FFI Forsvarets forskningsinstitutt

# 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 availablility
- 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

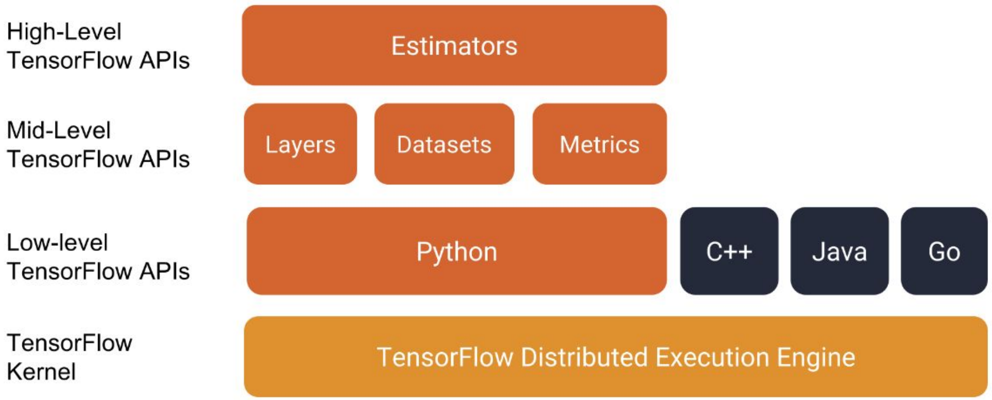
# What does it look like?

```
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.lbeta(x, name=lbeta)`
- `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

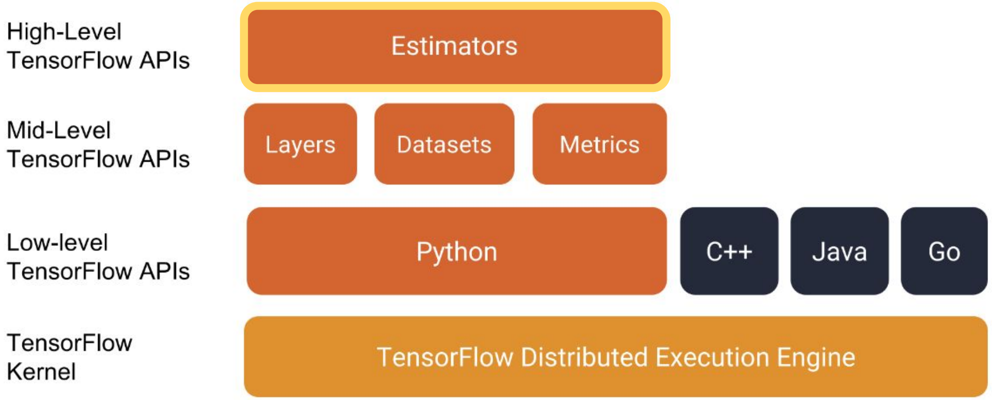

High-Level TensorFlow APIs
Estimators
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine

# Overview

Estimators

- Easy to use
- Harder to make

- Easier to reuse componets 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)
```

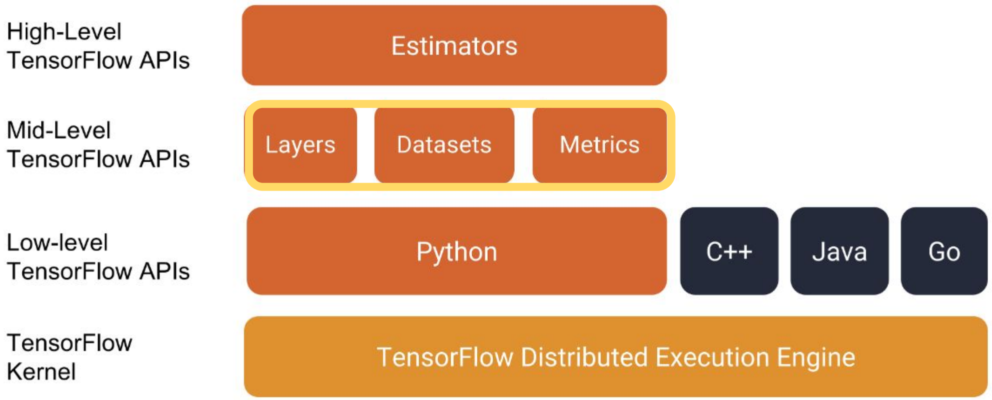
```
# Train the Model.
classifier.train(
    input_fn=lambda:iris_data.train_input_fn(train_x, train_y, args.batch_size),
    steps=args.train_steps)
```

```
# 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)

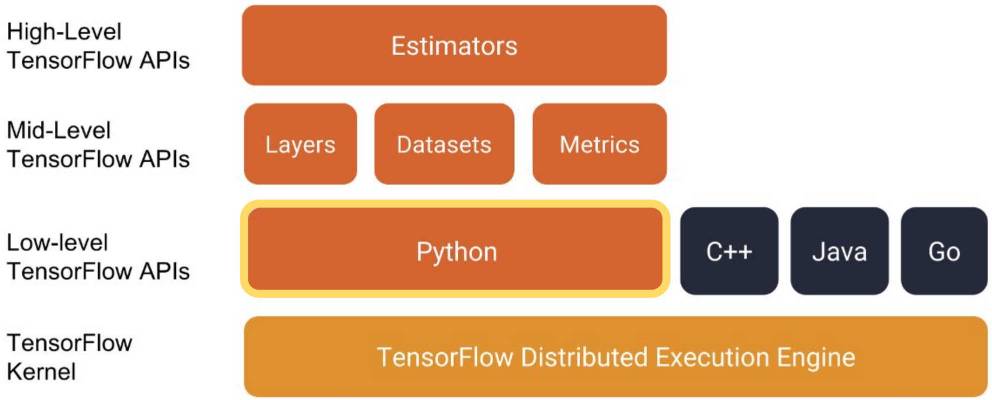
Simple to create deep networks

```
# Convolution Layer with 32 filters and a kernel size of 5
conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
# Max Pooling (down-sampling) with strides of 2 and kernel size of 2
conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

# Convolution Layer with 64 filters and a kernel size of 3
conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
# Max Pooling (down-sampling) with strides of 2 and kernel size of 2
conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

# Flatten the data to a 1-D vector for the fully connected layer
fc1 = tf.contrib.layers.flatten(conv2)

# 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

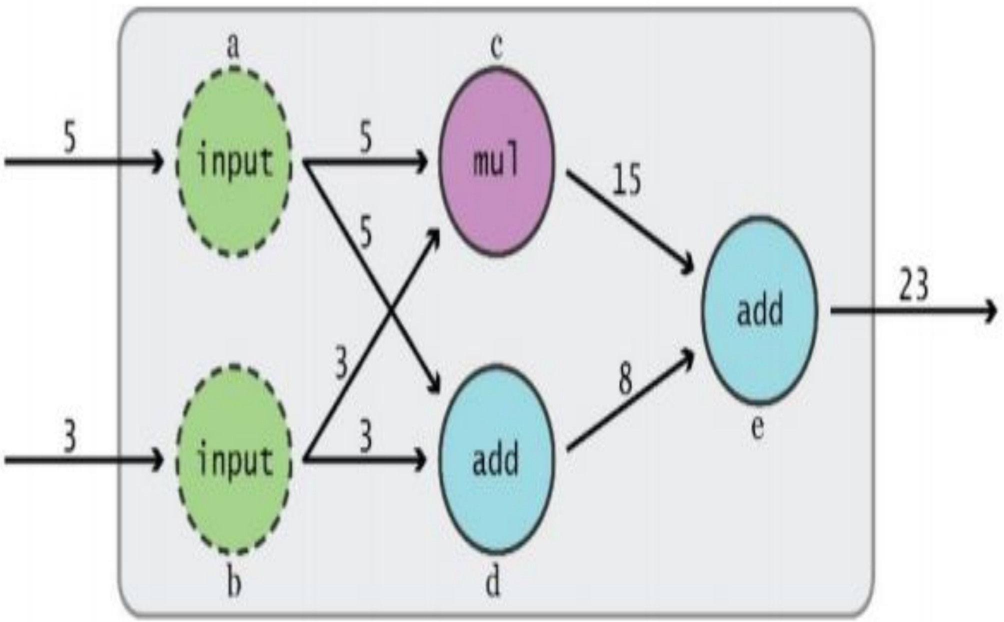
```
def conv_layer(x, filters=100, k_size=3, name='conv'):
  with tf.name_scope(name):
    filters_in = x.get_shape().as_list()[-1]
    kernel = tf.Variable(tf.random_normal([k_size, k_size, filters_in, filters],
                                          stddev=np.sqrt(2) / (filters_in*k_size**2)),
                         name='W')
    bias = tf.Variable(tf.zeros([filters]), name='bias')
    out = tf.nn.relu(tf.nn.conv2d(x, kernel, strides=(1, 2, 2, 1), padding='SAME') + bias)
    tf.summary.histogram('activations', out)
    return out
```

# Computational graph

# Computational graph

Separating definition of computations from execution.

- Build a computational graph
- Use a session to run operations in the graph



# Session

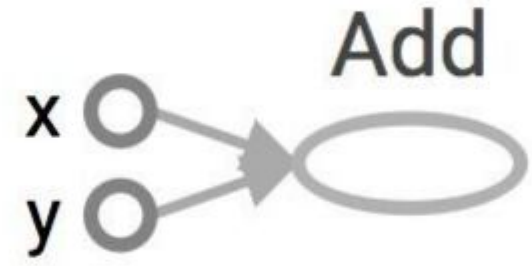
Responsible for managing resources.

Handles execution on different devices.

Keep variables in memory for the lifetime of a session.

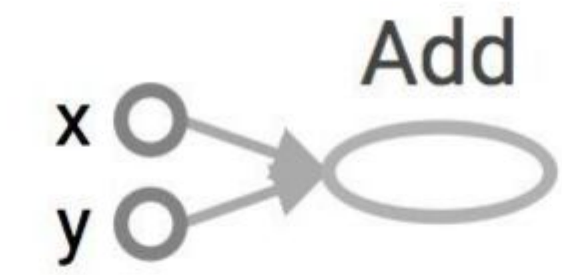
# Computational graph

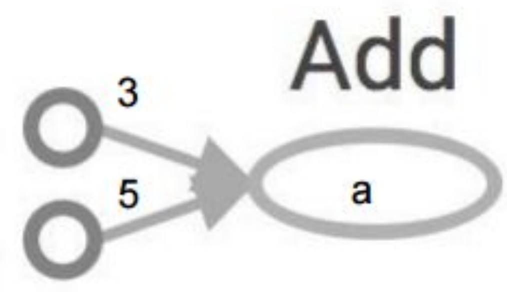
```
import tensorflow as tf
 a = tf.add(2, 3)
```



# Computational graph

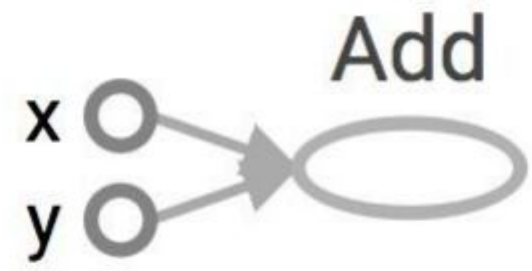
import tensorflow as tf
 a = tf.add(2, 3)

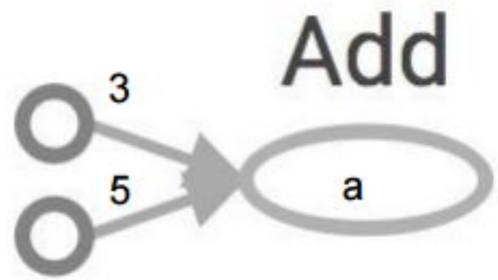


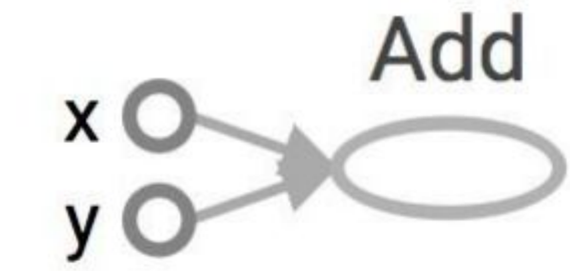
# Computational graph

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



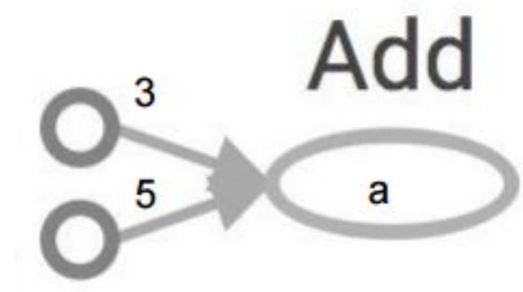



# Computational graph



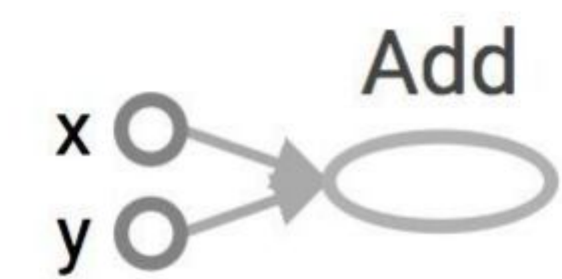

```
import tensorflow as tf
 a = tf.add(2, 3)
print a
>> Tensor("Add:0", shape=(), dtype=int32)
```

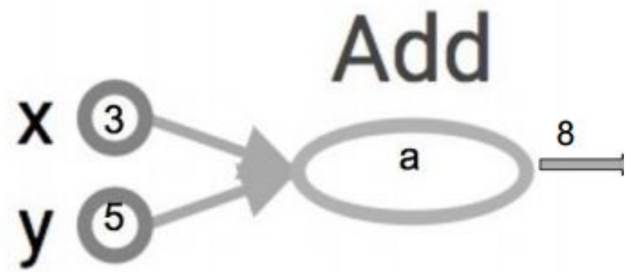
**This is graph definition, not computation**



# Evaluating the computational graph

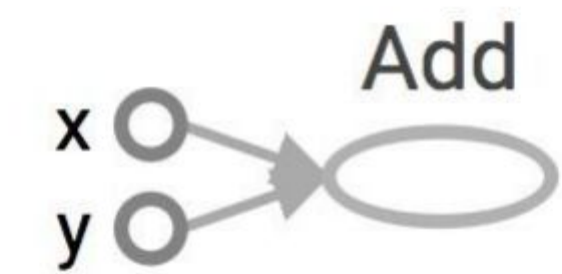
```
import tensorflow as tf
 a = tf.add(2, 3)
sess = tf.Session()
print sess.run(a)
>> 8
sess.close()
```



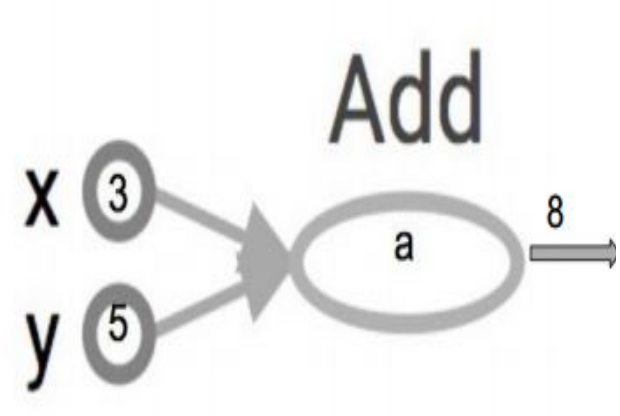



# Evaluating the computational graph

**With statement can clean up session by calling .close()**

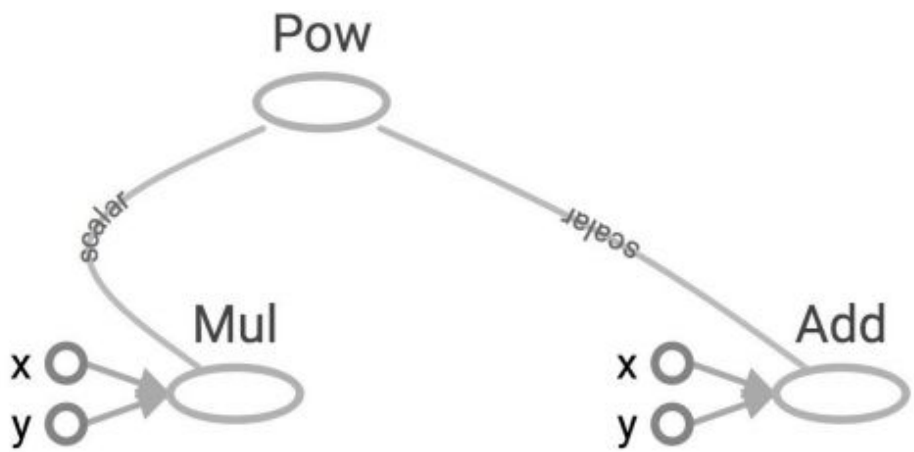



```
import tensorflow as tf
a = tf.add(3, 5)
# with clause takes care
# of sess.close()
with tf.Session() as sess:
    print sess.run(a)
```



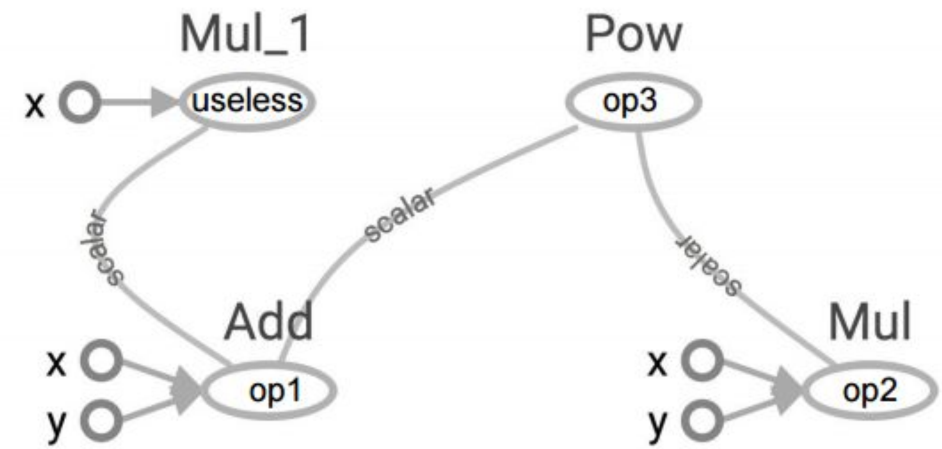
# A larger graph

```
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)
```



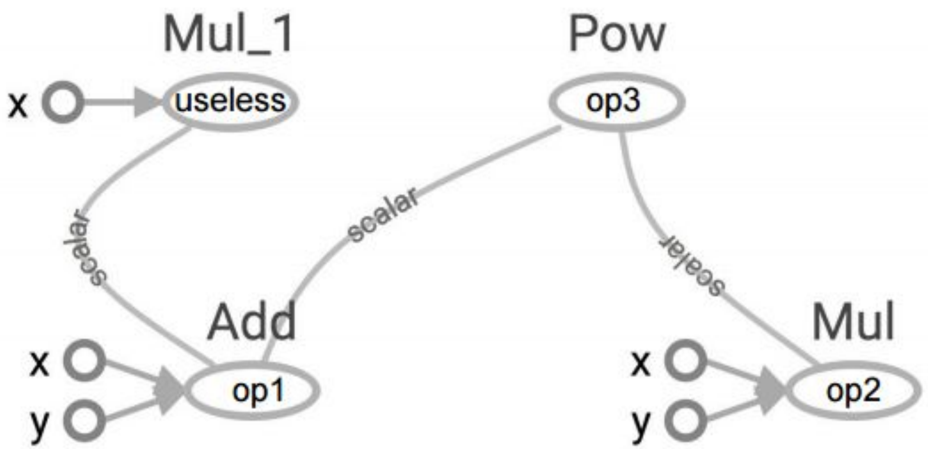
# A larger graph - running parts only

```
x = 2
y = 3
op1 = 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 = sess.run(op3)
```



# A larger graph - running multiple parts

```
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]
```

**“Graph world” - Tensorflow**

**“Numbers world” - numpy**

# Like numpy

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)

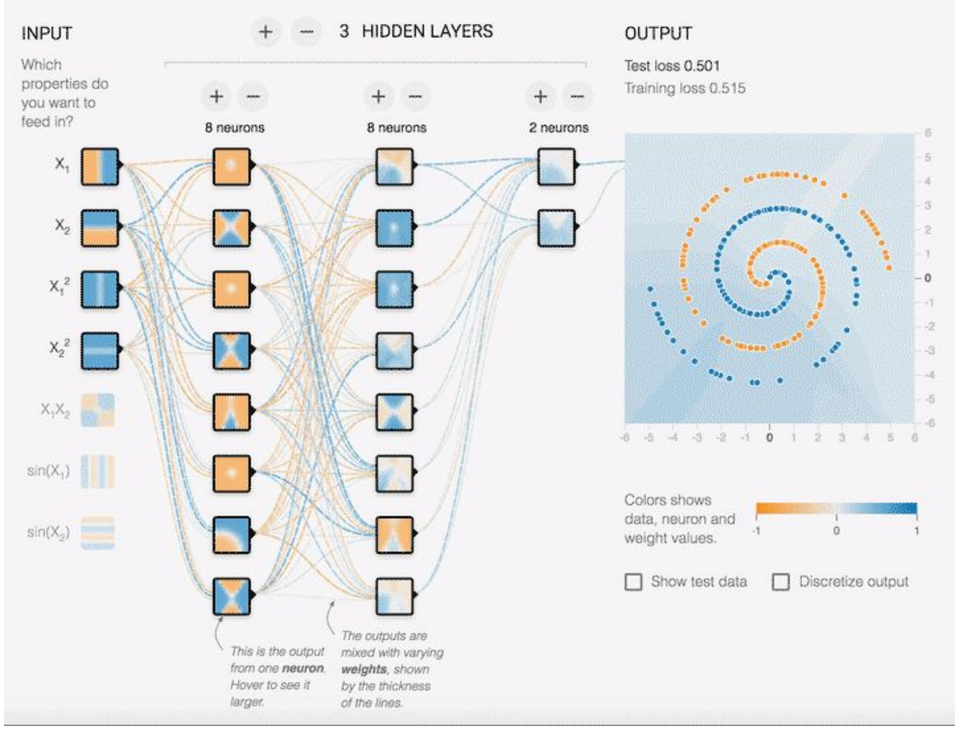
# Tensor (tf.Tensor)

- 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



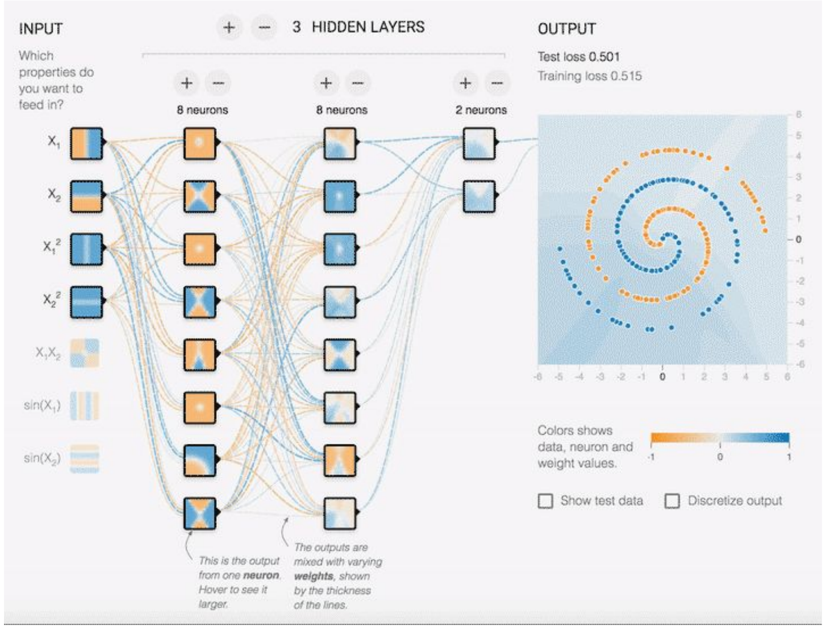

https://blog.interactivethings.com/notes-from-openvis-conference-2016-577c80cd7a01

# Problems with tensors

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

Looping through tensor:

- Python for-loop with tf.unstack etc.
  - Easy to interpret and debug
  - You need to know the size of the dimension your 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:

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
#Initialize only a subset of variables:
init_ab = tf.variables_initializer([a, b], name="init_ab")
with tf.Session() as sess:
    sess.run(init_ab)
```

Initialize a single variable

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

**If you run the initialization <u>again,</u> 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(W.initializer)
    print sess.run(W)
```

# Assigning to variables in the graph-world

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

Why?

# Assigning to variables in the graph-world

```
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

```
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

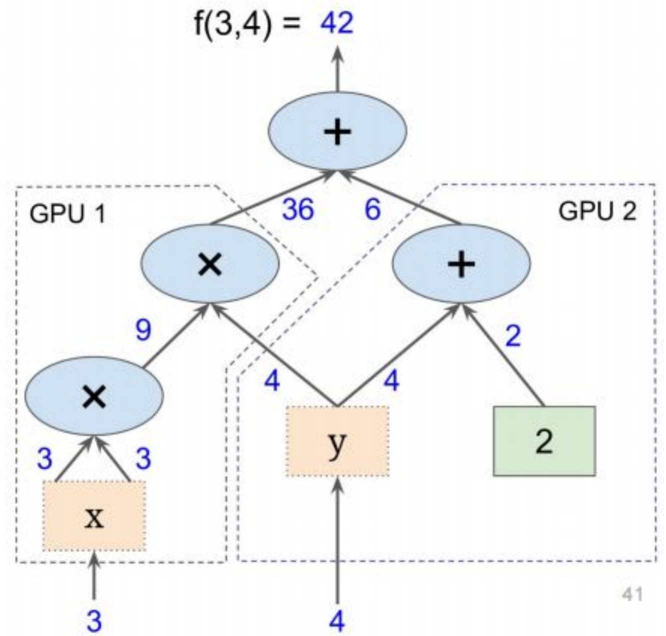
# Assigning to variables in the numbers-world

```
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=Tru
))
# Runs the op.
 print sess.run(c)
```



Forsvarets forskningsinstitutt

# Building a deep network with tensorflow

The dirty details

# Basic setup and imports

- Numpy is generally needed
- Tensorflow

```python
# 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

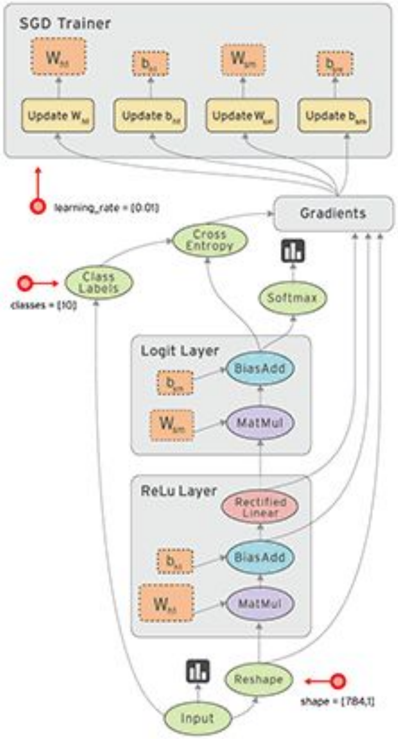
sess = tf.Session()

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 you application. (Keep your GPU running, if you have one)

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

```python
# 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

```
# 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

```
# 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(lambda: 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...

```
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

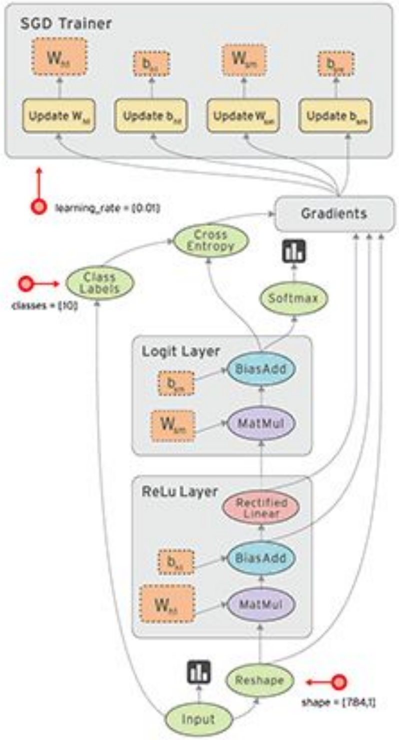
```
def augment_data(img, label):
    img = tf.image.random_crop(img, [224, 224])
    img = tf.image.random_brightness(img, max_delta=40)
    return img, label

# 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 variabels you are saving or restoring when creating your Saver with a **var_list**.

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

# MonitoredSession

Helps you:

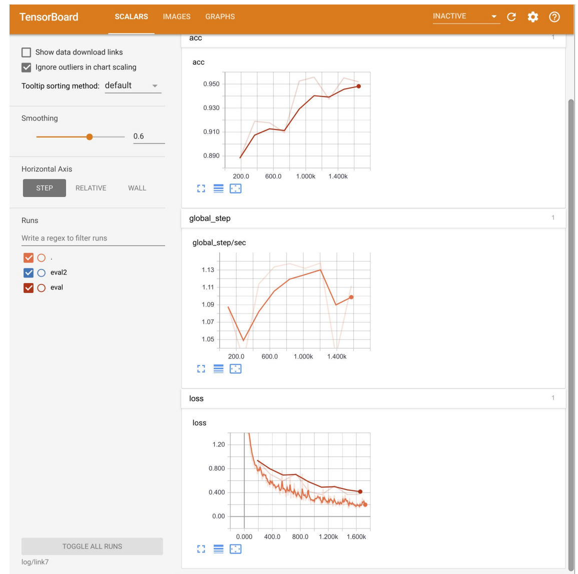
- Save or restore your variables
- Save summaries
- Run other Hooks like profiling

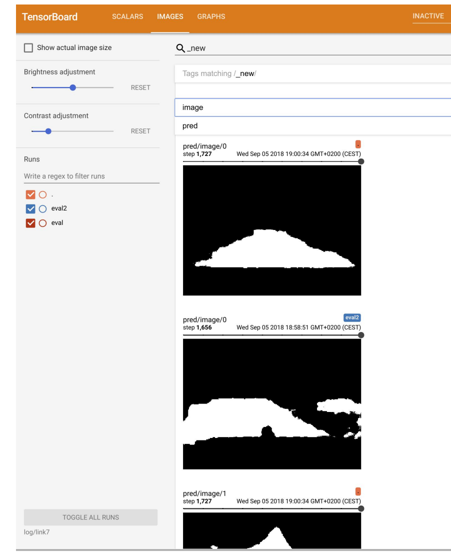
Create hooks, otherwise use Session as normal.

```
saver_hook = tf.train.CheckpointSaverHook('logs', save_secs=600)
summary_saver_hook = tf.train.SummarySaverHook(
    summary_op=tf.summary.merge_all(),
    output_dir='logs',
    save_secs=600
)

with tf.train.SingularMonitoredSession(
        hooks=[saver_hook, summary_saver_hook],
        checkpoint_dir='logs') as sess:
    while not sess.should_stop():
        _, loss_, step_ = sess.run([train_op, loss, step])
```

# 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

```
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].
```

```
print(tf.get_default_graph().get_operations())
tensor = tf.get_default_graph()\
    .get_tensor_by_name(
    'model/module_apply_default/resnet_v2_50/block2/unit_4/bottleneck_v2/conv3/Conv2D:0'
)
```

# 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

```
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

| Model | TF-Slim File | Checkpoint | Top-1 Accuracy | Top-5 Accuracy |
|---|---|---|---|---|
| Inception V1 | Code | inception_v1_2016_08_28.tar.gz | 69.8 | 89.6 |
| Inception V2 | Code | inception_v2_2016_08_28.tar.gz | 73.9 | 91.8 |
| Inception V3 | Code | inception_v3_2016_08_28.tar.gz | 78.0 | 93.9 |
| Inception V4 | Code | inception_v4_2016_09_09.tar.gz | 80.2 | 95.2 |
| Inception-ResNet-v2 | Code | inception_resnet_v2_2016_08_30.tar.gz | 80.4 | 95.3 |
| ResNet V1 50 | Code | resnet_v1_50_2016_08_28.tar.gz | 75.2 | 92.2 |
| ResNet V1 101 | Code | resnet_v1_101_2016_08_28.tar.gz | 76.4 | 92.9 |
| ResNet V1 152 | Code | resnet_v1_152_2016_08_28.tar.gz | 76.8 | 93.2 |
| ResNet V2 50^ | Code | resnet_v2_50_2017_04_14.tar.gz | 75.6 | 92.8 |
| ResNet V2 101^ | Code | resnet_v2_101_2017_04_14.tar.gz | 77.0 | 93.7 |
| ResNet V2 152^ | Code | resnet_v2_152_2017_04_14.tar.gz | 77.8 | 94.1 |
| ResNet V2 200 | Code | TBA | 79.9* | 95.2* |
| VGG 16 | Code | vgg_16_2016_08_28.tar.gz | 71.5 | 89.8 |
| VGG 19 | Code | vgg_19_2016_08_28.tar.gz | 71.1 | 89.8 |
| MobileNet_v1_1.0_224 | Code | mobilenet_v1_1.0_224.tgz | 70.9 | 89.9 |
| MobileNet_v1_0.50_160 | Code | mobilenet_v1_0.50_160.tgz | 59.1 | 81.9 |
| MobileNet_v1_0.25_128 | Code | mobilenet_v1_0.25_128.tgz | 41.5 | 66.3 |
| MobileNet_v2_1.4_224^* | Code | mobilenet_v2_1.4_224.tgz | 74.9 | 92.5 |
| MobileNet_v2_1.0_224^* | Code | mobilenet_v2_1.0_224.tgz | 71.9 | 91.0 |
| NASNet-A_Mobile_224# | Code | nasnet-a_mobile_04_10_2017.tar.gz | 74.0 | 91.6 |
| NASNet-A_Large_331# | Code | nasnet-a_large_04_10_2017.tar.gz | 82.7 | 96.2 |
| PNASNet-5_Large_331 | Code | pnasnet-5_large_2017_12_13.tar.gz | 82.9 | 96.2 |
| PNASNet-5_Mobile_224 | Code | pnasnet-5_mobile_2017_12_13.tar.gz | 74.2 | 91.9 |

# Endnote - protip

- Create global step

```
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)
```