.
classifier.train(
    input_fn=lambda:iris_data.train_input_fn(train_x, train_y, args.batch_size),
    steps=args.train_steps)

# Evaluate the model.
eval_result = classifier.evaluate(
    input_fn=lambda:iris_data.eval_input_fn(test_x, test_y, args.batch_size))

print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
Overview

Mid-level (Layers, Dataset, Metrics, Losses)
- Deep-learning/Machine learning specific
- Simpler to do common tasks
Mid-level (sweet spot)

Simple to create deep networks

```python
# Convolution Layer with 32 filters and a kernel size of 5
cnv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
# Max Pooling (down-sampling) with strides of 2 and kernel size of 2
cnv1 = tf.layers.max_pooling2d(cnv1, 2, 2)

# Convolution Layer with 64 filters and a kernel size of 3
cnv2 = tf.layers.conv2d(cnv1, 64, 3, activation=tf.nn.relu)
# Max Pooling (down-sampling) with strides of 2 and kernel size of 2
cnv2 = tf.layers.max_pooling2d(cnv2, 2, 2)

# Flatten the data to a 1-D vector for the fully connected layer
fc1 = tf.contrib.layers.flatten(cnv2)

# Fully connected layer (in tf contrib folder for now)
fc1 = tf.layers.dense(fc1, 1024)
# Apply Dropout (if is_training is False, dropout is not applied)
fc1 = tf.layers.dropout(fc1, rate=dropout, training=is_training)

# Output layer, class prediction
out = tf.layers.dense(fc1, n_classes)
```
Overview

Low-level
- Not specific for machine learning
  - Except for gradient calculation
- General computation/Linear algebra
- Simplifies GPU programming
- Same code run on many different platforms
Low-level

- Testing out new building blocks
  - New types of convolutions
  - New losses
  - New optimization functions
- More code = more errors
Computational graph
Computational graph

Separating definition of computations from execution.
- Build a computational graph
- Use a session to run operations in the graph
Session

Responsible for managing resources. Handles execution on different devices. Keep variables in memory for the lifetime of a session.
Computational graph

import tensorflow as tf
a = tf.add(2, 3)
Computational graph

import tensorflow as tf
a = tf.add(2, 3)
Computational graph

```python
import tensorflow as tf
a = tf.add(2, 3)
print(a)
>> Tensor("Add:0", shape=(), dtype=int32)
```
Computational graph

import tensorflow as tf
a = tf.add(2, 3)
print a
>> Tensor("Add:0", shape=(), dtype=int32)

This is graph definition, not computation
Evaluating the computational graph

import tensorflow as tf
a = tf.add(2, 3)
sess = tf.Session()
print sess.run(a)
>> 8
sess.close()
import tensorflow as tf
a = tf.add(3, 5)
# with clause takes care
# of sess.close()
with tf.Session() as sess:
    print sess.run(a)
A larger graph

```python
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.mul(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as sess:
    op3 = sess.run(op3)
```
A larger graph - running parts only

\[
x = 2 \\
y = 3 \\
op1 = \text{tf.add}(x, y) \\
op2 = \text{tf.mul}(x, y) \\
useless = \text{tf.mul}(x, \text{op1}) \\
op3 = \text{tf.pow}(\text{op2}, \text{op1}) \\
\]

```
with \text{tf.Session()}\ as\ \text{sess}:
    \text{op3 = sess.run(op3)}
```
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.mul(x, y)
useless = tf.mul(x, op1)
op3 = tf.pow(op2, op1)

with tf.Session() as sess:
    op3, not_useless = sess.run([op3, useless])
Parts of the graph

- Operators (add, matmul, conv2d…)
- Constants
- Tensors (temporary data)
- Variables (Values consistent over multiple graph-executions)
Creating constants

import tensorflow as tf
a = tf.constant([2, 2], name="a")
b = tf.constant([[0, 1], [2, 3]], name="b")
x = tf.add(a, b, name="add")
y = tf.mul(a, b, name="mul")
with tf.Session() as sess:
    x, y = sess.run([x, y])
    print x, y
# >> [5 8] [6 12]
Like numpy

```python
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
tf.ones(shape, dtype=tf.float32, name=None)
tf.fill(dims, value, name=None)
tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]
tf.linspace(10.0, 13.0, 4) ==> [10.0 11.0 12.0 13.0]
tf.range(start, limit, delta) ==> [3, 6, 9, 12, 15]
```
Random generated “constants”

New each execution

tf.set_random_seed(seed) #To generate same randoms each times

tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None, name=None)
Tensor (tf.Tensor)

- Input and output for operations
- Live only for one execution
- Temporary data that flow through the graph
- To keep:
  - Extract to numpy/python
  - Assign to Variable

Tensor objects are not iterable
for i in tf.range(4): # TypeError
for i in tf.unstack(tf.range(4)) #Works

https://blog.interactivethings.com/notes-from-openvi
s-conference-2016-577c80cd7a01
Problems with tensors

- Don’t have values when they are created, only during graph execution.
- Can have flexible shape/size

Looping through tensors:
- Python for-loop with tf.unstack etc.
  - Easy to interpret and debug
  - You need to know the size of the dimension you are iterating
- Using tf.py_func
  - Get numpy array, and do whatever you want in a function
- Use tf.scan, tf.while_loop
  - Fast, but hard to debug
- Don’t use vectorized functions

https://blog.interactivethings.com/notes-from-openvis-conference-2016-577c80cd7a01
tf.Variables()

# create variable a with scalar value
a = tf.Variable(2, name="scalar")
# create variable b as a vector
b = tf.Variable([2, 3], name="vector")
# create variable c as a 2x2 matrix
c = tf.Variable([[0, 1], [2, 3]], name="matrix")
# create variable W as 784 x 10 tensor, filled with zeros
W = tf.Variable(tf.zeros([784,10]))

Big V in tf.Variables, is because Variables is a class
tf.Variables() live in the graph world

Big V in tf.Variables, is because Variables is a class.
- Live for the lifetime of a Session
- To keep after a session is dead
  - Save checkpoint
  - Extract to numpy/python and store however you want

# create variable a with scalar value
a = tf.Variable(2, name="scalar")

# create variable b as a vector
b = tf.Variable([2, 3], name="vector")

# create variable c as a 2x2 matrix
# c = tf.Variable([[0, 1], [2, 3]], name="matrix")
# create variable W as 784 x 10 tensor, filled with zeros
W = tf.Variable(tf.zeros([784,10]))
Variables have to be initialized

The easiest way is initializing all variables at once:

```python
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
```

#Initialize only a subset of variables:

```python
init_ab = tf.variables_initializer([a, b], name="init_ab")
with tf.Session() as sess:
    sess.run(init_ab)
```

Initialize a single variable

```python
W = tf.Variable(tf.zeros([784, 10]))
with tf.Session() as sess:
    sess.run(W.initializer)
```

If you run the initialization again, the variables are reset
Assigning to variables in the graph-world

W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
    sess.run(Winitializer)
    print sess.run(W)
Assigning to variables in the graph-world

\[
W = \text{tf.Variable}(10) \\
W.\text{assign}(100) \\
\text{with tf.Session() as sess:} \\
\hspace{1cm} \text{sess.run(W.initializer)} \\
\hspace{1cm} \text{print sess.run(W)} \ # >> 10
\]

Why?
Assigning to variables in the graph-world

```python
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    print sess.run(W) # >> 10
```

Why?
Assign works in the **graph-world** and create an operator for assigning to `W`
Assigning to variables in the graph-world

```python
W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    sess.run(assign_op)
print sess.run(W)  # >> 100
```

Why?
Assign works in the **graph-world** and create an operator for assigning to W
Assigning to variables in the numbers-world

W = tf.Variable(10)
with tf.Session() as sess:
    sess.run(W.initializer)
    print sess.run(W, feed_dict={W: 100})
    # >> 100
    print sess.run(W) # >> 10

feed_dict input variables temporarily into any point in the graph (any feedable tensor tf.Graph.is_feedable(tensor))
Distributed computation

# Creates a graph.
with tf.device('/gpu:2'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='b')
    c = tf.matmul(a, b)

# Creates a session with log_device_placement set to True.
sess=tf.Session(config=tf.ConfigProto(log_device_placement=True))

# Runs the op.
print sess.run(c)
Building a deep network with tensorflow

The dirty details
Basic setup and imports

- Numpy is generally needed
- Tensorflow

```
# Imports
import numpy as np
import tensorflow as tf
```
Inputting data - feeding

Endless possibilities...
Data can be feed and and retrieved to and from anywhere in the grap
`sess = tf.Session()`
`sess.run(W, feed_dict={b: 3})`
You can also use string for the tensor names
`sess.run("W:0", feed_dict={"b:0": 3})`
Why use any other method?
Inputting data - python generator

You don’t want reading data to block your application. (Keep your GPU running, if you have one)

- Continues loop after yield
- When asked for a new value the generator continues its loop

# a generator that yields items instead of returning a list

def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1

sum_of_first_n = sum(firstn(1000000))

for i in firstn(5):
    print(i)
Inputting data - generator to tensorflow

```python
# a generator that yields items instead of returning a list
def first100():
    num = 0
    while num < 100:
        yield num
        num += 1

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(first100, output_types=[tf.int64], output_shapes=())
# Get the actual tensor
tensor_value = dataset.make_one_shot_iterator().get_next()
```
Inputting data - generator to tensorflow

```python
# a generator that yields items instead of returning a list
def first100():
    num = 0
    while num < 100:
        yield num
    num += 1

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(first100, output_types=[tf.int64], output_shapes=())
# Get the actual tensor
tensor_value = dataset.make_one_shot_iterator().get_next()
```
## Inputting data - generator to tensorflow

```python
# a generator that yields items instead of returning a list
def first100():
    num = 0
    while num < 100:
        yield num
    num += 1

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(first100, output_types=[tf.int64], output_shapes=())
# Get the actual tensor
tensor_value = dataset.make_one_shot_iterator().get_next()
```

```python
# a generator that yields items instead of returning a list
def firstn(n):
    num = 0
    while num < n:
        yield num
    num += 1

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(lamba: firstn(100), output_types=[tf.int64], output_shapes=())
# Get the actual tensor
tensor_value = dataset.make_one_shot_iterator().get_next()
```
Inputting data - reading images

Read data with whatever you want...

```python
def image_data(filenames):
    import cv2
    num = 0
    for i, f in enumerate(filenames):
        yield cv2.imread(f), i

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(lambda: image_data(glob.glob('/data/**/*.png')),
                                          output_types=[tf.uint8, tf.int64],
                                          output_shapes=([1, None, None, 3], ()))

# Get the actual tensors
image, label = dataset.make_one_shot_iterator().get_next()
```
tf.data.Dataset - process your data

def augment_data(img, label):
    img = tf.image.random_crop(img, [224, 224])
    img = tf.image.random_brightness(img, max_delta=40)
    return img, label

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(lambda: image_data(glob.glob('/data/**/*.png')),
    output_types=[tf.uint8, tf.int64],
    output_shapes=((1, None, None, 3), ()))

# Process the data
dataset = dataset.map(augment_data, num_parallel_calls=4)

# Get the actual tensors
image, label = dataset.make_one_shot_iterator().get_next()
tf.data.Dataset - process your data

def augment_data(img, label):
    img = tf.image.random_crop(img, [224, 224])
    img = tf.image.random_brightness(img, max_delta=40)
    return img, label

# Create tensorflow dataset from generator
dataset = tf.data.Dataset.from_generator(lambda: image_data(glob.glob('/data/**/*.png')),
                                          output_types=[tf.uint8, tf.int64],
                                          output_shapes=((1, None, None, 3), ()))

# Process the data
dataset = dataset.map(augment_data, num_parallel_calls=4)
dataset = dataset.prefetch(20).batch(8)
# Get the actual tensors
image, label = dataset.make_one_shot_iterator().get_next()
Training your model

```python
# Get the actual tensors
image, label = dataset.make_one_shot_iterator().get_next()

x = tf.layers.conv2d(image, 256, 3)
loss = tf.reduce_mean((x - label)**2)

gradient_decent = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)

sess = tf.Session()
sess.run(gradient_decent)
```
Saving and restoring models

You can decide what variables you are saving or restoring when creating your Saver with a `var_list`.

```python
saver = tf.train.Saver(var_list=tf.global_variables())
saver.save(sess, 'checkpoint_dir')
saver.restore(sess, 'checkpoint_dir')
```
MonitoredSession

Helps you:
- Save or restore your variables
- Save summaries
- Run other Hooks like profiling

Create hooks, otherwise use Session as normal.
Tensorboard and summaries

- SummarySaverHook, saves your summaries to an output_dir
- run $tensorboard --logdir 'output_dir'
- open webbrowser to localhost:6006

tf.summary.scalar('loss', loss)
tf.summary.image('image', img, max_outputs=5)
tf.summary.histogram('logits', logits)
Reusing your model

- Run new data through the same network
- Easy to mess up

```python
def model(img, seg):
    x = tf.layers.conv2d(img, 32, 5, strides=(2, 2), padding='same', activation=tf.nn.relu)
    x = tf.layers.conv2d(x, 64, 5, strides=(2, 2), padding='same', activation=tf.nn.relu)
    x = tf.layers.conv2d(x, 1, 1, padding='same')
    x = tf.image.resize_images(x, [512, 512])
    loss = tf.reduce_mean((x - seg)**2)
    return x, loss

def main():
    image_names, segmentation_names = kitti_image_filenames('/data/data_road')

    img, seg = kitti_generator_from_filenames(
        image_names[:-3],
        segmentation_names[:-3],
        batch_size=8)

    img_val, seg_val = kitti_generator_from_filenames(
        image_names[-3:], segmentation_names[-3:], batch_size=8)

    with tf.variable_scope('model'):
        logits, loss = model(img, seg)

    with tf.variable_scope('model'):
        logits_val, loss_val = model(img_val, seg_val)

    with tf.variable_scope('model', reuse=True):
        logits_val, loss_val = model(img_val, seg_val)
```
Loading a pretrained model - easy way

Tensorflow hub:
- Very easy
- Problem with fixed image size
- Not a “nice” way to get intermediate results

```python
module = hub.Module("https://tfhub.dev/google/imagenet/mobilenet_v2_140_224/classification/2")

height, width = hub.get_expected_image_size(module)

images = ... # A batch of images with shape [batch_size, height, width, 3].

logits = module(images) # Logits with shape [batch_size, num_classes].
```
Loading a pretrained model - harder way

Tensorflow slim/detection api:
- More flexible
- Get endpoints
- More work

https://github.com/tensorflow/models/tree/master/research/slim
https://github.com/tensorflow/models/tree/master/research/object_detection

```python
from tensorflow.contrib import slim
from tensorflow.contrib.slim import nets
with slim.arg_scope(nets.resnet_v2.resnet_arg_scope()):
    out, end_points = nets.resnet_v2.resnet_v2_50(x, is_training=is_training, global_pool=False)
sess = None

enc1 = end_points['resnet_v2_50/block1']
enc2 = end_points['resnet_v2_50/block2']
enc3 = end_points['resnet_v2_50/block3']

saver = tf.train.Saver(
    var_list=[v for v in tf.global_variables() if 'resnet_v2_50' in v.name]
)
saver.restore(sess, 'resnet_v2_50.ckpt')
```
### Loading a pretrained model - harder way

<table>
<thead>
<tr>
<th>Model</th>
<th>TF-Slim File</th>
<th>Checkpoint</th>
<th>Top-1 Accuracy</th>
<th>Top-5 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V1</td>
<td>Code</td>
<td>inception_v1_2016_08_28.tar.gz</td>
<td>69.8</td>
<td>89.6</td>
</tr>
<tr>
<td>Inception V2</td>
<td>Code</td>
<td>inception_v2_2016_08_28.tar.gz</td>
<td>73.9</td>
<td>91.8</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Code</td>
<td>inception_v3_2016_08_28.tar.gz</td>
<td>78.0</td>
<td>93.9</td>
</tr>
<tr>
<td>Inception V4</td>
<td>Code</td>
<td>inception_v4_2016_09_09.tar.gz</td>
<td>80.2</td>
<td>95.2</td>
</tr>
<tr>
<td>Inception-ResNet-v2</td>
<td>Code</td>
<td>inception_resnet_v2_2016_08_30.tar.gz</td>
<td>80.4</td>
<td>96.3</td>
</tr>
<tr>
<td>ResNet V1 50</td>
<td>Code</td>
<td>resnet_v1_50_2016_08_28.tar.gz</td>
<td>75.2</td>
<td>92.2</td>
</tr>
<tr>
<td>ResNet V1 101</td>
<td>Code</td>
<td>resnet_v1_101_2016_08_28.tar.gz</td>
<td>76.4</td>
<td>92.9</td>
</tr>
<tr>
<td>ResNet V1 152</td>
<td>Code</td>
<td>resnet_v1_152_2016_08_28.tar.gz</td>
<td>76.8</td>
<td>93.2</td>
</tr>
<tr>
<td>ResNet V2 50*</td>
<td>Code</td>
<td>resnet_v2_50_2017_04_14.tar.gz</td>
<td>75.6</td>
<td>92.8</td>
</tr>
<tr>
<td>ResNet V2 101*</td>
<td>Code</td>
<td>resnet_v2_101_2017_04_14.tar.gz</td>
<td>77.0</td>
<td>93.7</td>
</tr>
<tr>
<td>ResNet V2 152*</td>
<td>Code</td>
<td>resnet_v2_152_2017_04_14.tar.gz</td>
<td>77.8</td>
<td>94.1</td>
</tr>
<tr>
<td>ResNet V2 200</td>
<td>Code</td>
<td>TBA</td>
<td>79.9*</td>
<td>95.2*</td>
</tr>
<tr>
<td>VGG 16</td>
<td>Code</td>
<td>vgg_16_2016_08_28.tar.gz</td>
<td>71.5</td>
<td>89.8</td>
</tr>
<tr>
<td>VGG 19</td>
<td>Code</td>
<td>vgg_19_2016_08_28.tar.gz</td>
<td>71.1</td>
<td>89.8</td>
</tr>
<tr>
<td>MobileNet_v1_1.0_224</td>
<td>Code</td>
<td>mobilenet_v1_1.0_224.tar.gz</td>
<td>70.9</td>
<td>89.9</td>
</tr>
<tr>
<td>MobileNet_v1_0.5_160</td>
<td>Code</td>
<td>mobilenet_v1_0.5_160.tar.gz</td>
<td>59.1</td>
<td>81.9</td>
</tr>
<tr>
<td>MobileNet_v1_0.25_128</td>
<td>Code</td>
<td>mobilenet_v1_0.25_128.tar.gz</td>
<td>41.5</td>
<td>66.3</td>
</tr>
<tr>
<td>MobileNet_v2_1.4_224**</td>
<td>Code</td>
<td>mobilenet_v2_1.4_224.tar.gz</td>
<td>74.9</td>
<td>92.5</td>
</tr>
<tr>
<td>MobileNet_v2_1.0_224**</td>
<td>Code</td>
<td>mobilenet_v2_1.0_224.tar.gz</td>
<td>71.9</td>
<td>91.0</td>
</tr>
<tr>
<td>NASNet-A_Mobile_224#</td>
<td>Code</td>
<td>nasnet-a_mobile_04_10_2017.tar.gz</td>
<td>74.0</td>
<td>91.6</td>
</tr>
<tr>
<td>NASNet-A_Large_331#</td>
<td>Code</td>
<td>nasnet-a_large_04_10_2017.tar.gz</td>
<td>82.7</td>
<td>96.2</td>
</tr>
<tr>
<td>PNASNet-5_Large_331</td>
<td>Code</td>
<td>pnasnet-5_larger_12_13.tar.gz</td>
<td>82.9</td>
<td>96.2</td>
</tr>
<tr>
<td>PNASNet-5_Mobile_224</td>
<td>Code</td>
<td>pnasnet-5_mobile_12_13.tar.gz</td>
<td>74.2</td>
<td>91.9</td>
</tr>
</tbody>
</table>
Endnote - protip

- Create global step

```python
step = tf.train.get_or_create_global_step()

with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
  train_op = tf.train.AdamOptimizer().minimize(
      loss,
      global_step=step)
```