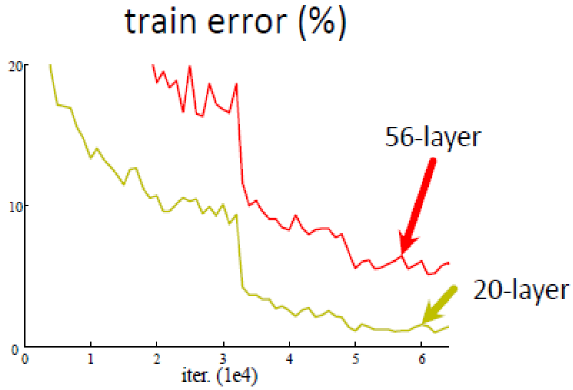
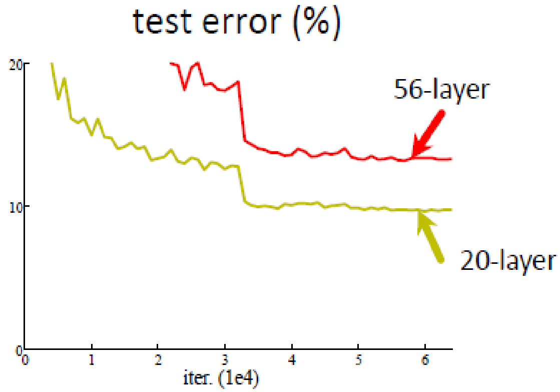
RESNET:
BATCH NORMALIZATION AND SKIP CONNECTION

Problems with Deeper network

- Vanishing/Exploding gradient problem

- Degradation problem
  - Overly deep plain nets have higher training error





Solutions for Deeper Networks

- ReLU

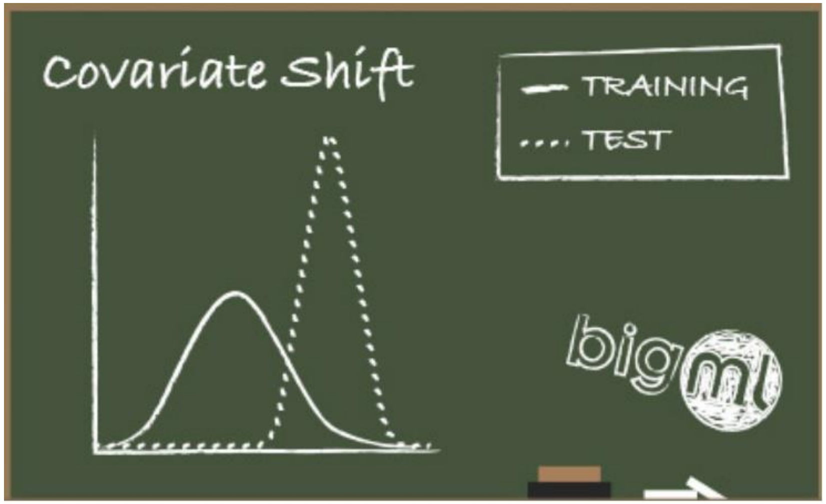
- Regularization methods
  - Dropout
  - Batch Normalization

- Skip connections/Residual Learning

BATCH NORMALIZATION

Covariate Shift

- Covariate
  - Predictor variable ~ Independent variable ~ Feature
- Covariate shift
  - $P_S(X) \neq P_T(X)$
  - The feature distribution in the source domain (e.g., training set) is different from that of target domain (e.g., test set).



Internal Covariate Shift

- Change of the input distribution for each sub-network during training is called internal covariate shift problem

Batch Normalization Algorithm

For clarity, $x \equiv x^{(k)}$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

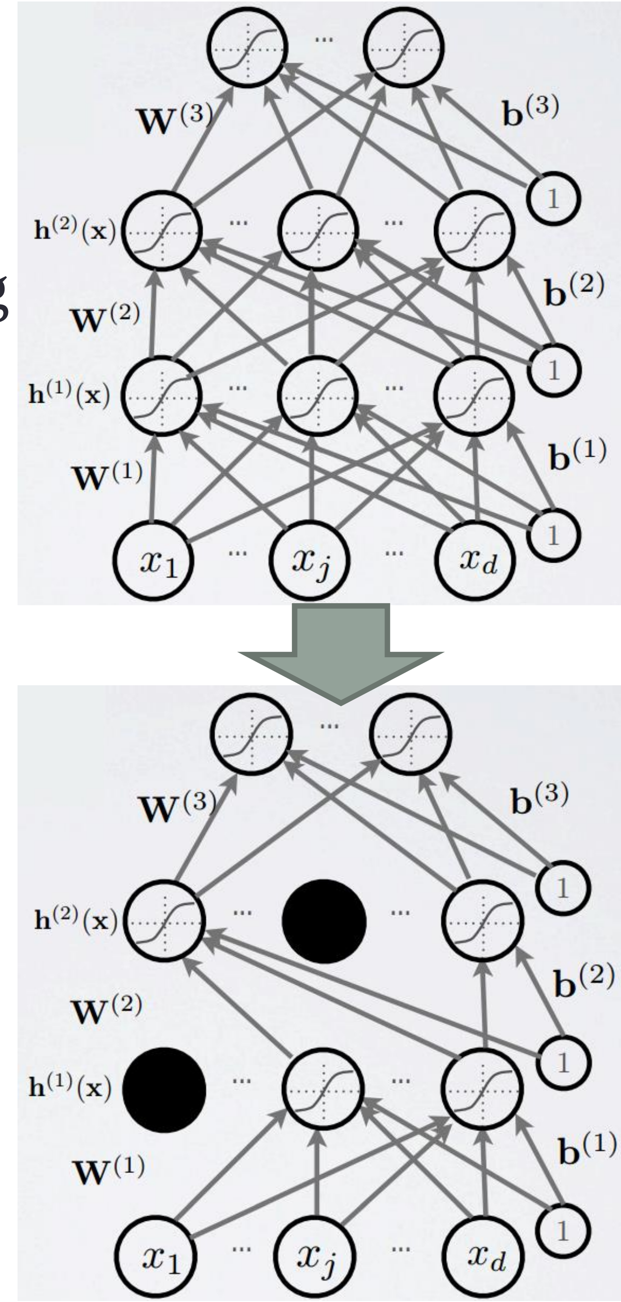
TensorFlow code

- Example Code
- Example Code (LeNet)

DROPOUT

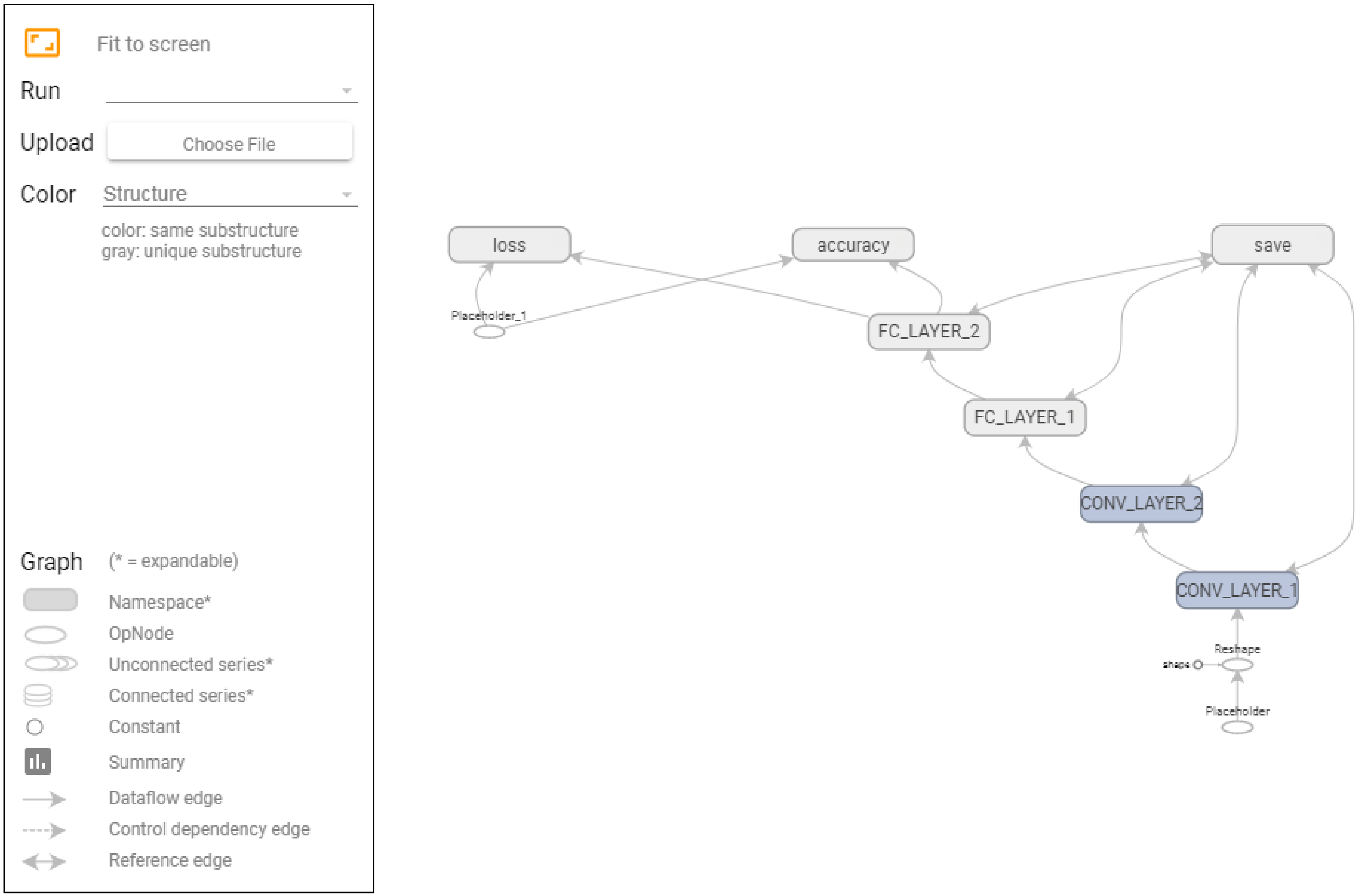
DROP-OUT

- To cripple neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful



LeNet

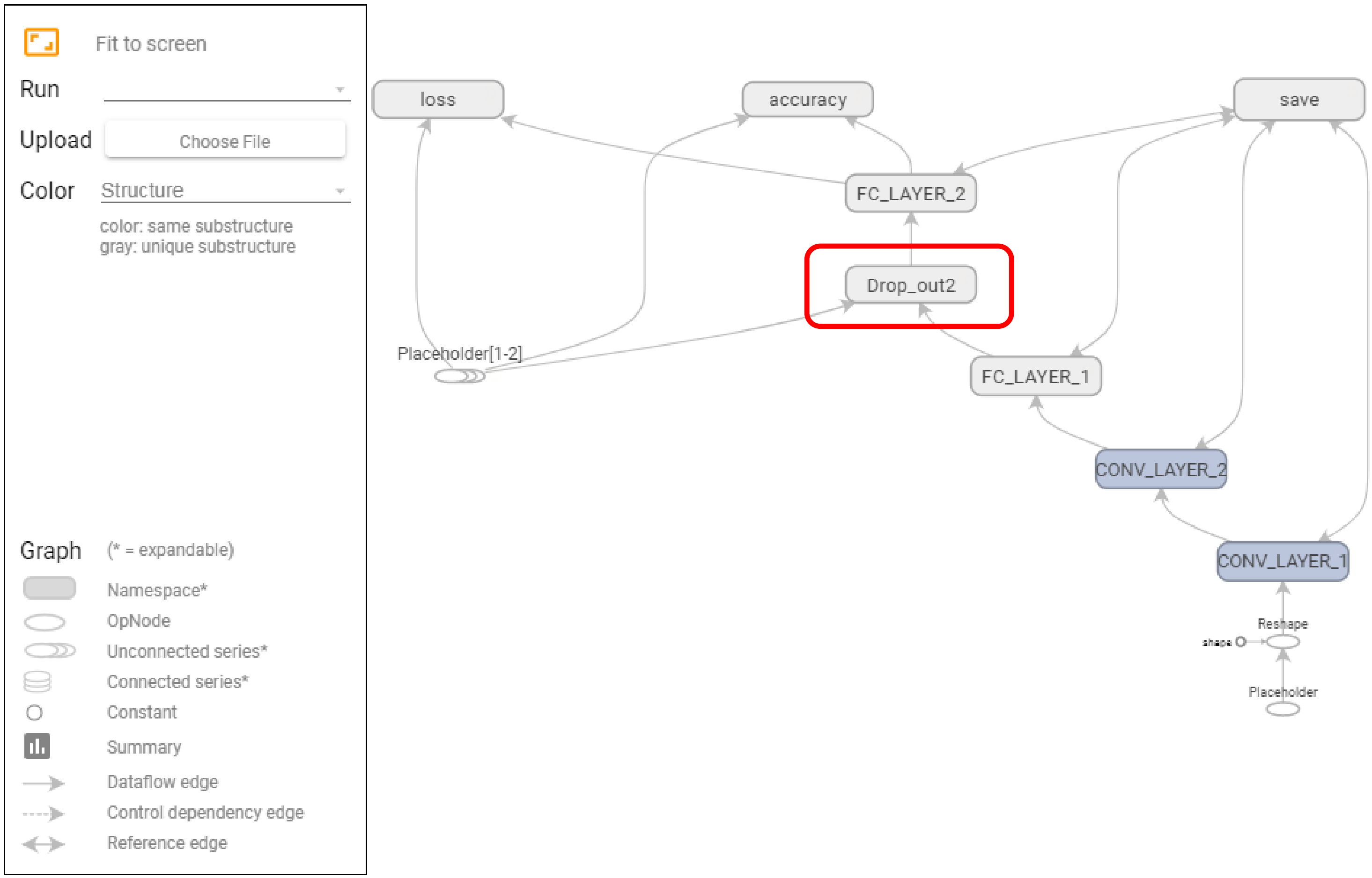
```
tf.reset_default_graph()

x = tf.placeholder( tf.float32, [None, 784] )
y = tf.placeholder( tf.float32, [None, 10])

x_image = tf.reshape( x, [-1,28,28,1] )

with tf.name_scope('CONV_LAYER_1'):
    W_conv1 = weight_variable( [5,5,1,32], name='Wconv1' )
    b_conv1 = bias_variable( [32], 1.0, name='bconv1' )
    h_conv1 = tf.nn.relu( conv2d(x_image,W_conv1) + b_conv1 )
    h_pool1 = max_pool_2x2( h_conv1 )

with tf.name_scope('CONV_LAYER_2'):
    W_conv2 = weight_variable( [5,5,32,64], name='Wconv2' )
    b_conv2 = bias_variable( [64], 1.0, name='bconv2' )
    h_conv2 = tf.nn.relu( conv2d(h_pool1,W_conv2) + b_conv2 )
    h_pool2 = max_pool_2x2( h_conv2 )

with tf.name_scope('FC_LAYER_1'):
    W_fc1 = weight_variable([7*7*64, 1024], name='Wfc1')
    b_fc1 = bias_variable( [1024], 1.0, name='bfc1' )

    h_pool2_flat = tf.reshape(h_pool2, [-1,7*7*64])
    h_fc1 = tf.nn.relu( tf.matmul(h_pool2_flat, W_fc1) + b_fc1 )

with tf.name_scope('FC_LAYER_2'):
    W_fc2 = weight_variable([1024,10], name='Wfc2')
    b_fc2 = bias_variable( [10], 0.0, name='bfc2' )
    pred = tf.matmul(h_fc1, W_fc2) + b_fc2

with tf.name_scope('loss'):
    cross_entropy = tf.reduce_mean( tf.nn.sigmoid_cross_entropy_with_logits(logits=pred,labels=y))

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(pred,1),tf.argmax(y,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,"float"))

saver = tf.train.Saver()
```


Fit to screen
Run
Upload
Choose File
Color
Structure
color: same substructure
gray: unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Unconnected series*
Connected series*
Constant
Summary
Dataflow edge
Control dependency edge
Reference edge
loss
accuracy
save
Placeholder_1
FC_LAYER_2
FC_LAYER_1
CONV_LAYER_2
CONV_LAYER_1
Reshape
shape
Placeholder

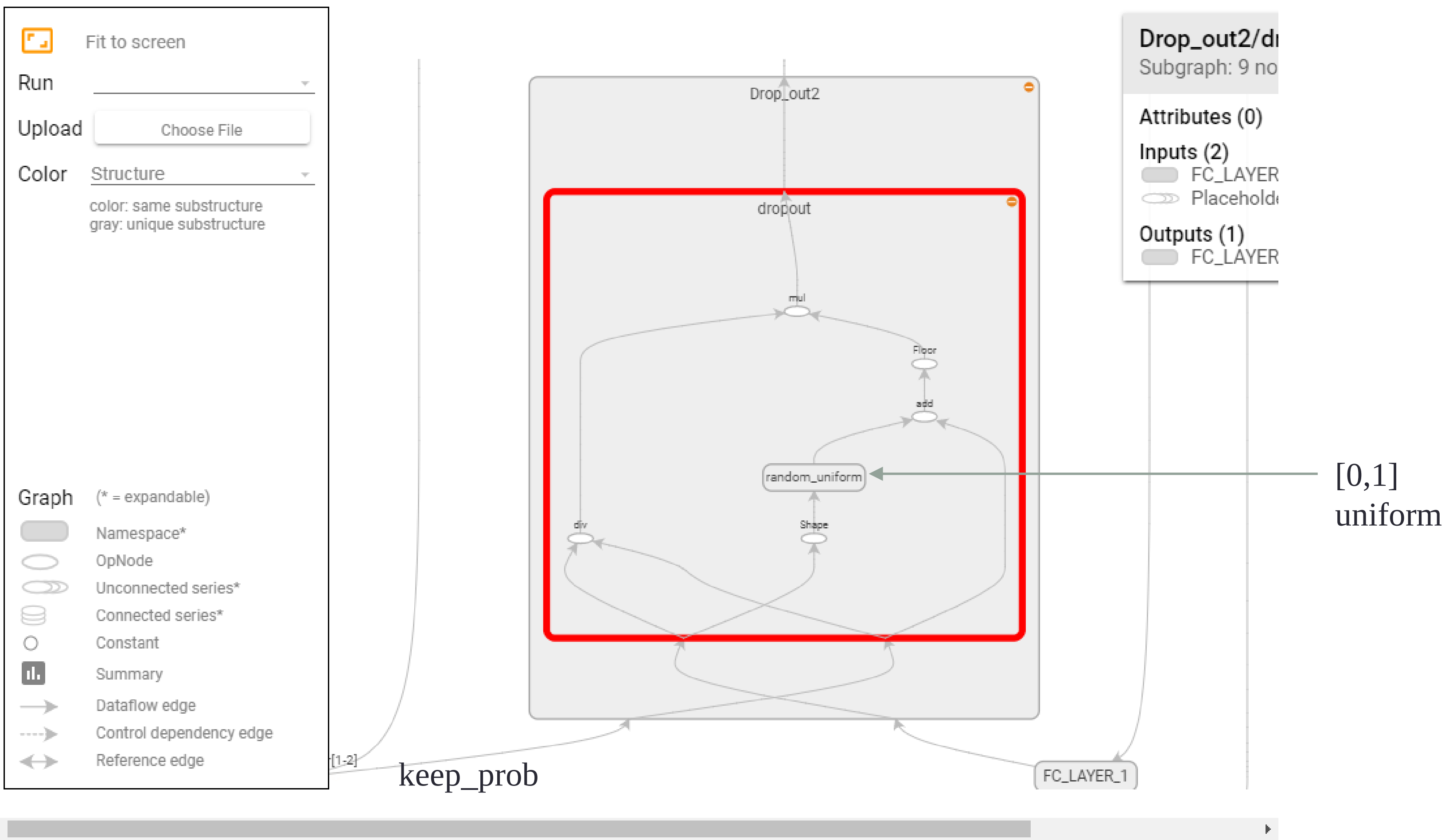
LeNet with dropout

```
tf.reset_default_graph()

x = tf.placeholder( tf.float32, [None, 784] )
y = tf.placeholder( tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)      ##########################

x_image = tf.reshape( x, [-1,28,28,1] )

with tf.name_scope('CONV_LAYER_1'):
    W_conv1 = weight_variable( [5,5,1,32], name='Wconv1' )
    b_conv1 = bias_variable( [32], 1.0, name='bconv1' )
    h_conv1 = tf.nn.relu( conv2d(x_image,W_conv1) + b_conv1 )
    h_pool1 = max_pool_2x2( h_conv1 )

with tf.name_scope('CONV_LAYER_2'):
    W_conv2 = weight_variable( [5,5,32,64], name='Wconv2' )
    b_conv2 = bias_variable( [64], 1.0, name='bconv2' )
    h_conv2 = tf.nn.relu( conv2d(h_pool1,W_conv2) + b_conv2 )
    h_pool2 = max_pool_2x2( h_conv2 )

with tf.name_scope('FC_LAYER_1'):
    W_fc1 = weight_variable([7*7*64, 1024], name='Wfc1')
    b_fc1 = bias_variable( [1024], 1.0, name='bfc1' )

    h_pool2_flat = tf.reshape(h_pool2, [-1,7*7*64])
    h_fc1 = tf.nn.relu( tf.matmul(h_pool2_flat, W_fc1) + b_fc1 )

with tf.name_scope('Drop_out2'):
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)     ##########################

with tf.name_scope('FC_LAYER_2'):
    W_fc2 = weight_variable([1024,10], name='Wfc2')
    b_fc2 = bias_variable( [10], 0.0, name='bfc2' )
    pred = tf.matmul(h_fc1_drop, W_fc2) + b_fc2 ##########################

with tf.name_scope('loss'):
    cross_entropy = tf.reduce_mean( tf.nn.sigmoid_cross_entropy_with_logits(logits=pred,labels=y))

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(pred,1),tf.argmax(y,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,"float"))

saver = tf.train.Saver()
```

```
show_graph(tf.get_default_graph().as_graph_def())
```



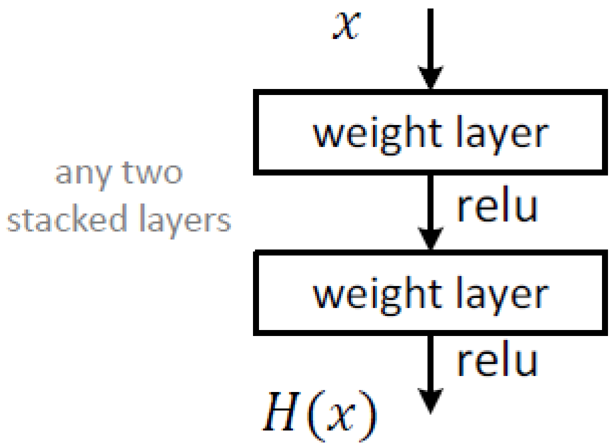
```
show_graph(tf.get_default_graph().as_graph_def())
```



SKIP CONNECTION
RESIDUAL LEARNING

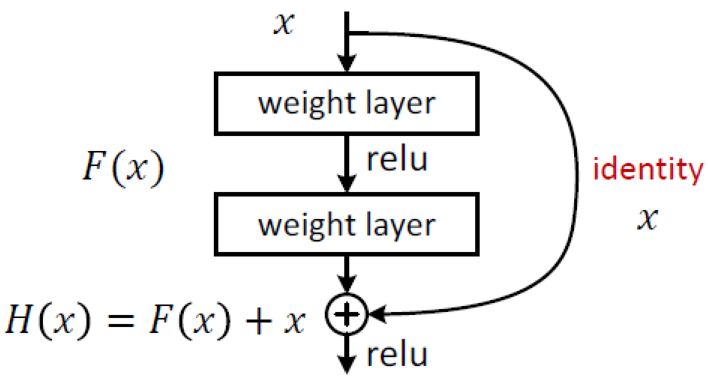
Plain block

- Difficult to make identity mapping because of multiple non-linear layers



Residual Block

- Identity mapping shortcut connections
  - If identity were optimal, easy to set weight as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations
  - Add neither extra parameter nor computational complexity



Very Deep Networks

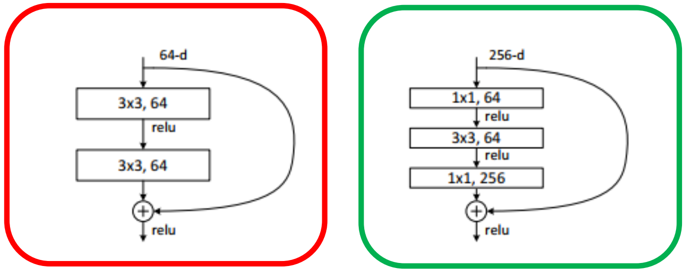
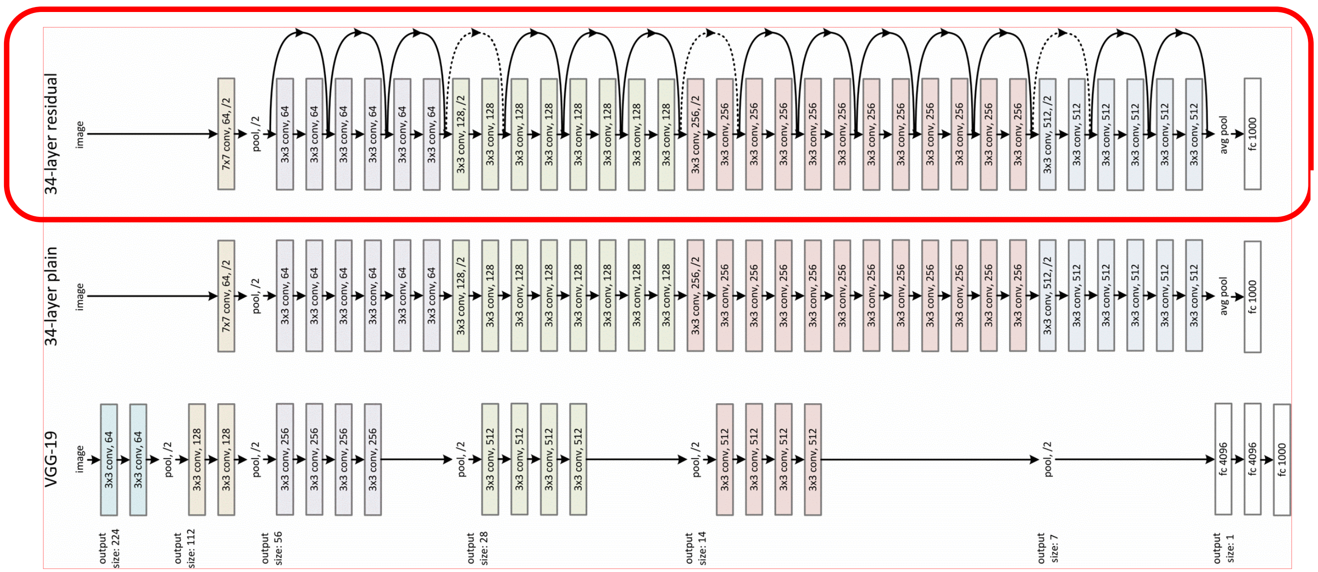
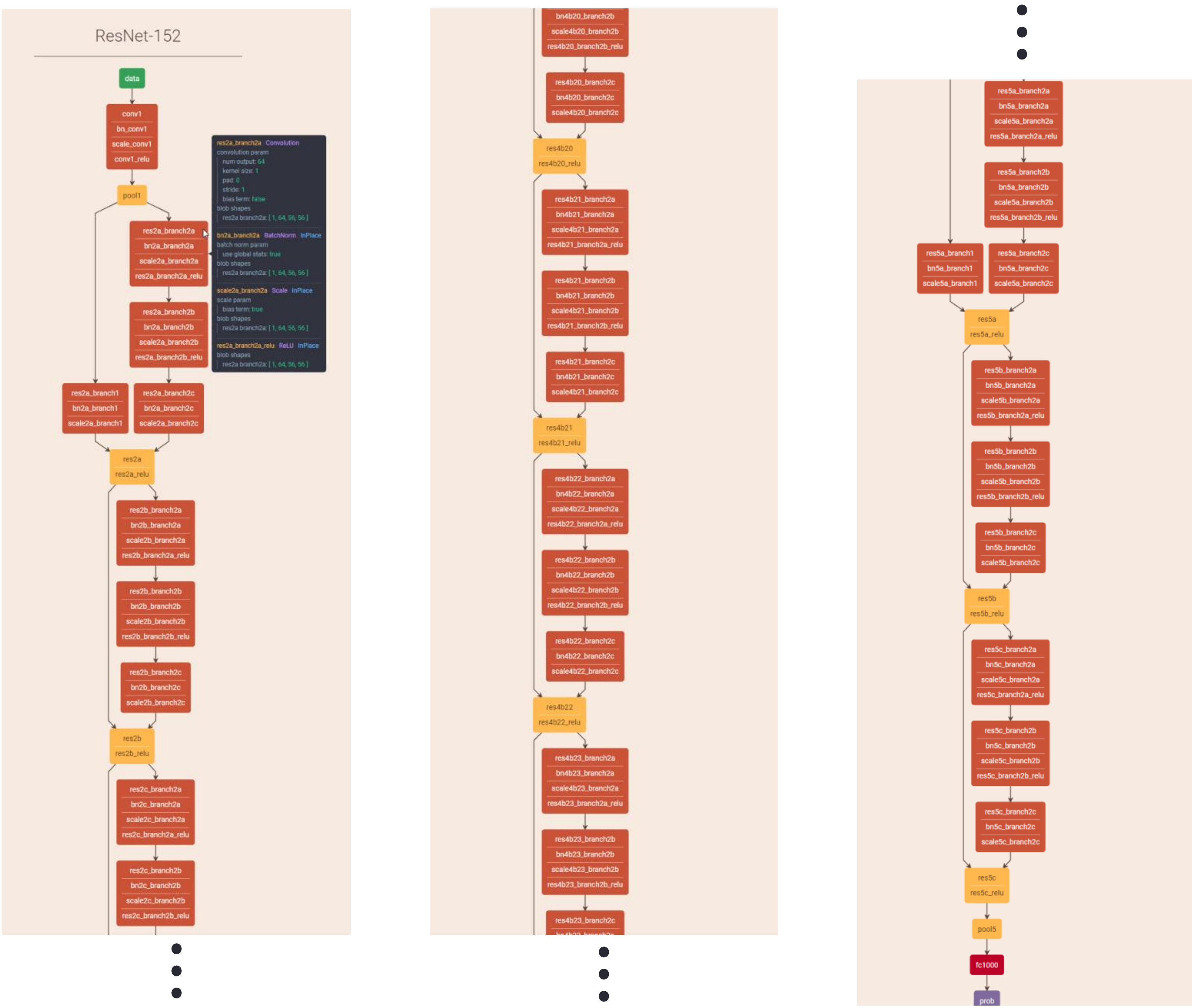


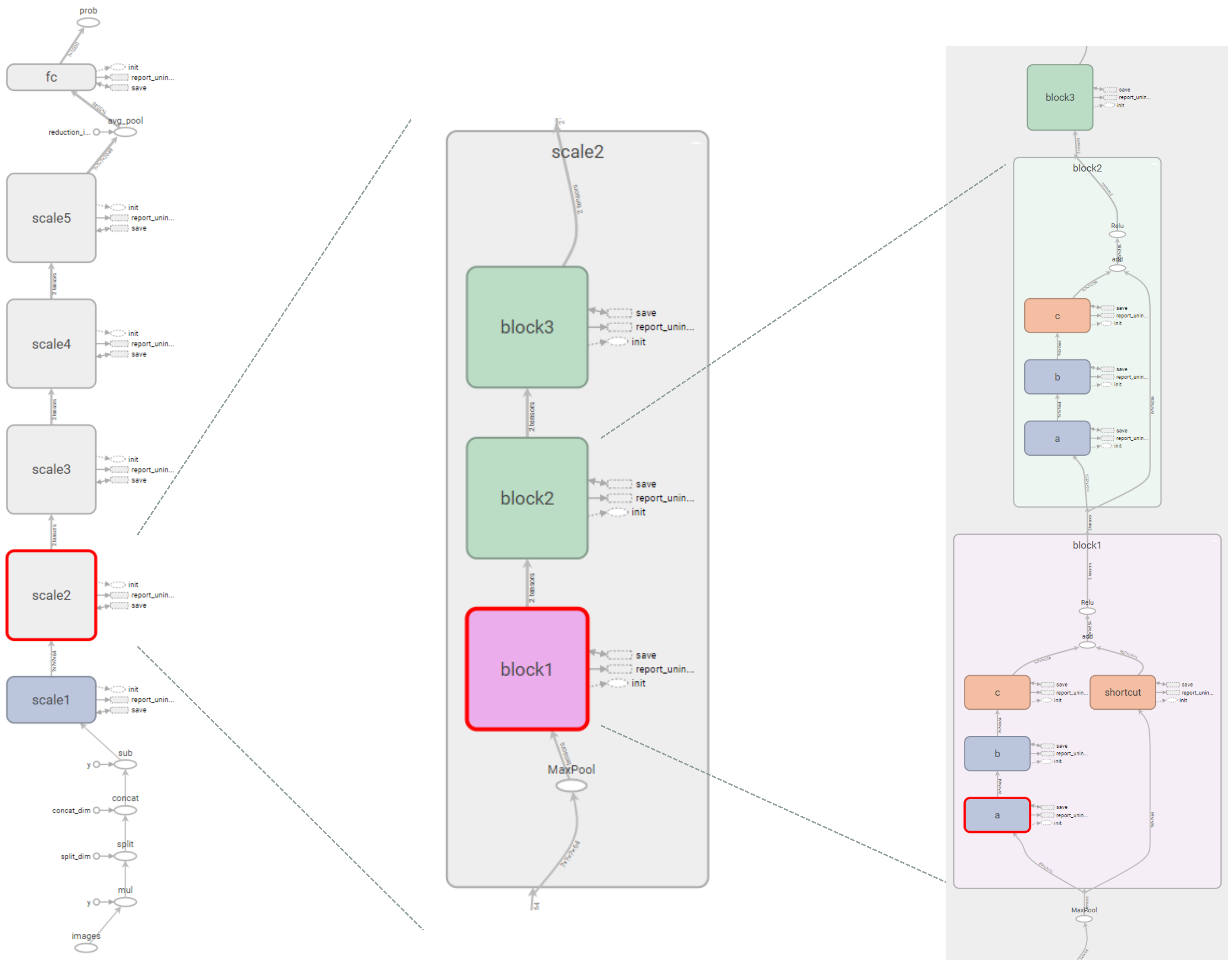
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

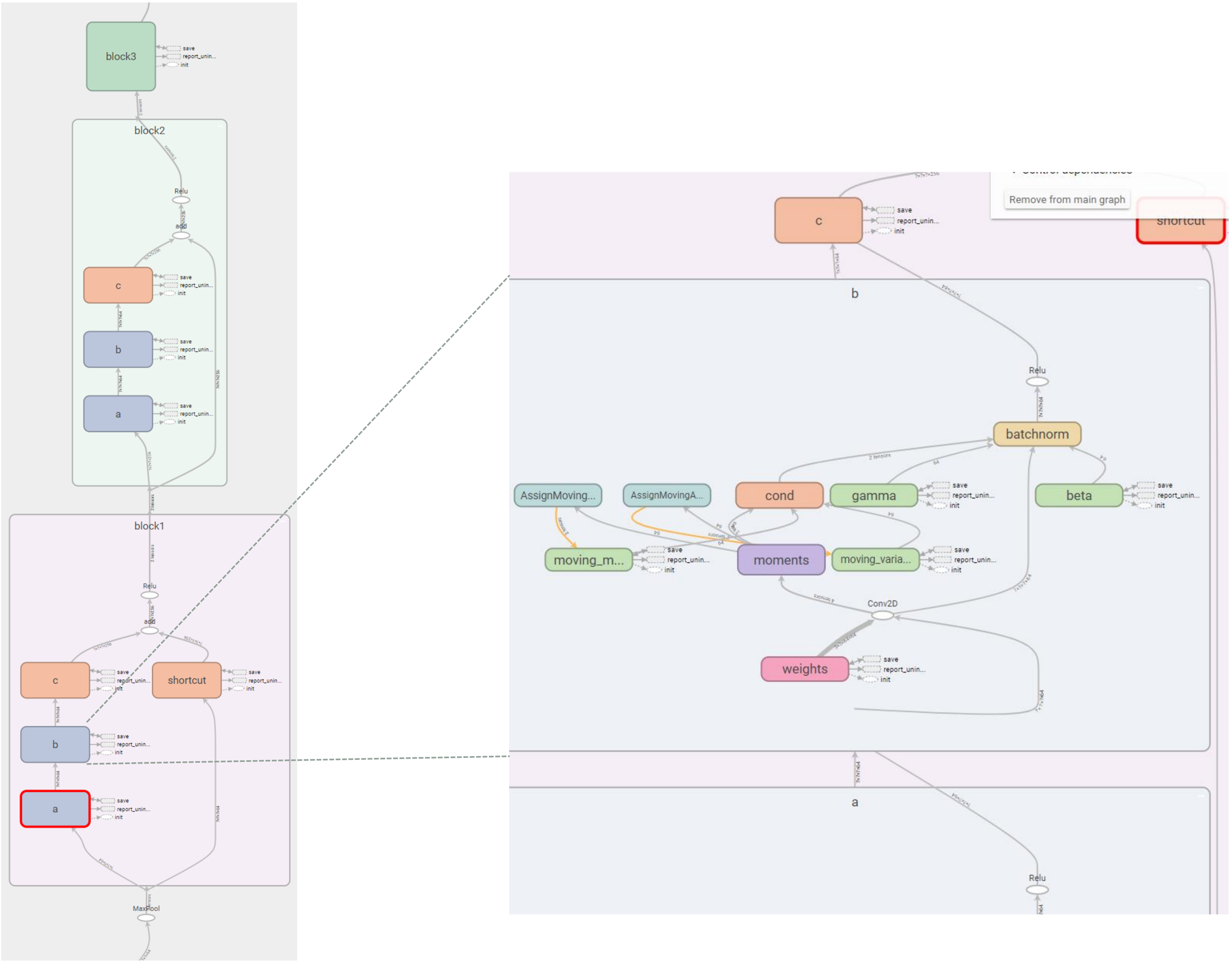
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

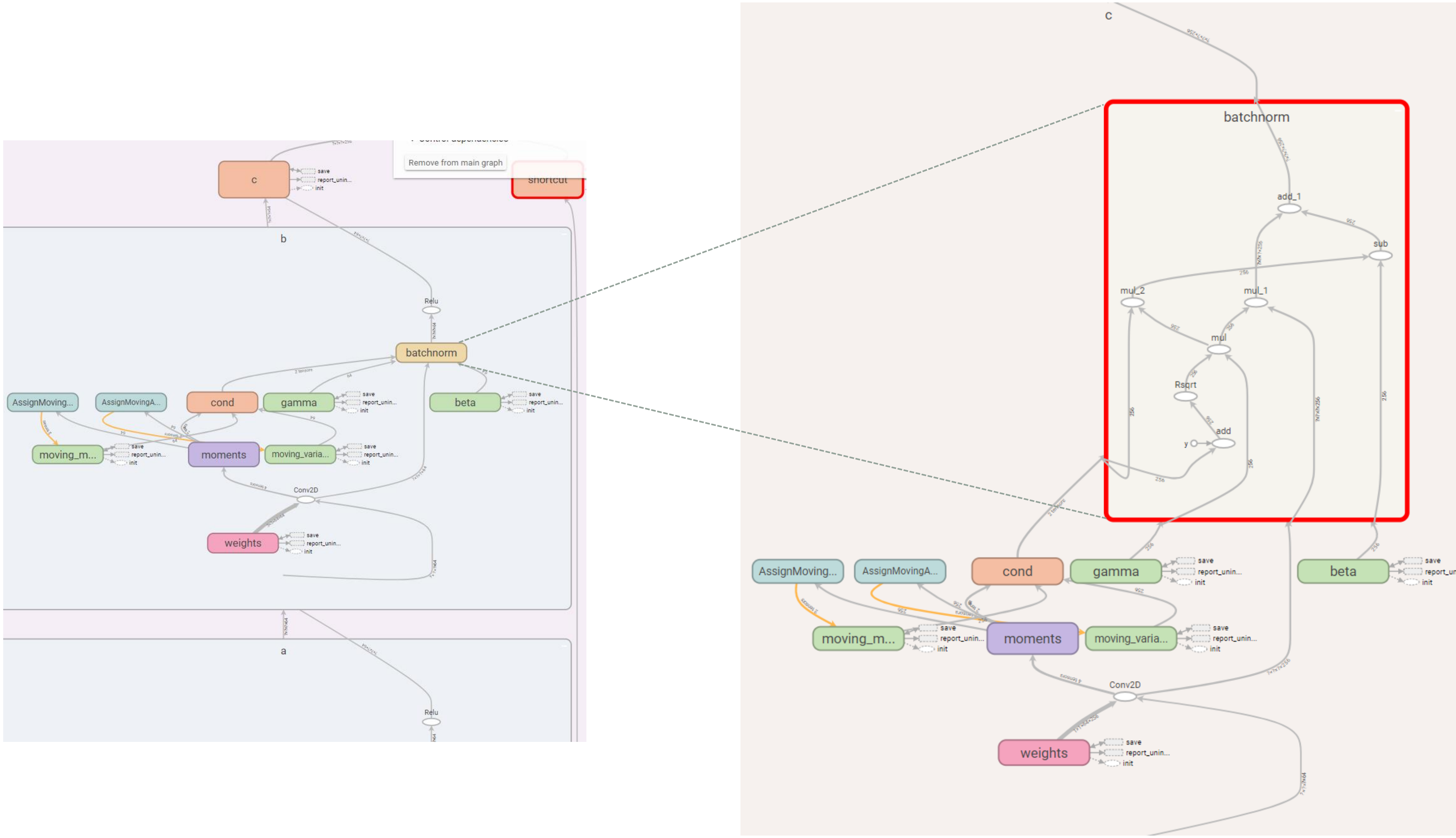


method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**


ResNet-152
data
conv1
bn_conv1
scale_conv1
conv1_relu
pool1
res2a_branch2a
bn2a_branch2a
scale2a_branch2a
res2a_branch2a_relu
res2a_branch2b
bn2a_branch2b
scale2a_branch2b
res2a_branch2b_relu
res2a_branch1
bn2a_branch1
scale2a_branch1
res2a_branch2c
bn2a_branch2c
scale2a_branch2c
res2a
res2a_relu
res2b_branch2a
bn2b_branch2a
scale2b_branch2a
res2b_branch2a_relu
res2b_branch2b
bn2b_branch2b
scale2b_branch2b
res2b_branch2b_relu
res2b_branch2c
bn2b_branch2c
scale2b_branch2c
res2b
res2b_relu
res2c_branch2a
bn2c_branch2a
scale2c_branch2a
res2c_branch2a_relu
res2c_branch2b
bn2c_branch2b
scale2c_branch2b
res2c_branch2b_relu
bn4b20_branch2b
scale4b20_branch2b
res4b20_branch2b_relu
res4b20_branch2c
bn4b20_branch2c
scale4b20_branch2c
res4b20
res4b20_relu
res4b21_branch2a
bn4b21_branch2a
scale4b21_branch2a
res4b21_branch2a_relu
res4b21_branch2b
bn4b21_branch2b
scale4b21_branch2b
res4b21_branch2b_relu
res4b21_branch2c
bn4b21_branch2c
scale4b21_branch2c
res4b21
res4b21_relu
res4b22_branch2a
bn4b22_branch2a
scale4b22_branch2a
res4b22_branch2a_relu
res4b22_branch2b
bn4b22_branch2b
scale4b22_branch2b
res4b22_branch2b_relu
res4b22_branch2c
bn4b22_branch2c
scale4b22_branch2c
res4b22
res4b22_relu
res4b23_branch2a
bn4b23_branch2a
scale4b23_branch2a
res4b23_branch2a_relu
res4b23_branch2b
bn4b23_branch2b
scale4b23_branch2b
res4b23_branch2b_relu
res4b23_branch2c
res5a_branch2a
bn5a_branch2a
scale5a_branch2a
res5a_branch2a_relu
res5a_branch2b
bn5a_branch2b
scale5a_branch2b
res5a_branch2b_relu
res5a_branch1
bn5a_branch1
scale5a_branch1
res5a_branch2c
bn5a_branch2c
scale5a_branch2c
res5a
res5a_relu
res5b_branch2a
bn5b_branch2a
scale5b_branch2a
res5b_branch2a_relu
res5b_branch2b
bn5b_branch2b
scale5b_branch2b
res5b_branch2b_relu
res5b_branch2c
bn5b_branch2c
scale5b_branch2c
res5b
res5b_relu
res5c_branch2a
bn5c_branch2a
scale5c_branch2a
res5c_branch2a_relu
res5c_branch2b
bn5c_branch2b
scale5c_branch2b
res5c_branch2b_relu
res5c_branch2c
bn5c_branch2c
scale5c_branch2c
res5c
res5c_relu
pool5
fc1000
prob


prob
fc
init
report_unin...
save
avg_pool
reduction_i...
scale5
scale4
scale3
scale2
scale1
sub
y
concat
concat_dim
split
split_dim
mul
images
scale2
block3
block2
block1
MaxPool
save
report_unin...
init
2 tensors
block3
block2
Relu
add
c
b
a
block1
shortcut
MaxPool


block3
save
report_unin...
init
block2
Relu
add
c
b
a
block1
Relu
add
c
shortcut
b
a
MaxPool
Remove from main graph
shortcut
c
b
Relu
batchnorm
AssignMoving...
AssignMovingA...
cond
gamma
beta
moving_m...
moments
moving_varia...
Conv2D
weights
a
Relu


c
shortcut
Remove from main graph
b
Relu
batchnorm
AssignMoving...
AssignMovingA...
cond
gamma
beta
moving_m...
moments
moving_varia...
Conv2D
weights
save
report_unin...
init
a
add_1
sub
mul_2
mul_1
mul
Rsqrt
add
y

CAFFE

- Deep networks are compositional models
- a collection of inter-connected layers that work on chunks of data.
- A network defines the entire model bottom-to-top from input data to loss

Example

- Neural Network
- LeNet example
- Non-image example