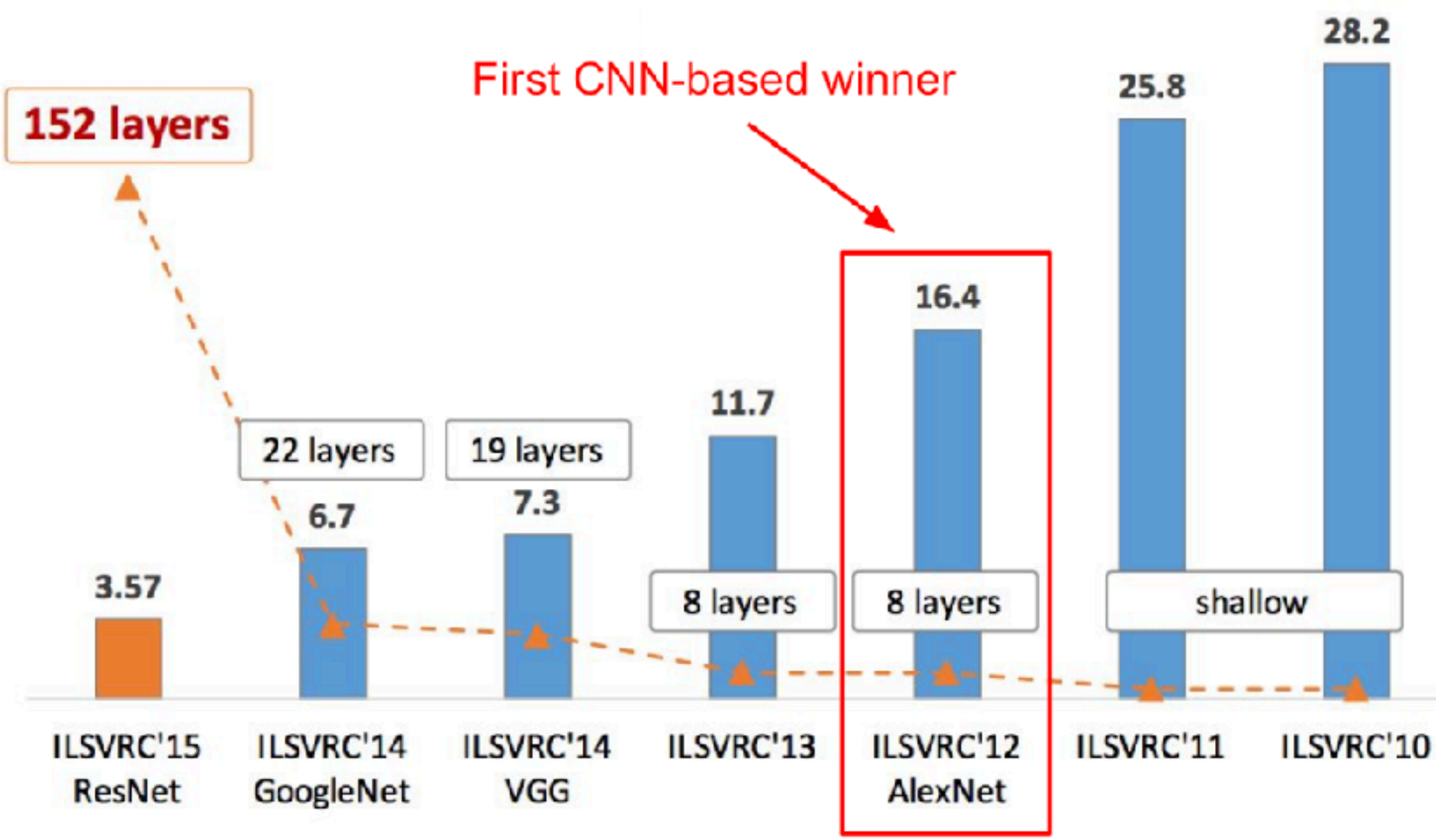
AlexNet

Cho Sung Man

Index

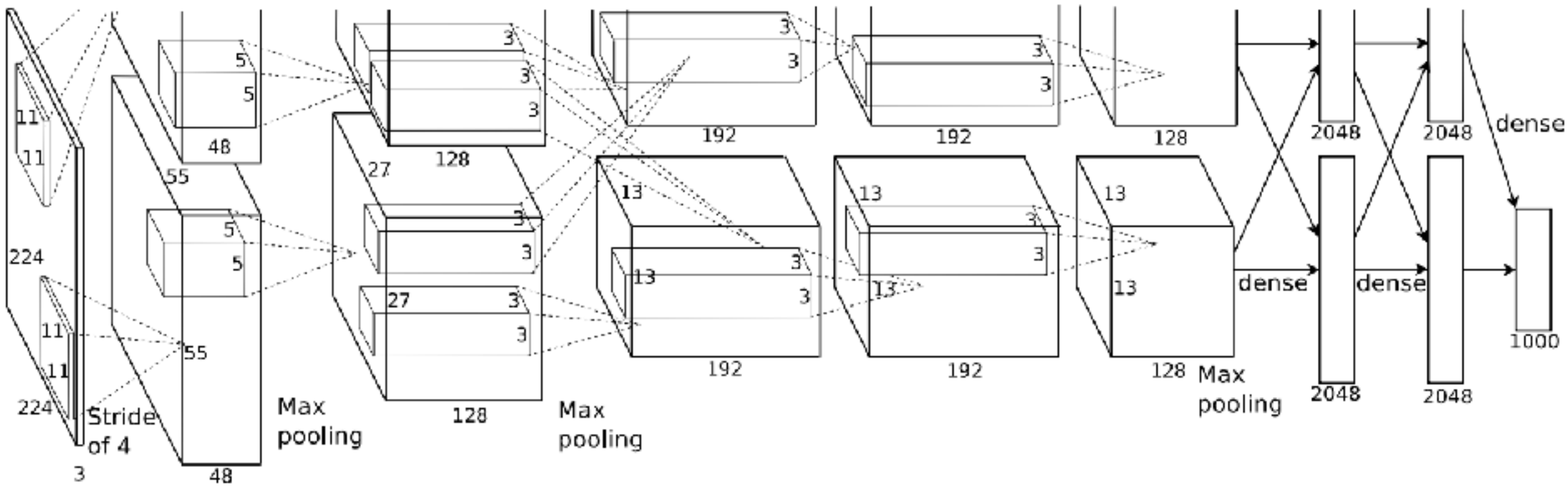
Introduction

Introduction



Architecture

Architecture



Architecture

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Input: 227x227x3 images

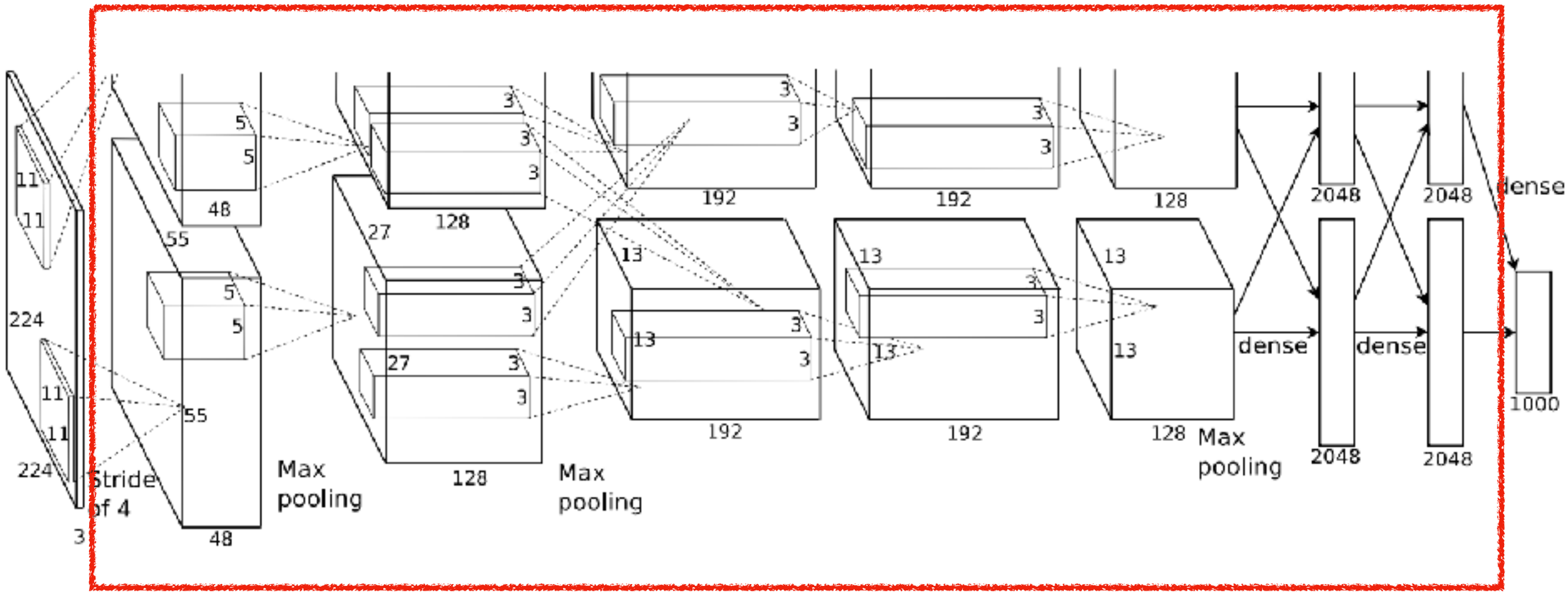
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

```python
slim = tf.contrib.slim
trunc_normal = lambda stddev: tf.truncated_normal_initializer(0.0, stdde


def alexnet_v2_arg_scope(weight_decay=0.0005):
  with slim.arg_scope([slim.conv2d, slim.fully_connected],
                      activation_fn=tf.nn.relu,
                      biases_initializer=tf.constant_initializer(0.1),
                      weights_regularizer=slim.l2_regularizer(weight_dec
    with slim.arg_scope([slim.conv2d], padding='SAME'):
      with slim.arg_scope([slim.max_pool2d], padding='VALID') as arg_sc:
        return arg_sc


def alexnet_v2(inputs,
               num_classes=1000,
               is_training=True,
               dropout_keep_prob=0.5,
               spatial_squeeze=True,
               scope='alexnet_v2',
               global_pool=False):
"""AlexNet version 2.
```

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

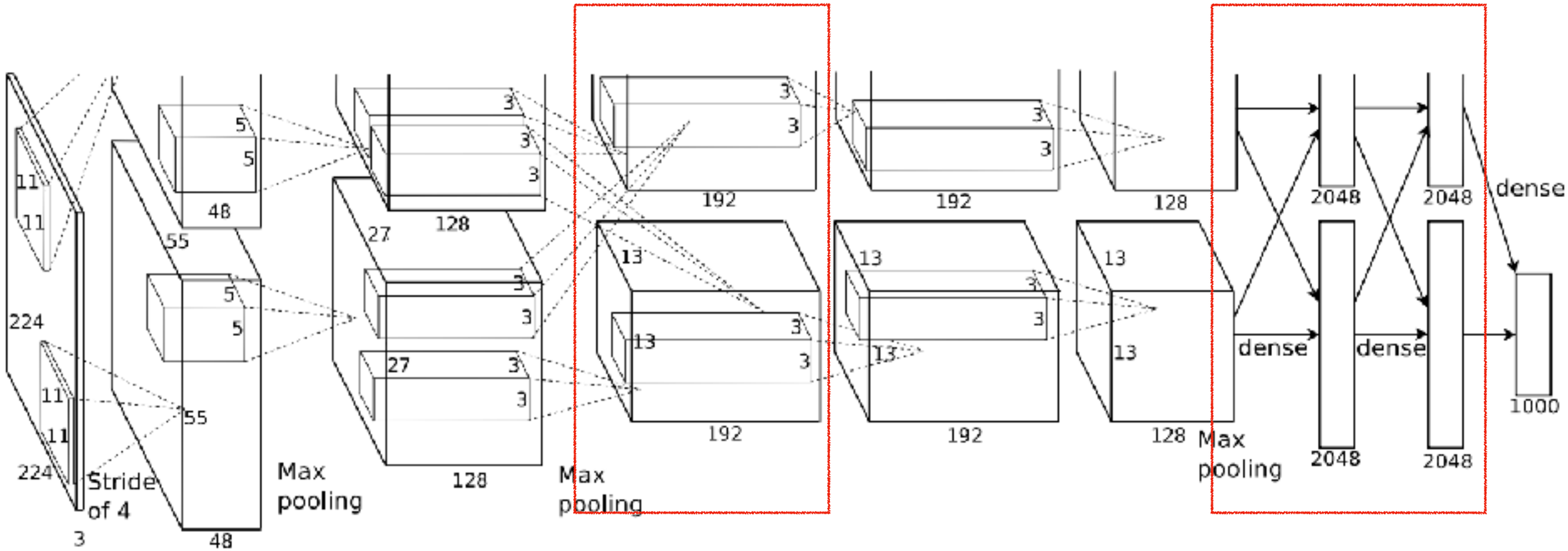
```python
with slim.arg_scope([slim.conv2d, slim.fully_connected, slim.max_pool2d],
                    outputs_collections=[end_points_collection]):
  net = slim.conv2d(inputs, 64, [11, 11], 4, padding='VALID',
                    scope='conv1')
  net = slim.max_pool2d(net, [3, 3], 2, scope='pool1')
  net = slim.conv2d(net, 192, [5, 5], scope='conv2')
  net = slim.max_pool2d(net, [3, 3], 2, scope='pool2')
  net = slim.conv2d(net, 384, [3, 3], scope='conv3')
  net = slim.conv2d(net, 384, [3, 3], scope='conv4')
  net = slim.conv2d(net, 256, [3, 3], scope='conv5')
  net = slim.max_pool2d(net, [3, 3], 2, scope='pool5')

  # Use conv2d instead of fully_connected layers.
  with slim.arg_scope([slim.conv2d],
                      weights_initializer=trunc_normal(0.005),
                      biases_initializer=tf.constant_initializer(0.1)):
    net = slim.conv2d(net, 4096, [5, 5], padding='VALID',
                      scope='fc6')
    net = slim.dropout(net, dropout_keep_prob, is_training=is_training,
                       scope='dropout6')
    net = slim.conv2d(net, 4096, [1, 1], scope='fc7')
    # Convert end_points_collection into a end_point dict.
    end_points = slim.utils.convert_collection_to_dict(
        end_points_collection)
    if global_pool:
      net = tf.reduce_mean(net, [1, 2], keep_dims=True, name='global_pool')
      end_points['global_pool'] = net
    if num_classes:
      net = slim.dropout(net, dropout_keep_prob, is_training=is_training,
                         scope='dropout7')
      net = slim.conv2d(net, num_classes, [1, 1],
                        activation_fn=None,
                        normalizer_fn=None,
                        biases_initializer=tf.zeros_initializer(),
                        scope='fc8')
      if spatial_squeeze:
        net = tf.squeeze(net, [1, 2], name='fc8/squeezed')
      end_points[sc.name + '/fc8'] = net
    return net, end_points
alexnet_v2.default_image_size = 224
```

Architecture



Multi-GPU

Architecture

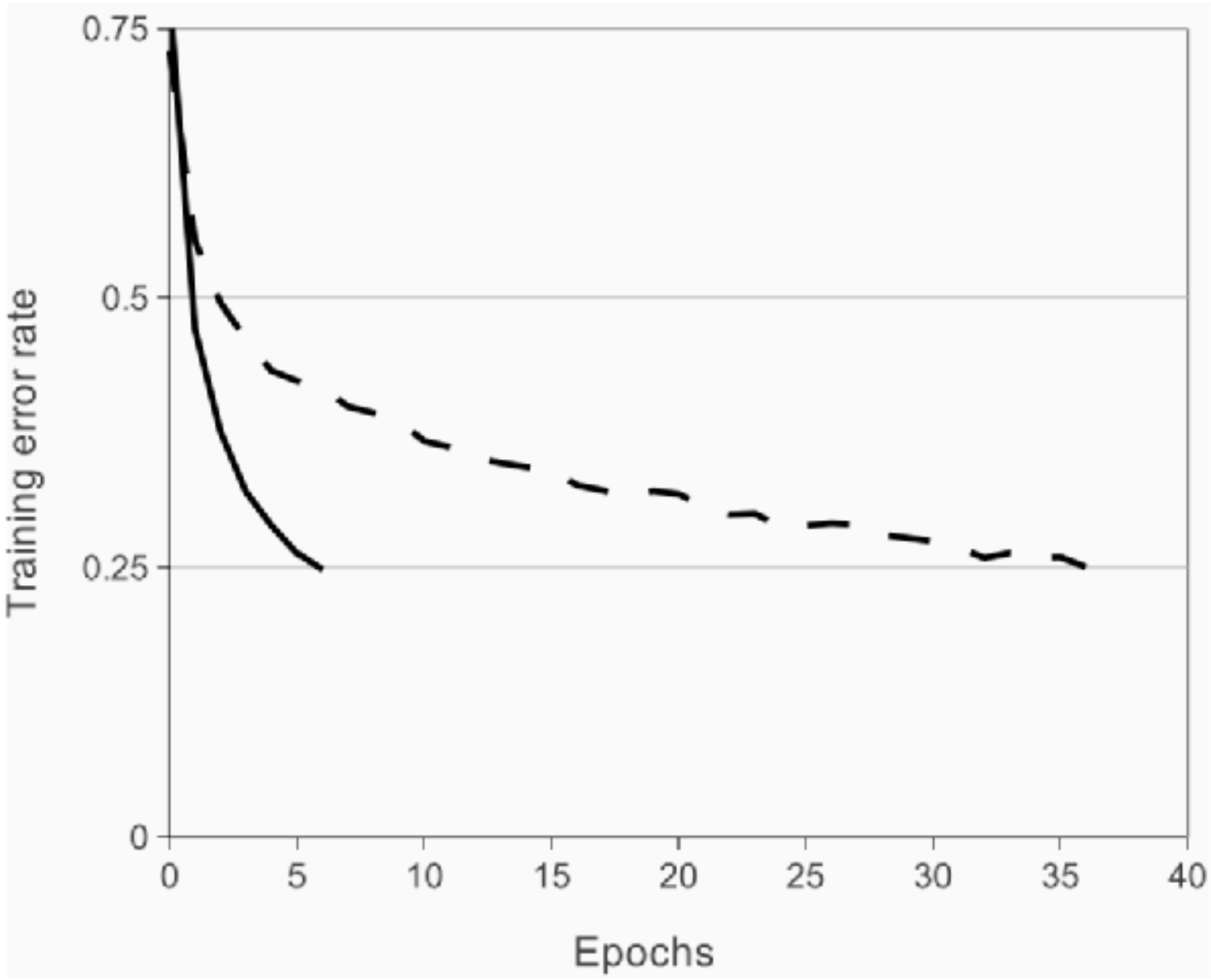


Communication Layer

Details / Retrospectives

Details / Retrospectives

- **ReLU**
- **Norm Layer**
- **Drop Out**



Details / Retrospectives

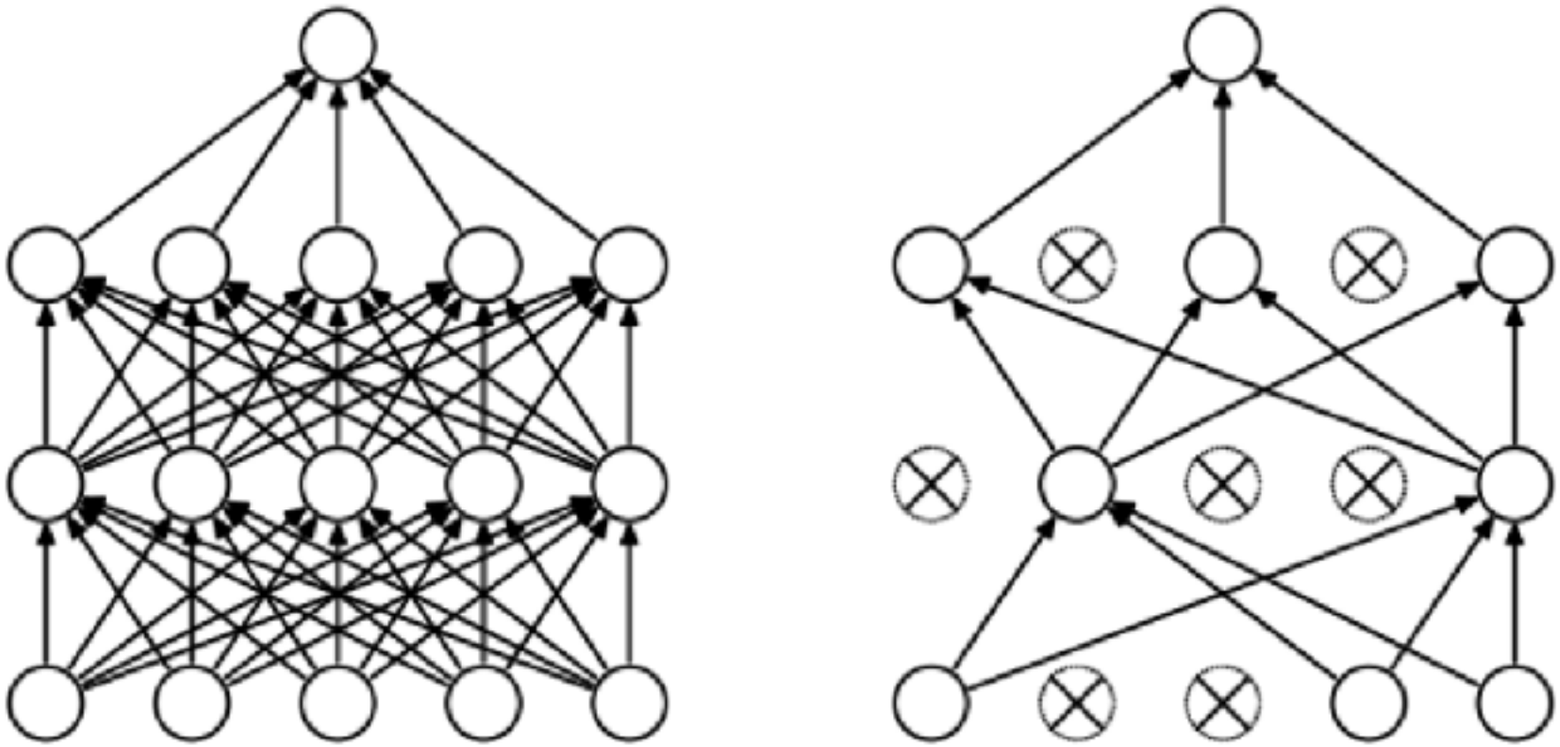
- **ReLU**
- **Norm Layer**
- **Drop Out**

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

adjacent kernel

Details / Retrospectives

- **ReLU**
- **Norm Layer**
- **Drop Out**



(a) Standard Neural Net

(b) After applying dropout.

Results

Results

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**