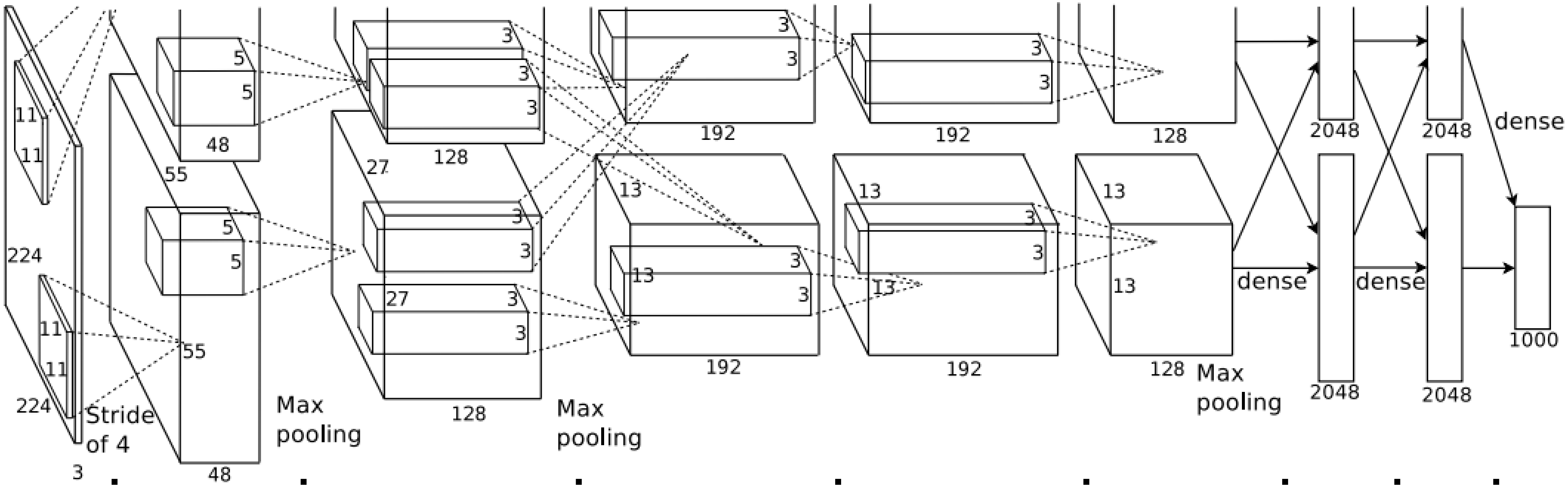
AlexNet

Winner of ILSVRC 2012

Overall Architecture



Input | 1st Layer | 2nd Layer | 3rd Layer | 4th Layer | 5th Layer | 6th Layer | 7th Layer | Output

Values from the paper

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

outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We applied this normalization after applying the ReLU nonlinearity in certain layers (see Section 3.5).

[17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size $z \times z$ centered at the location of the pooling unit. If we set $s = z$, we obtain traditional local pooling as commonly employed in CNNs. If we set $s < z$, we obtain overlapping pooling. This is what we use throughout our network, with $s = 2$ and $z = 3$. This scheme reduces the top-1 and top-5 error rates by 0.4% and

Values from the paper

connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, of the kind described in Section 3.4, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

factor of two during training. The recently-introduced technique, called "dropout" [10], consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are

We use dropout in the first two fully-connected layers of Figure 2. Without dropout, our network exhibits substantial overfitting. Dropout roughly doubles the number of iterations required to converge.

1^{st} Layer

The first convolutional layer filters the $224 \times 224 \times 3$ input image with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring

```
import conv2d from tensorflow.contrib.layers
import max_pool2d from tensorflow.contrib.layers
import local_response_normalization from tensorflow.nn

conv1 = conv2d(input,
              num_outputs=96,
              kernel_size=[11,11],
              stride=4,
              padding="VALID",
              activation_fn=tf.nn.relu)

lrn1 = local_response_normalization(conv1,
                                    bias=2
                                    alpha=0.0001,
                                    beta=0.75)

pool1 = max_pool2d(lrn1,
                   kernel_size=[3,3],
                   stride=2)
```

2nd Layer

neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$.

```
conv2 = conv2d(lrn1,
               num_outputs=256,
               kernel_size=[5,5],
               stride=1,
               padding="VALID",
               activation_fn=tf.nn.relu)

lrn2 = local_response_normalization(conv2,
                                    bias=2
                                    alpha=0.0001,
                                    beta=0.75)

pool2 = max_pool2d(lrn2,
                   kernel_size=[3,3],
                   stride=2)
```

3rd Layer

pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth

```
conv3 = conv2d(pool2,
               num_outputs=384,
               kernel_size=[3,3],
               stride=1,
               padding="VALID",
               activation_fn=tf.nn.relu)
```

4^{th} Layer

256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

```
conv4 = conv2d(conv3,
               num_outputs=384,
               kernel_size=[3,3],
               stride=1,
               padding="VALID",
               activation_fn=tf.nn.relu)
```

5^{th} Layer

256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

```
conv5 = conv2d(conv4,
               num_outputs=256,
               kernel_size=[3,3],
               stride=1,
               padding="VALID",
               activation_fn=tf.nn.relu)

pool5 = max_pool2d(conv5,
                   kernel_size=[3,3],
                   stride=2)
```

$6^{th} \sim 7^{th}$, output Layers

256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

```
import flatten from tensorflow.contrib.layers
import fully_connected from tensorflow.contrib.layers

flat = flatten(pool5)

fcl1 = fully_connected(flat,
                       num_outputs=4096,
                       activation_fn=tf.nn.relu)

fcl2 = fully_connected(fcl1,
                       num_outputs=4096,
                       activation_fn=tf.nn.relu)

out = fully_connected(fc12,
                      num_outputs=1000,
                      activation_fn=None)
```