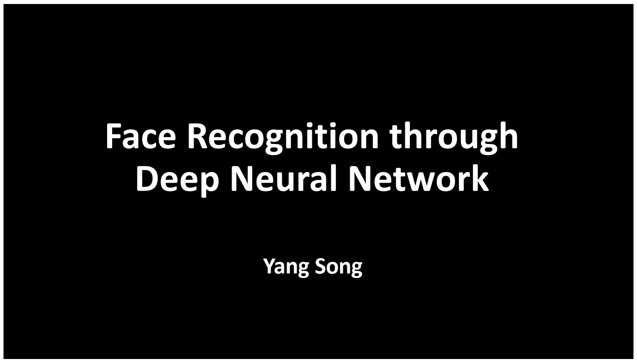
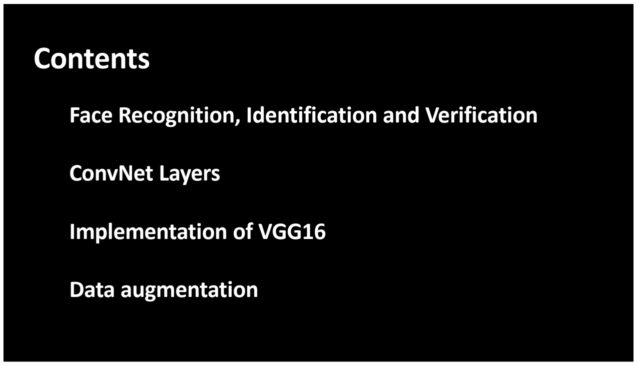


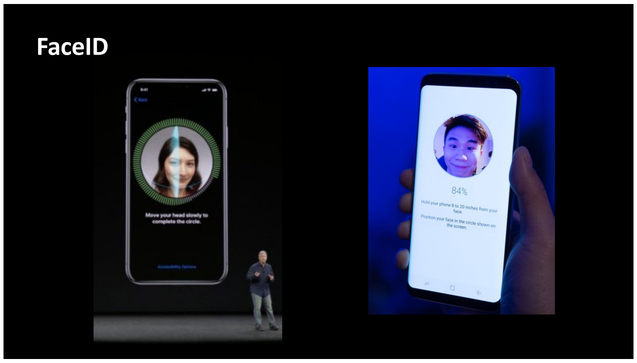
Face Recognition through Deep Neural Network

Yang Song

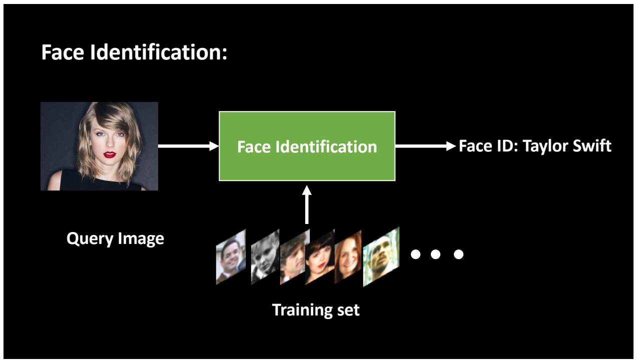


Contents

- Face Recognition, Identification and Verification
- ConvNet Layers
- Implementation of VGG16
- Data augmentation



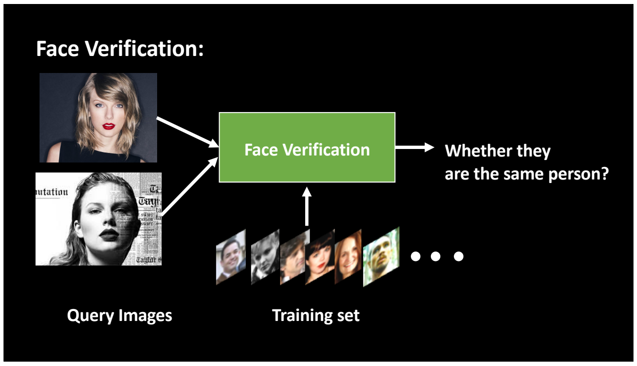
FaceID



Face Identification:

Query Image → Face Identification → Face ID: Taylor Swift

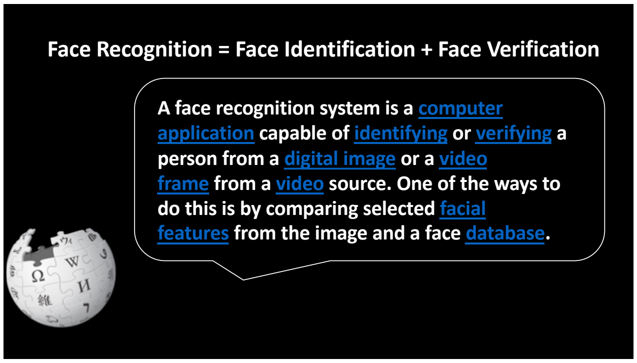
Training set • • •



Face Verification:

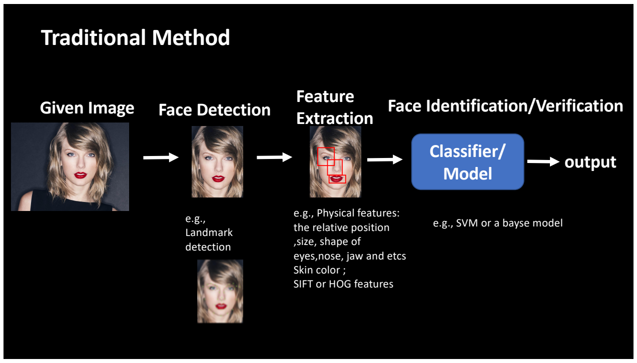
Query Images → Face Verification → Whether they are the same person?

Training set • • •



Face Recognition = Face Identification + Face Verification

A face recognition system is a computer application capable of identifying or verifying a person from a digital image or a video frame from a video source. One of the ways to do this is by comparing selected facial features from the image and a face database.



Traditional Method

Given Image

Face Detection

Feature Extraction

Face Identification/Verification

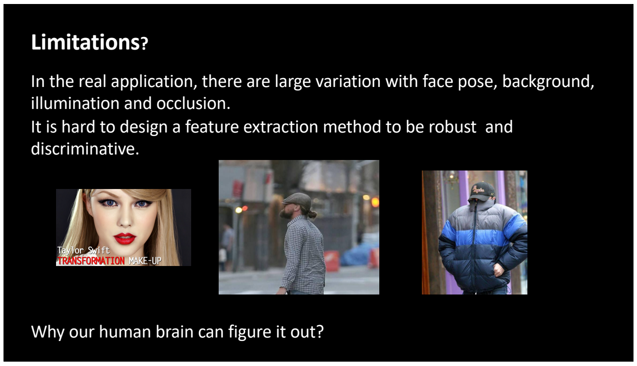
Classifier/ Model

output

e.g., Landmark detection

e.g., Physical features: the relative position ,size, shape of eyes,nose, jaw and etcs Skin color ; SIFT or HOG features

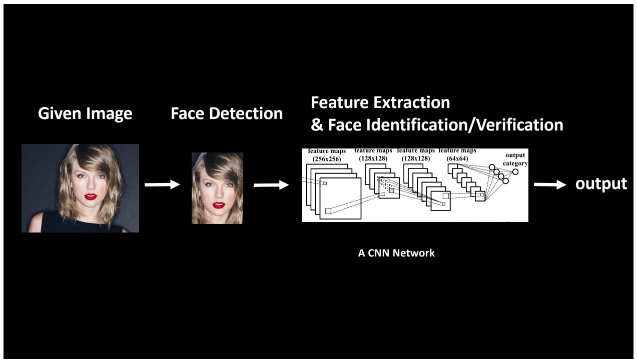
e.g., SVM or a bayse model



Limitations?

In the real application, there are large variation with face pose, background, illumination and occlusion.
It is hard to design a feature extraction method to be robust and discriminative.

Why our human brain can figure it out?



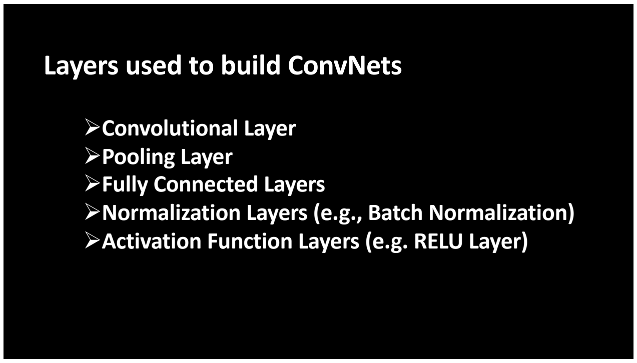
Given Image

Face Detection

Feature Extraction
& Face Identification/Verification

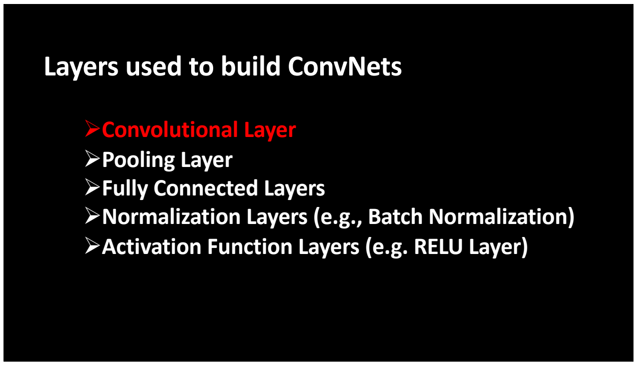
output

A CNN Network



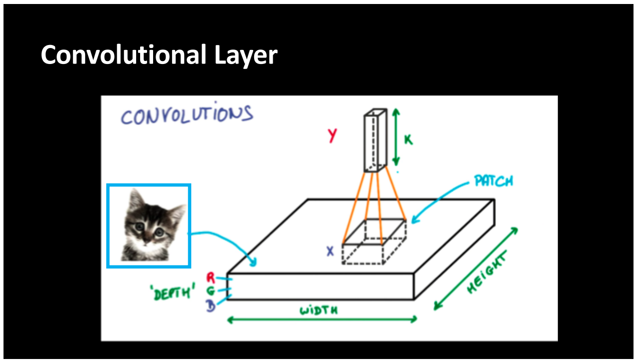
Layers used to build ConvNets

- ➢Convolutional Layer
- ➢Pooling Layer
- ➢Fully Connected Layers
- ➢Normalization Layers (e.g., Batch Normalization)
- ➢Activation Function Layers (e.g. RELU Layer)

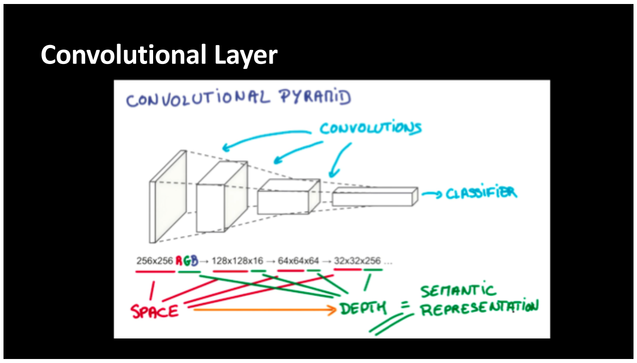


Layers used to build ConvNets

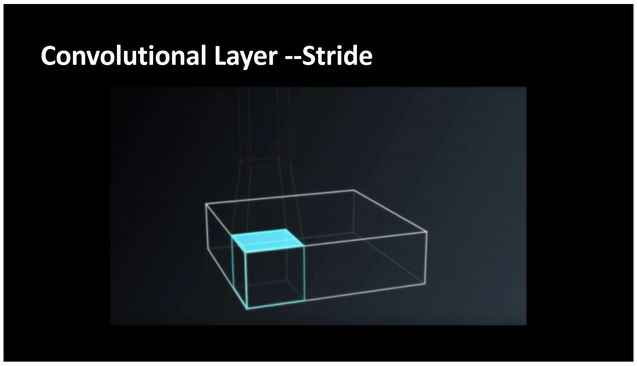
- ➢Convolutional Layer
- ➢Pooling Layer
- ➢Fully Connected Layers
- ➢Normalization Layers (e.g., Batch Normalization)
- ➢Activation Function Layers (e.g. RELU Layer)



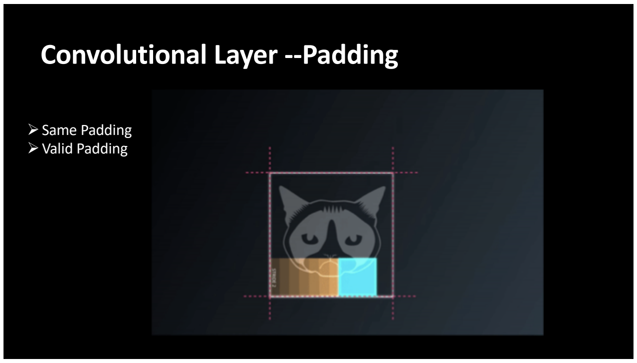
Convolutional Layer



Convolutional Layer

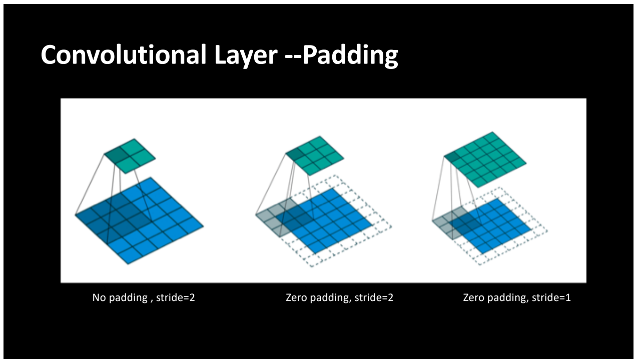


Convolutional Layer --Stride



Convolutional Layer --Padding

- ➢ Same Padding
- ➢ Valid Padding

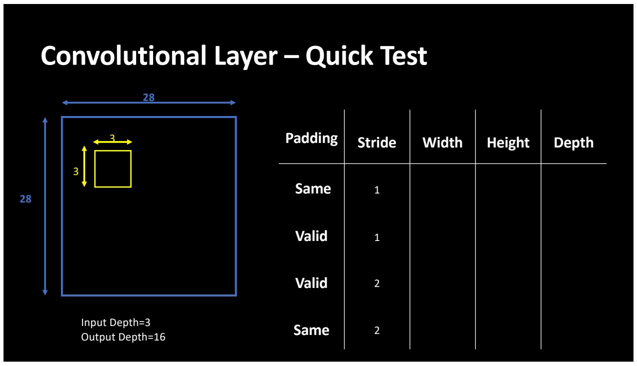


Convolutional Layer --Padding

No padding , stride=2

Zero padding, stride=2

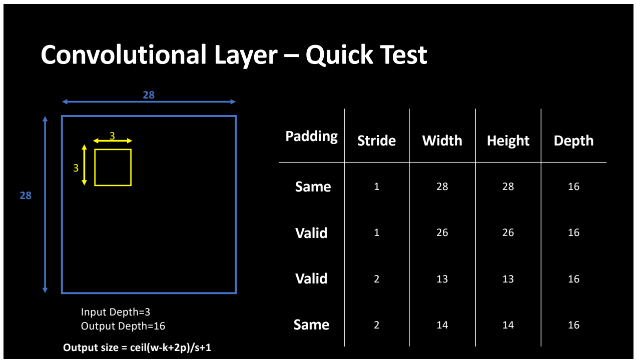
Zero padding, stride=1



Convolutional Layer – Quick Test

Input Depth=3
Output Depth=16

Padding	Stride	Width	Height	Depth
Same	1			
Valid	1			
Valid	2			
Same	2			

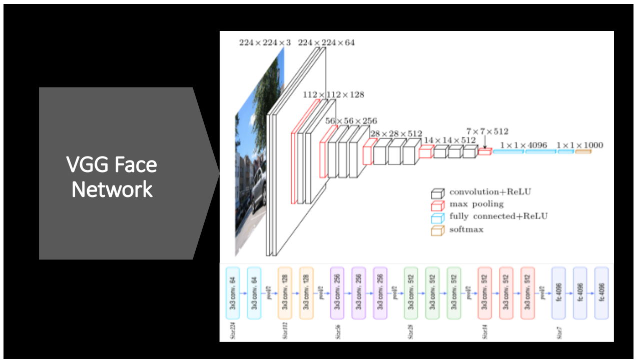


Convolutional Layer – Quick Test

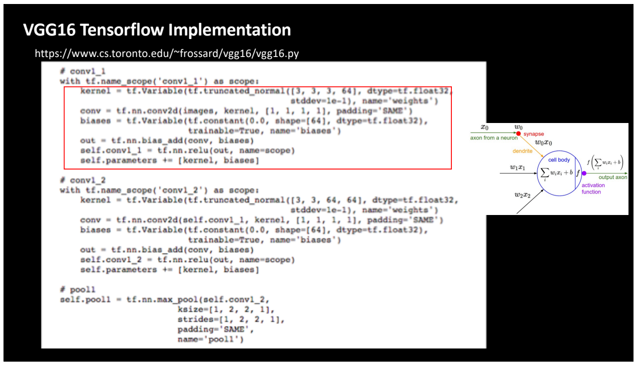
Input Depth=3
Output Depth=16

Output size = ceil(w-k+2p)/s+1

Padding	Stride	Width	Height	Depth
Same	1	28	28	16
Valid	1	26	26	16
Valid	2	13	13	16
Same	2	14	14	16



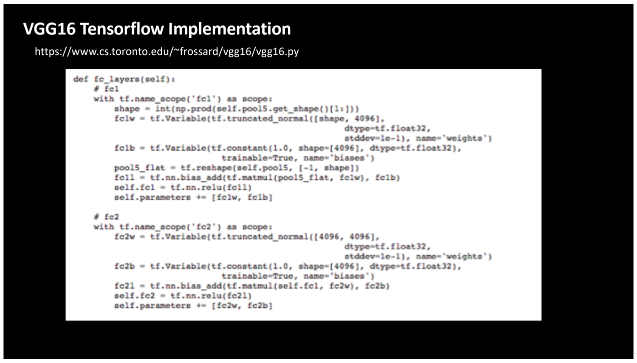
VGG Face Network



VGG16 Tensorflow Implementation

https://www.cs.toronto.edu/~frossard/vgg16/vgg16.py

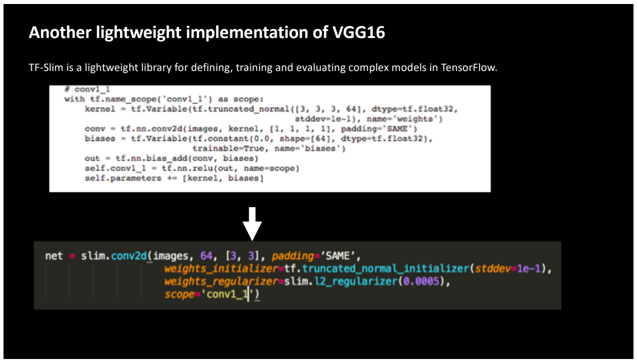
```
# conv1_1
with tf.name_scope('conv1_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 3, 64], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv1_1 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]

# conv1_2
with tf.name_scope('conv1_2') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 64], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(self.conv1_1, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv1_2 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]

# pool1
self.pool1 = tf.nn.max_pool(self.conv1_2,
                       ksize=[1, 2, 2, 1],
                       strides=[1, 2, 2, 1],
                       padding='SAME',
                       name='pool1')
```



VGG16 Tensorflow Implementation

https://www.cs.toronto.edu/~frossard/vgg16/vgg16.py

```
def fc_layers(self):
    # fc1
    with tf.name_scope('fc1') as scope:
        shape = int(np.prod(self.pool5.get_shape()[1:]))
        fc1w = tf.Variable(tf.truncated_normal([shape, 4096],
                                                     dtype=tf.float32,
                                                     stddev=1e-1), name='weights')
        fc1b = tf.Variable(tf.constant(1.0, shape=[4096], dtype=tf.float32),
                           trainable=True, name='biases')
        pool5_flat = tf.reshape(self.pool5, [-1, shape])
        fc1l = tf.nn.bias_add(tf.matmul(pool5_flat, fc1w), fc1b)
        self.fc1 = tf.nn.relu(fc1l)
        self.parameters += [fc1w, fc1b]

    # fc2
    with tf.name_scope('fc2') as scope:
        fc2w = tf.Variable(tf.truncated_normal([4096, 4096],
                                                     dtype=tf.float32,
                                                     stddev=1e-1), name='weights')
        fc2b = tf.Variable(tf.constant(1.0, shape=[4096], dtype=tf.float32),
                           trainable=True, name='biases')
        fc2l = tf.nn.bias_add(tf.matmul(self.fc1, fc2w), fc2b)
        self.fc2 = tf.nn.relu(fc2l)
        self.parameters += [fc2w, fc2b]
```

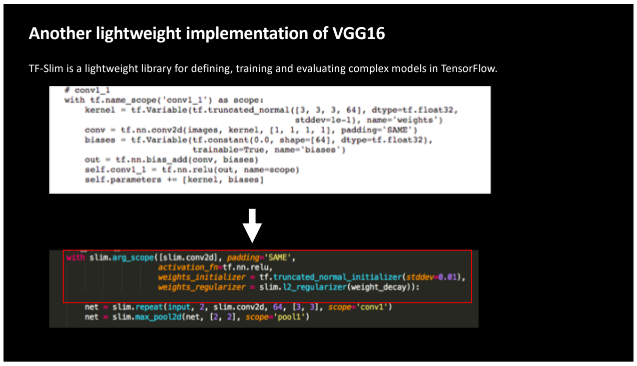


Another lightweight implementation of VGG16

TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow.

```
# conv1_1
with tf.name_scope('conv1_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 3, 64], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv1_1 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]
```

```
net = slim.conv2d(images, 64, [3, 3], padding='SAME',
                  weights_initializer=tf.truncated_normal_initializer(stddev=1e-1),
                  weights_regularizer=slim.l2_regularizer(0.0005),
                  scope='conv1_1')
```

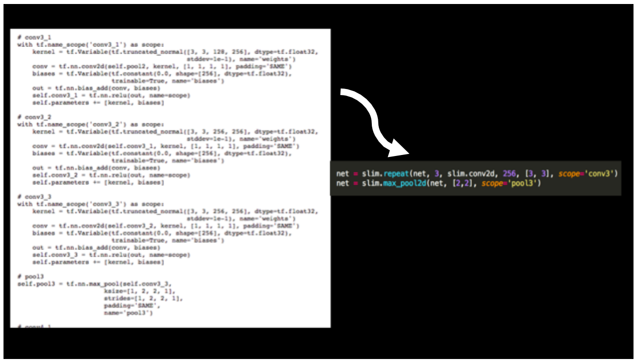


Another lightweight implementation of VGG16

TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow.

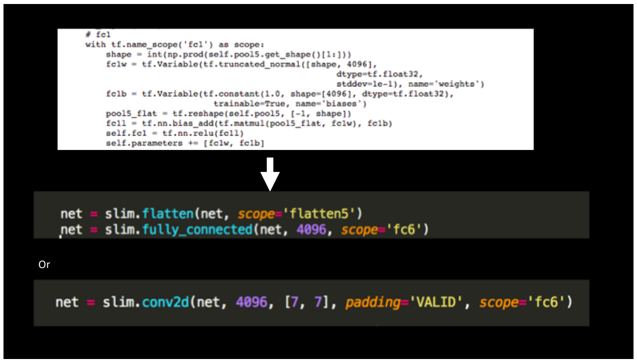
```
# conv1_1
with tf.name_scope('conv1_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 3, 64], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[64], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv1_1 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]
```

```
with slim.arg_scope([slim.conv2d], padding='SAME',
                    activation_fn=tf.nn.relu,
                    weights_initializer = tf.truncated_normal_initializer(stddev=0.01),
                    weights_regularizer = slim.l2_regularizer(weight_decay)):

    net = slim.repeat(input, 2, slim.conv2d, 64, [3, 3], scope='conv1')
    net = slim.max_pool2d(net, [2, 2], scope='pool1')
```



```
# conv3_1
with tf.name_scope('conv3_1') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 128, 256], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(self.pool2, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[256], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv3_1 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]

# conv3_2
with tf.name_scope('conv3_2') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 256, 256], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(self.conv3_1, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[256], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv3_2 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]

# conv3_3
with tf.name_scope('conv3_3') as scope:
    kernel = tf.Variable(tf.truncated_normal([3, 3, 256, 256], dtype=tf.float32,
                                             stddev=1e-1), name='weights')
    conv = tf.nn.conv2d(self.conv3_2, kernel, [1, 1, 1, 1], padding='SAME')
    biases = tf.Variable(tf.constant(0.0, shape=[256], dtype=tf.float32),
                         trainable=True, name='biases')
    out = tf.nn.bias_add(conv, biases)
    self.conv3_3 = tf.nn.relu(out, name=scope)
    self.parameters += [kernel, biases]

# pool3
self.pool3 = tf.nn.max_pool(self.conv3_3,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding='SAME',
                            name='pool3')

# conv4_1
```

```
net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3], scope='conv3')
net = slim.max_pool2d(net, [2,2], scope='pool3')
```

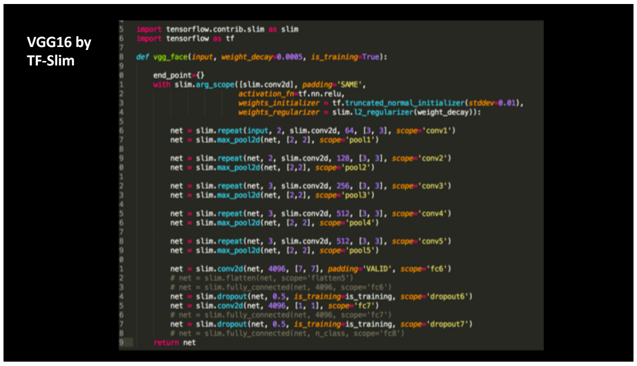


```
    # fc1
    with tf.name_scope('fc1') as scope:
        shape = int(np.prod(self.pool5.get_shape()[1:]))
        fc1w = tf.Variable(tf.truncated_normal([shape, 4096],
                                                     dtype=tf.float32,
                                                     stddev=1e-1), name='weights')
        fc1b = tf.Variable(tf.constant(1.0, shape=[4096], dtype=tf.float32),
                                 trainable=True, name='biases')
        pool5_flat = tf.reshape(self.pool5, [-1, shape])
        fc1l = tf.nn.bias_add(tf.matmul(pool5_flat, fc1w), fc1b)
        self.fc1 = tf.nn.relu(fc1l)
        self.parameters += [fc1w, fc1b]
```

```
net = slim.flatten(net, scope='flatten5')
net = slim.fully_connected(net, 4096, scope='fc6')
```

Or

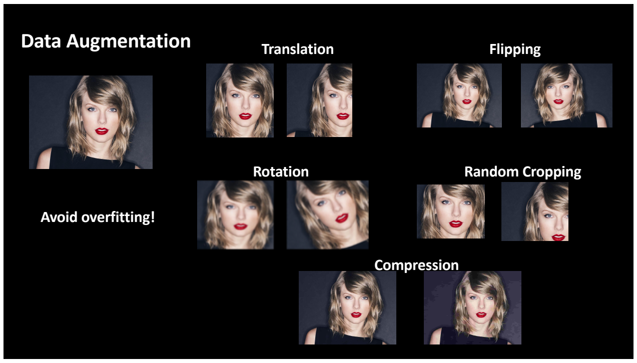
```
net = slim.conv2d(net, 4096, [7, 7], padding='VALID', scope='fc6')
```



VGG16 by TF-Slim

```
import tensorflow.contrib.slim as slim
import tensorflow as tf

def vgg_face(input, weight_decay=0.0005, is_training=True):

    end_point={}
    with slim.arg_scope([slim.conv2d], padding='SAME',
                        activation_fn=tf.nn.relu,
                        weights_initializer = tf.truncated_normal_initializer(stddev=0.01),
                        weights_regularizer = slim.l2_regularizer(weight_decay)):

        net = slim.repeat(input, 2, slim.conv2d, 64, [3, 3], scope='conv1')
        net = slim.max_pool2d(net, [2, 2], scope='pool1')

        net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], scope='conv2')
        net = slim.max_pool2d(net, [2,2], scope='pool2')

        net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3], scope='conv3')
        net = slim.max_pool2d(net, [2,2], scope='pool3')

        net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], scope='conv4')
        net = slim.max_pool2d(net, [2, 2], scope='pool4')

        net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], scope='conv5')
        net = slim.max_pool2d(net, [2, 2], scope='pool5')

        net = slim.conv2d(net, 4096, [7, 7], padding='VALID', scope='fc6')
        # net = slim.flatten(net, scope='flatten5')
        # net = slim.fully_connected(net, 4096, scope='fc6')
        net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout6')
        net = slim.conv2d(net, 4096, [1, 1], scope='fc7')
        # net = slim.fully_connected(net, 4096, scope='fc7')
        net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout7')
        # net = slim.fully_connected(net, n_class, scope='fc8')
    return net
```

Data Augmentation



Avoid overfitting!

Thanks!