ELEG 4040
Tutorial 9
Caffe Reference Models

Kai KANG
kkang@ee.cuhk.edu.hk

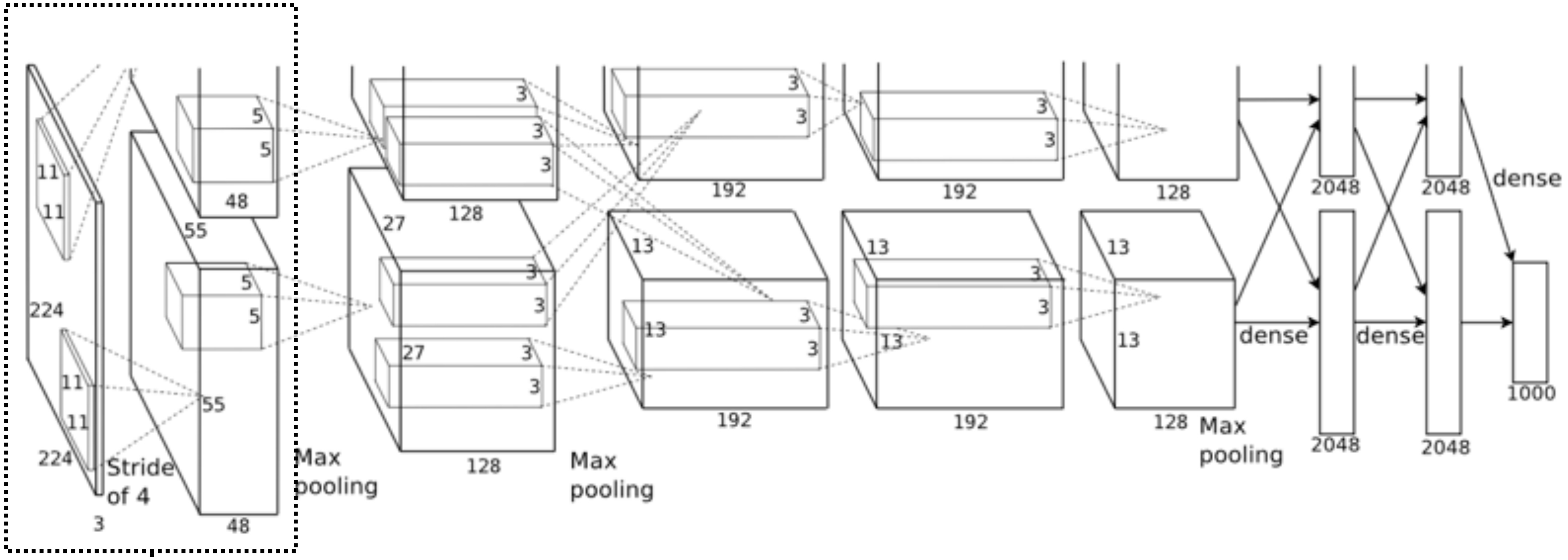
Reference Models

- AlexNet
- GoogLeNet
- Network in Network
- FCN-Xs
- …

AlexNet

- ImageNet 2012 Image Classification Challenge winner
- Originally trained on 2 GTX 580 GPUs because of insufficient memory
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

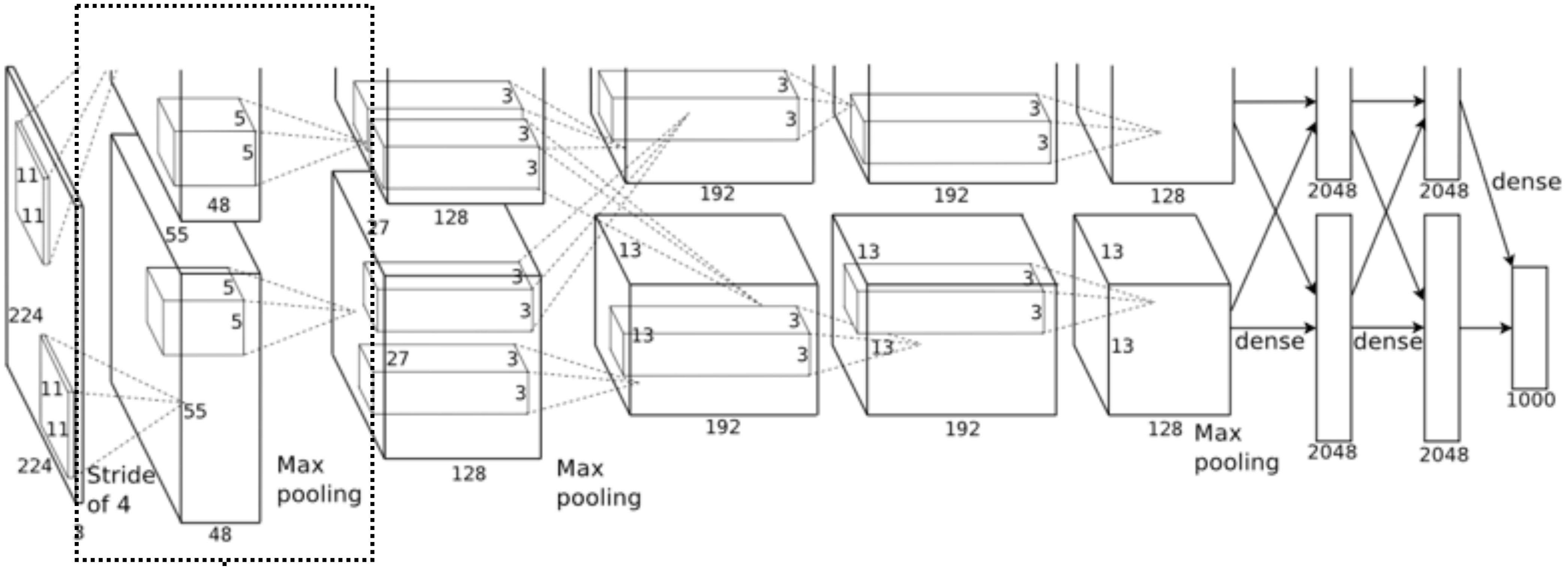
AlexNet



```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96 kernel_size: 11 stride: 4
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

put all 2x48 channels on a single GPU

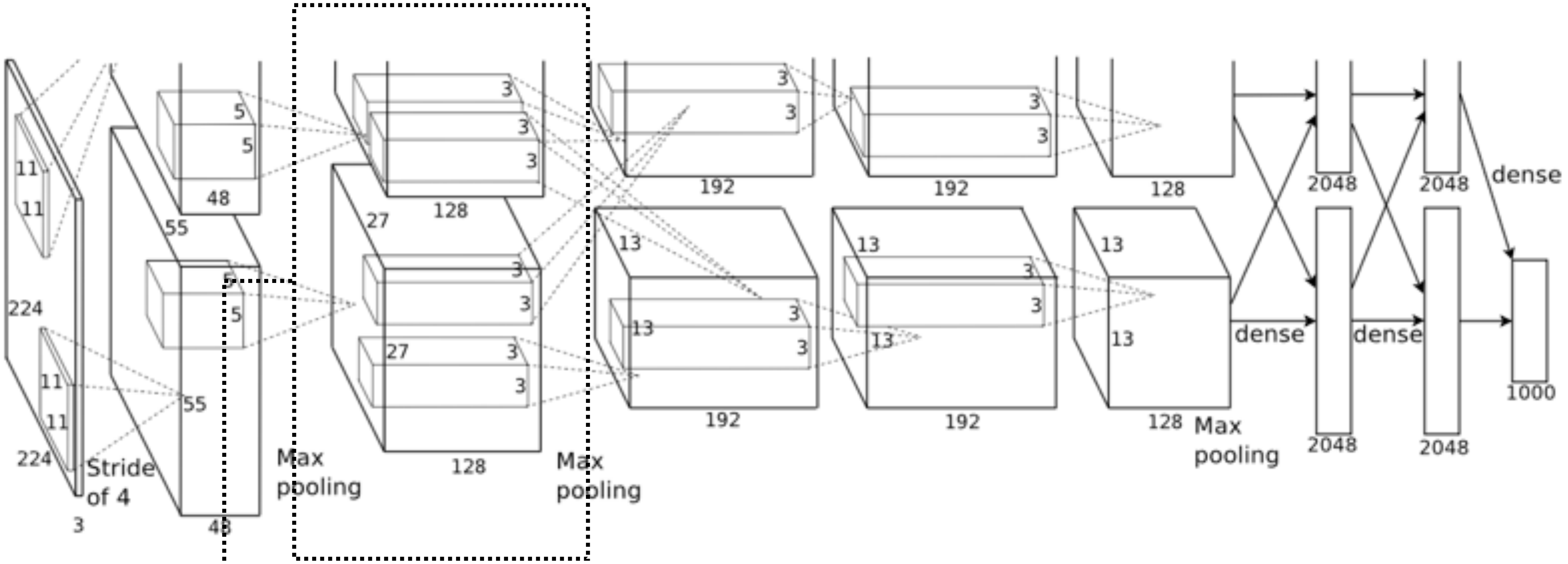
AlexNet



```
layer {
  name: "relu1" type: "ReLU" bottom: "conv1" top: "conv1"
}
layer {
  name: "norm1" type: "LRN" bottom: "conv1" top: "norm1"
  lrn_param { local_size: 5 alpha: 0.0001 beta: 0.75 }
}
layer {
  name: "pool1" type: "Pooling" bottom: "norm1" top: "pool1"
  pooling_param { pool: MAX kernel_size: 3 stride: 2 }
}
```

Local Response Normalization Layer

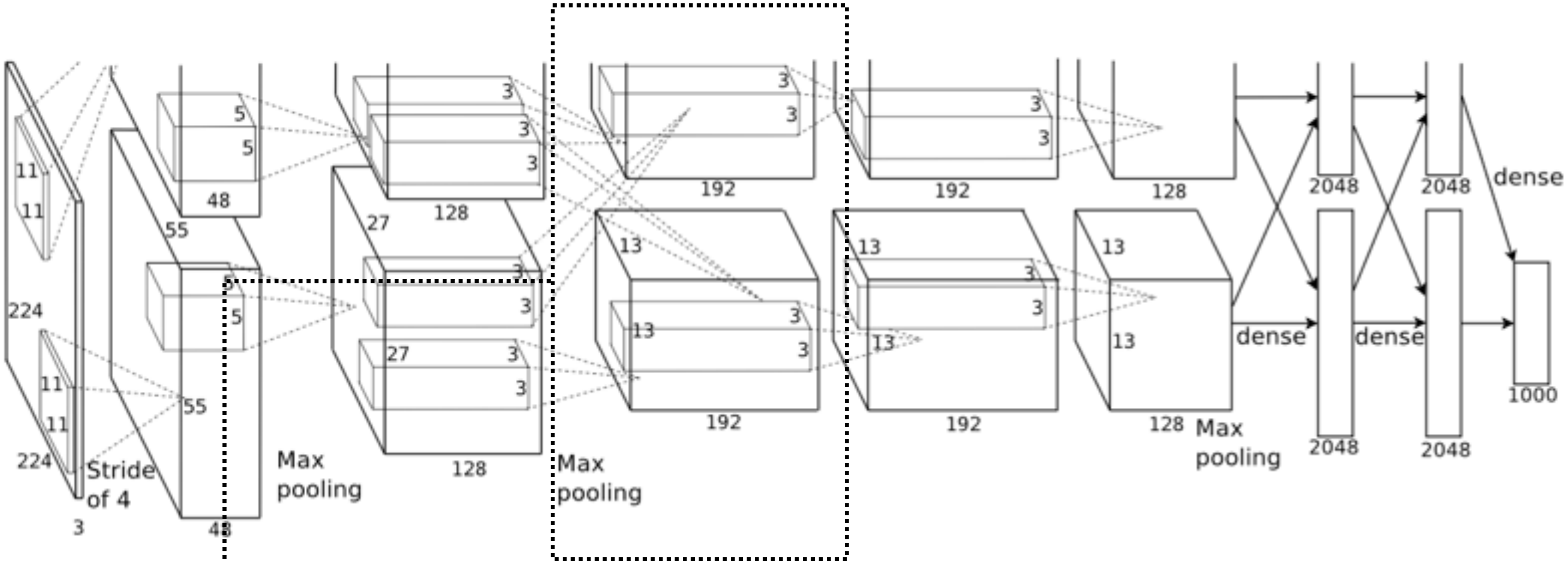
AlexNet



```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 256 pad: 2 kernel_size: 5 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

Divide input and output into two groups, equivalent to put on two GPUs

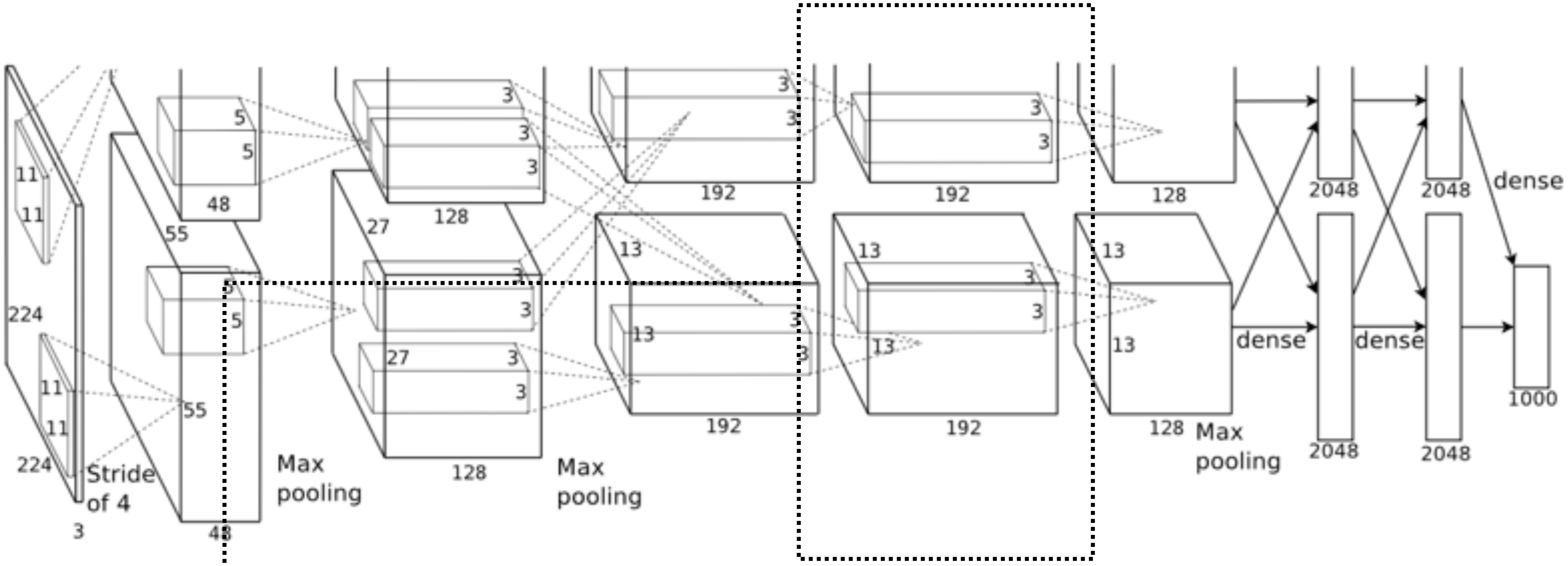
AlexNet



```
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 384 pad: 1 kernel_size: 3
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

No groups, use all input channels

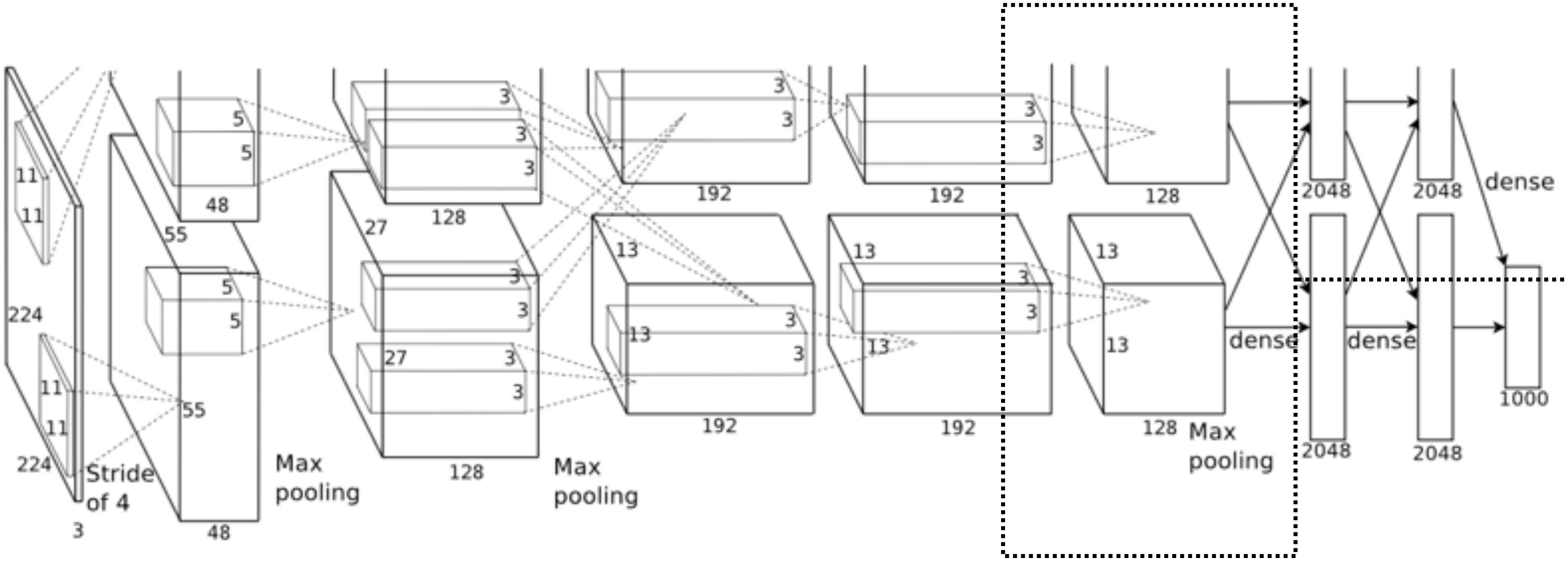
AlexNet



```
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 384 pad: 1 kernel_size: 3 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

Divide input and output into two groups, equivalent to put on two GPUs

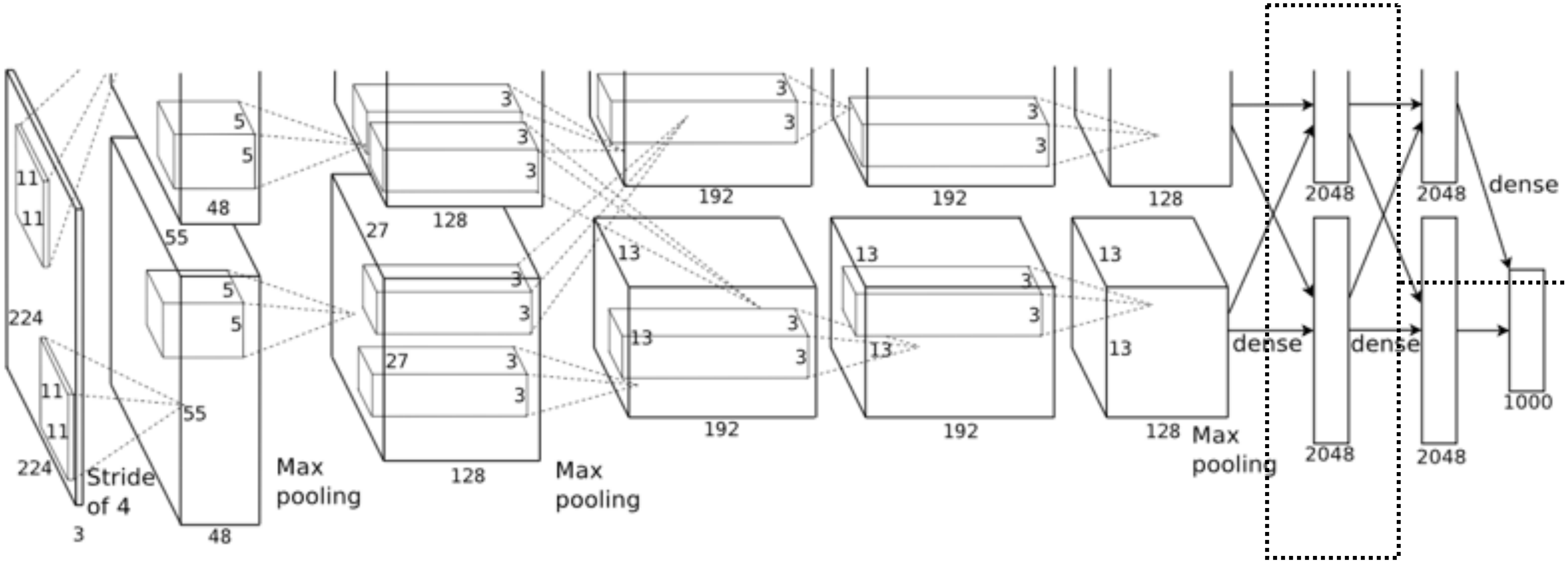
AlexNet



```
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 256 pad: 1 kernel_size: 3 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

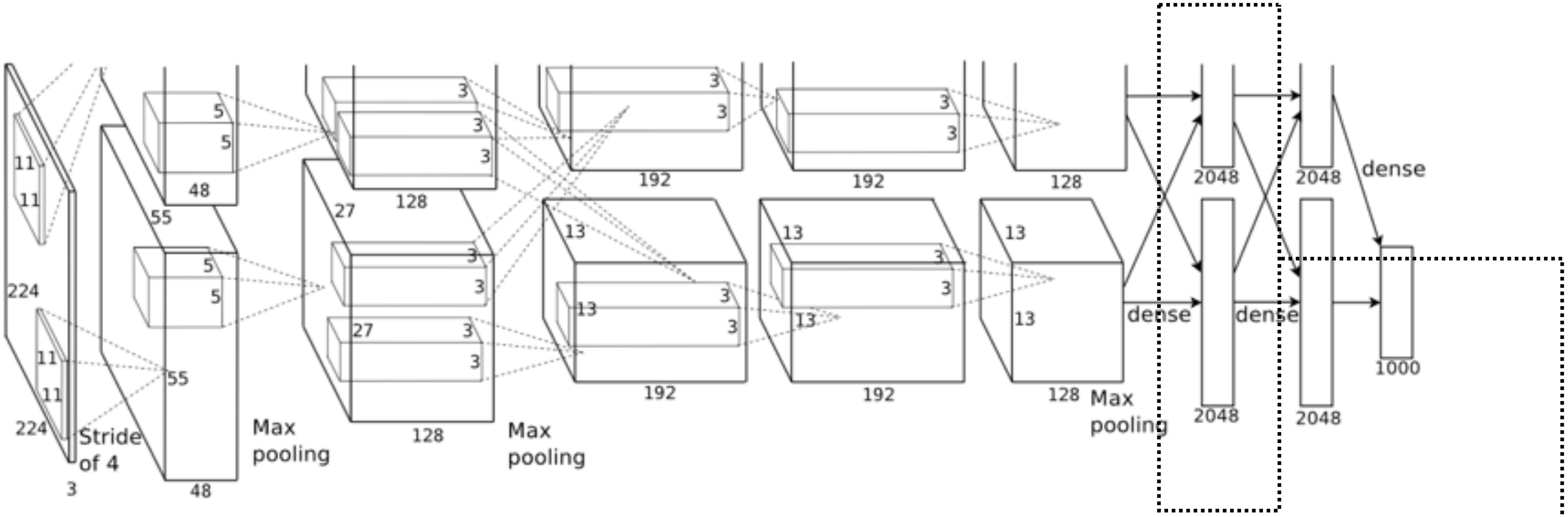
Divide input and output into two groups, equivalent to put on two GPUs

AlexNet



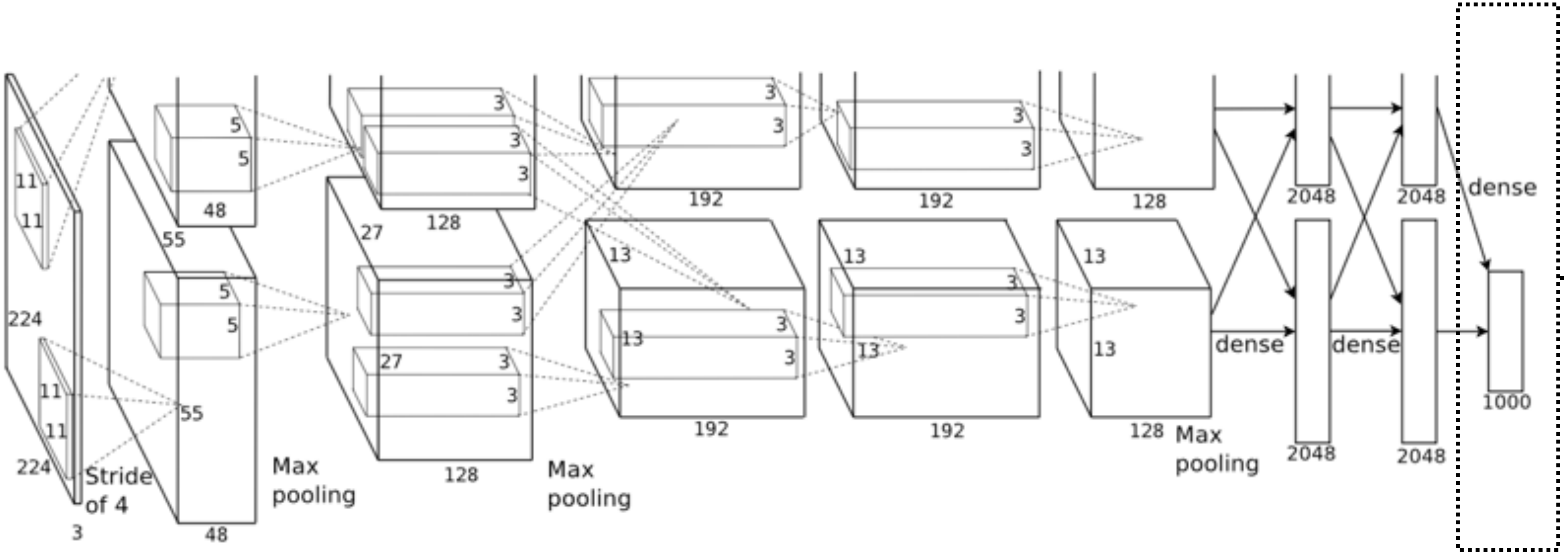
```
layer {
  name: "fc6"
  type: "InnerProduct"
  bottom: "pool5"
  top: "fc6"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 4096
    weight_filler { type: "gaussian" std: 0.005 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

AlexNet



```
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "fc6"
  top: "fc6"
}
layer {
  name: "drop6"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  dropout_param { dropout_ratio: 0.5 }
}
```

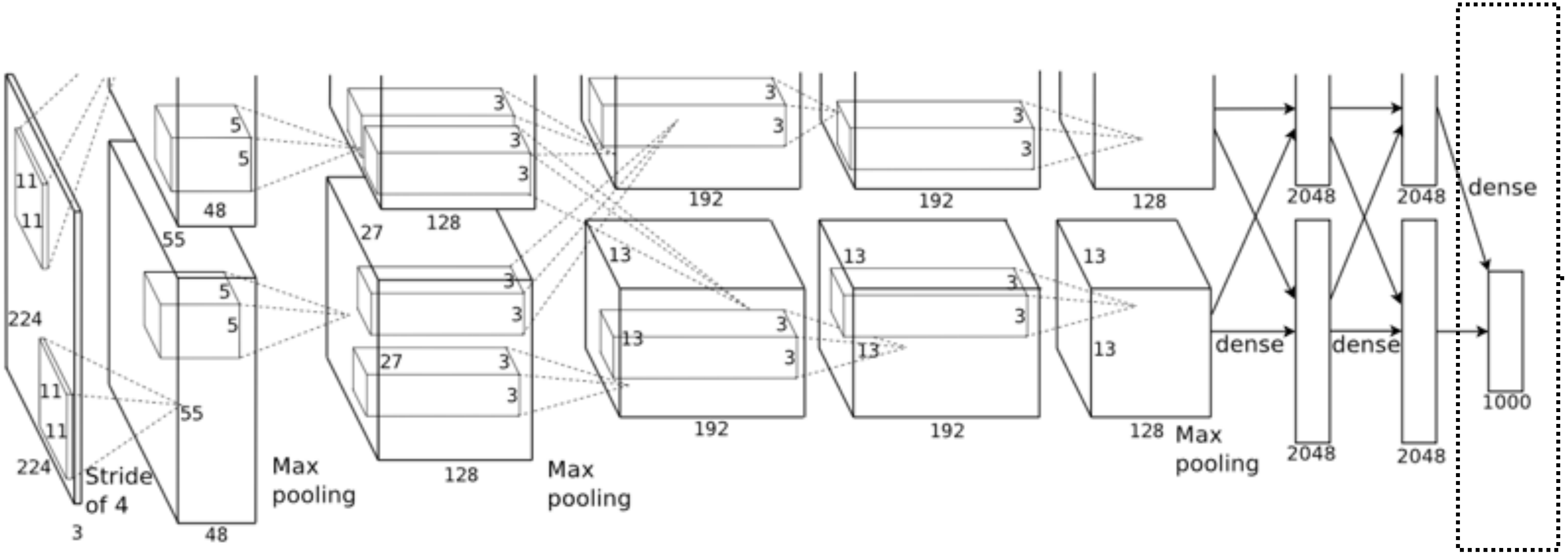
AlexNet



```
layer {
  name: "fc8"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 1000
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

1000 neurons for
1000 classes

AlexNet



```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8"
  bottom: "label"
  top: "loss"
}
```

AlexNet

- Use `group` to convolve part of input channels to simulate two-GPU implementations
- Use local response normalization layer to normalize input responses
- Differences to original implementation
  - no relighting data augmentation
  - initializing non-zero biases to 0.1 instead of 1

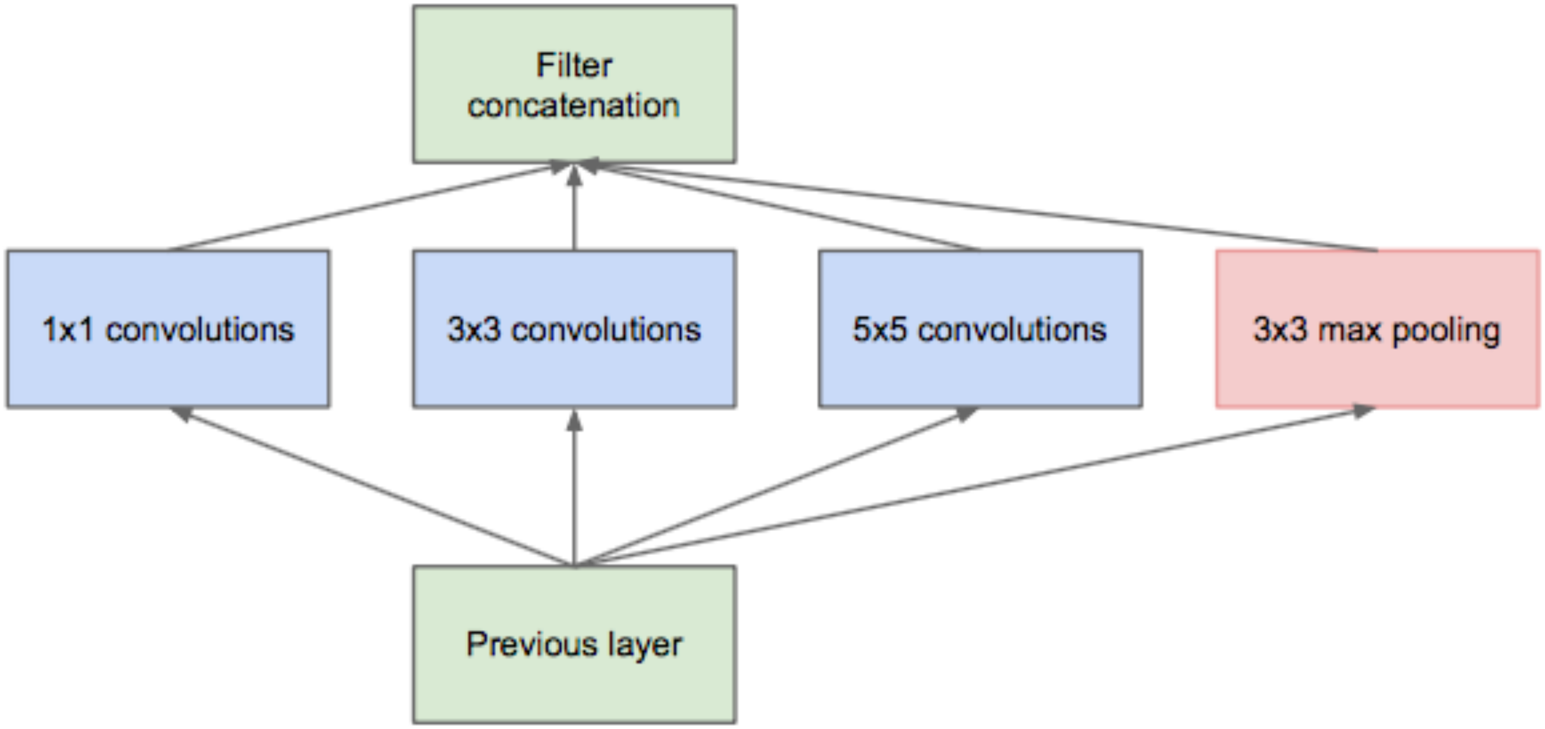
GoogLeNet

- ImageNet 2014 Image Classification and Object Detection winner

- Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).

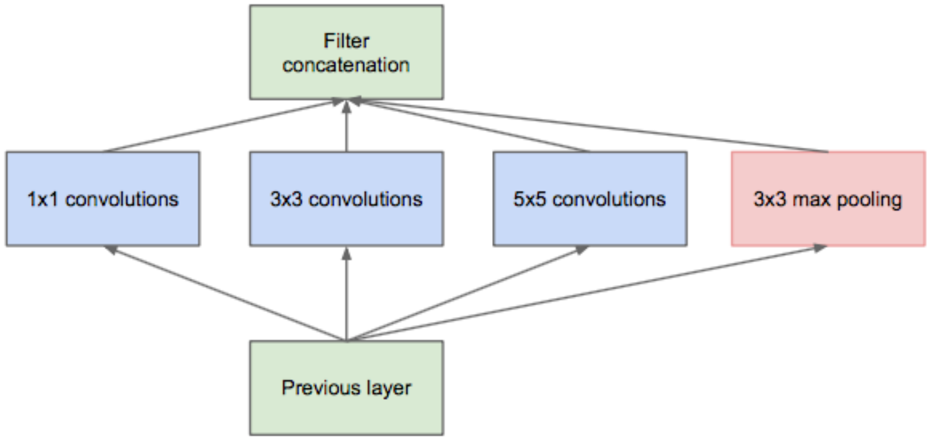
GoogLeNet

Inception Module - Naive Version



GoogLeNet

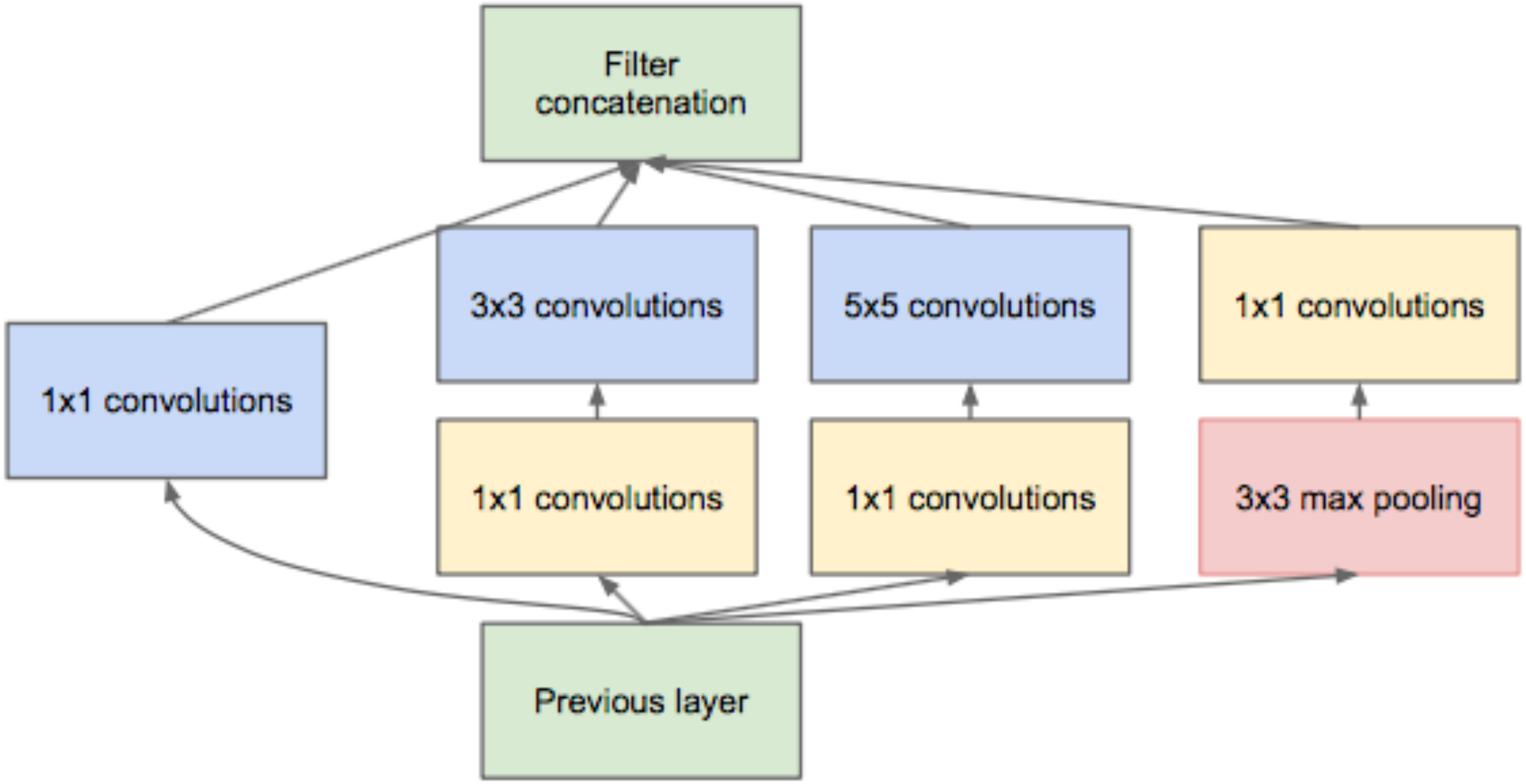
Inception Module - Naive Version



```
layer { name: "inception/1x1" type: "Convolution"
  bottom: "previous" top: "inception/1x1" ... }
layer { name: "inception/relu_1x1" type: "ReLU"
  bottom: "inception/1x1" top: "inception/1x1" }
layer { name: "inception/3x3" type: "Convolution"
  bottom: "previous" top: "inception/3x3" ... }
layer { name: "inception/relu_3x3" type: "ReLU"
  bottom: "inception/3x3" top: "inception/3x3" }
layer { name: "inception/5x5" type: "Convolution"
  bottom: "previous" top: "inception/5x5" ...}
layer { name: "inception/relu_5x5" type: "ReLU"
  bottom: "inception/5x5" top: "inception/5x5" ...}
layer { name: "inception/pool" type: "Pooling"
  bottom: "previous" top: "inception/pool"
  pooling_param { pool: MAX kernel_size: 3 stride: 1 pad: 1 } }
layer { name: "inception/relu_pool_proj" type: "ReLU"
  bottom: "inception/pool_proj" top: "inception/pool_proj" ...}
layer { name: "inception/output"
  type: "Concat"
  bottom: "inception/1x1"
  bottom: "inception/3x3"
  bottom: "inception/5x5"
  bottom: "inception/pool_proj"
  top: "inception/output"
}
```

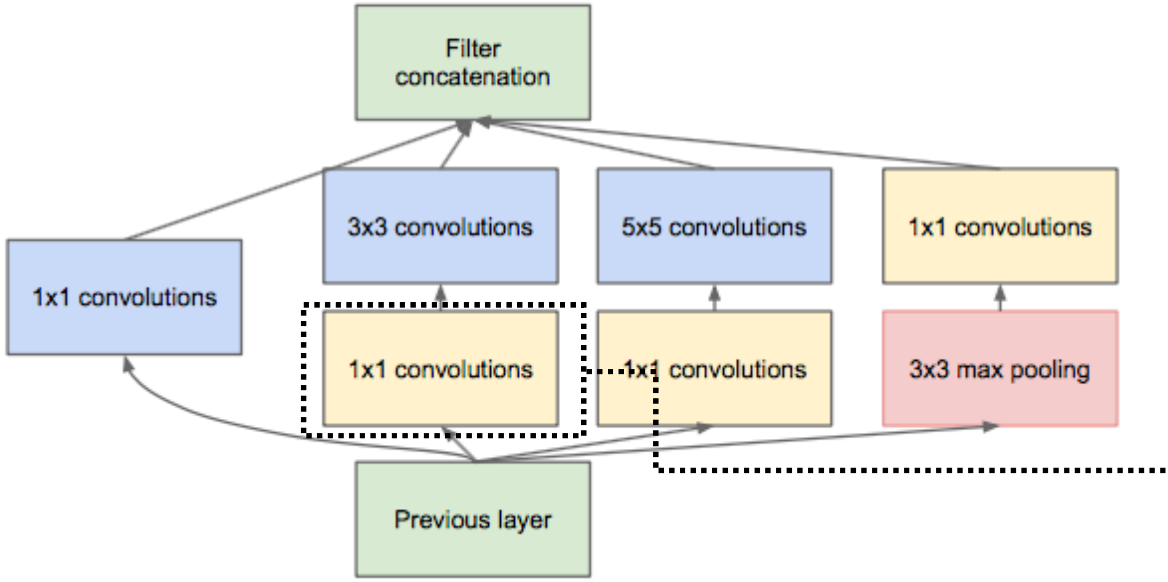
GoogLeNet

Inception Module - Dimension Reduction



GoogLeNet

Inception Module - Dimension Reduction

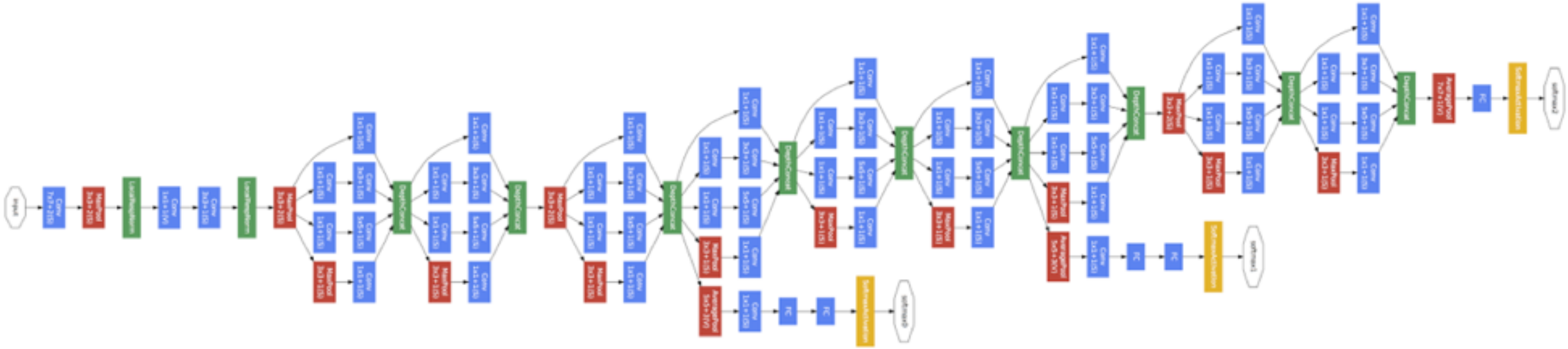


```
layer {
  name: "inception/3x3_reduce"
  type: "Convolution"
  bottom: "previous"
  top: "inception/3x3_reduce"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96 kernel_size: 1
    weight_filler { type: "xavier" std: 0.09 }
    bias_filler { type: "constant" value: 0.2 } }
}
layer {
  name: "inception/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception/3x3_reduce"
  top: "inception/3x3_reduce"
}
```

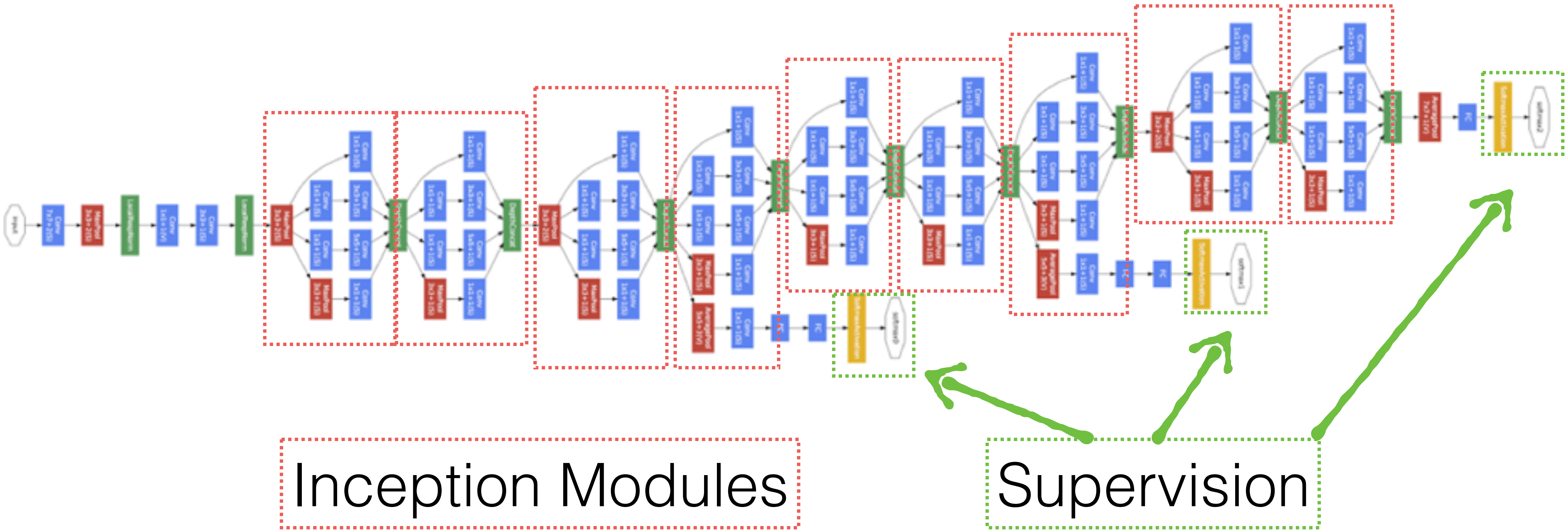
reduce channels with 1x1 kernels

GoogLeNet

Net Structure



GoogLeNet



Inception Modules

Supervision

```
layer {
  name: "loss1/classifier"
  type: "InnerProduct"
  bottom: "loss1/fc"
  top: "loss1/classifier"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 1000
    weight_filler { type: "xavier" std: 0.0009765625 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

```
layer {
  name: "loss1/loss"
  type: "SoftmaxWithLoss"
  bottom: "loss1/classifier"
  bottom: "label"
  top: "loss1/loss1"
  loss_weight: 0.3
}
```

GoogLeNet

- Differences:
  - not training with the relighting data-augmentation;
  - not training with the scale or aspect-ratio data-augmentation;
  - uses "xavier" to initialize the weights instead of "gaussian";

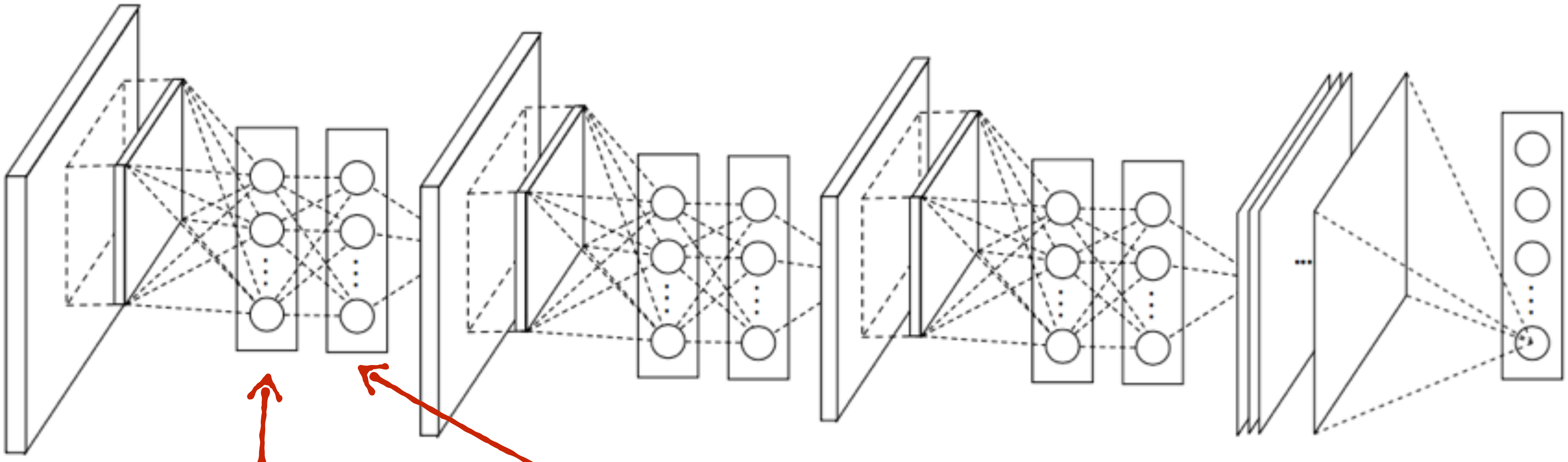
Network in Network

- ImageNet 2014 Object Detection (with provided data only) winner

- Use global pooling rather than fully connected layers for classification

- Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv: 1312.4400 (2013).

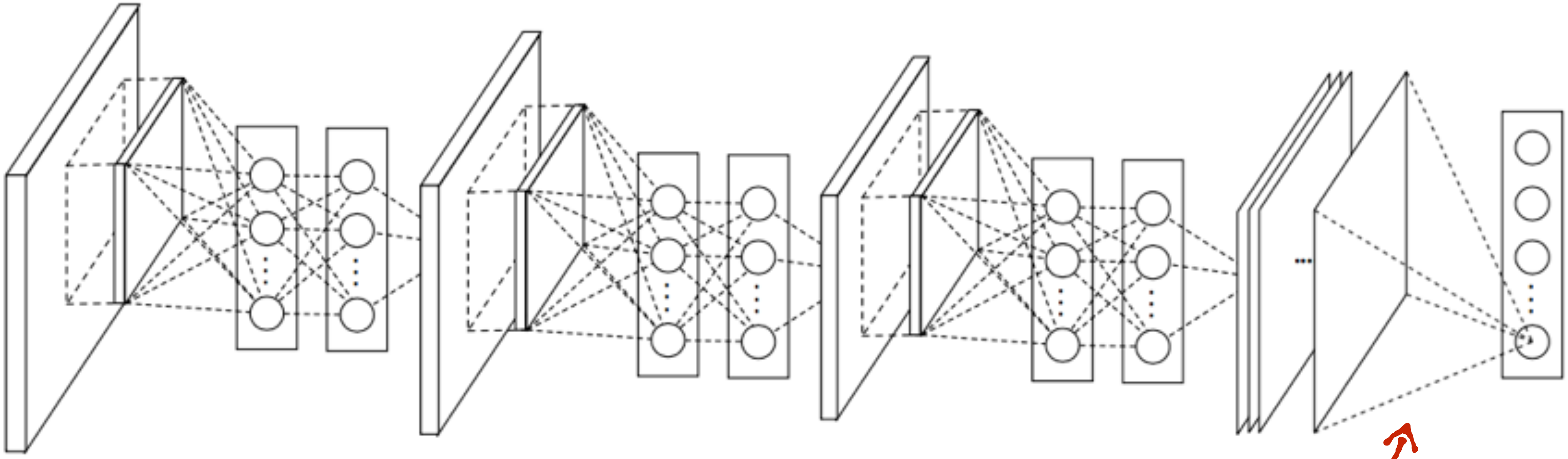
Network in Network



```
layers {
  bottom: "conv1"
  top: "cccp1"
  name: "cccp1"
  type: CONVOLUTION
  convolution_param {
    num_output: 96
    kernel_size: 1
    stride: 1
    weight_filler { type: "gaussian" mean: 0 std: 0.05 }
    bias_filler { type: "constant" value: 0 }
  }
}
layers {
  bottom: "cccp1"
  top: "cccp1"
  name: "relu1"
  type: RELU
}
```

```
layers {
  bottom: "cccp1"
  top: "cccp2"
  name: "cccp2"
  type: CONVOLUTION
  convolution_param {
    num_output: 96
    kernel_size: 1
    stride: 1
    weight_filler { type: "gaussian" mean: 0 std: 0.05 }
    bias_filler { type: "constant" value: 0 }
  }
}
layers {
  bottom: "cccp2"
  top: "cccp2"
  name: "relu2"
  type: RELU
}
```

Network in Network

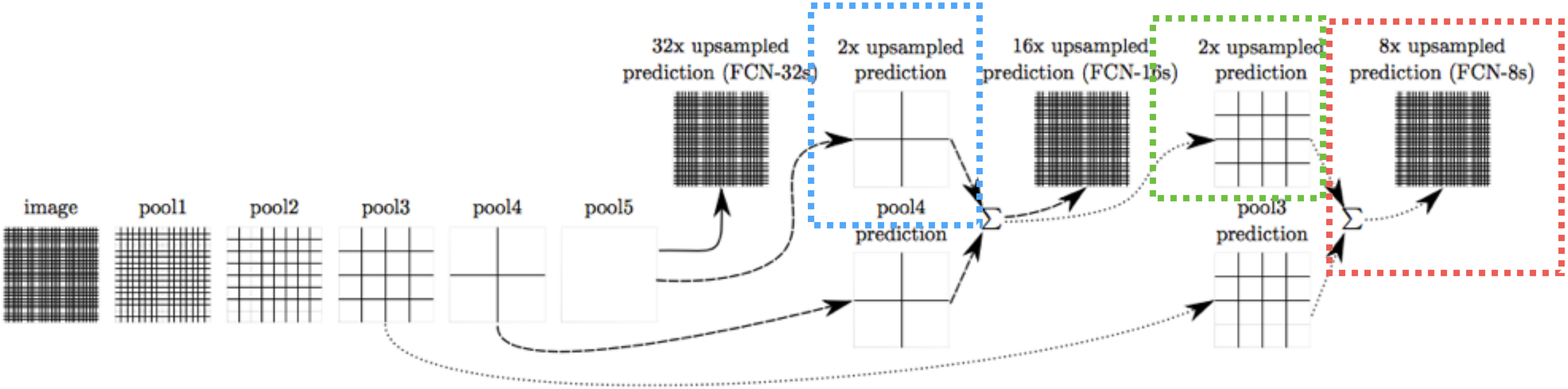


```
layers {
  bottom: "cccp8"
  top: "pool4"
  name: "pool4"
  type: POOLING
  pooling_param {
    pool: AVE
    global_pooling: true
  }
}
```

FCN-Xs

- State-of-the-art results on PASCAL VOC segmentation challenges
- Use trainable deconvolution layer to get dense prediction
- https://github.com/longjon/caffe/tree/future
- Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." arXiv preprint arXiv:1411.4038 (2014).

FCN-Xs



```
layers { type: DECONVOLUTION name: 'score2' bottom: 'score' top: 'score2'
  blobs_lr: 1 blobs_lr: 2 weight_decay: 1 weight_decay: 0
  convolution_param { kernel_size: 4 stride: 2 num_output: 21 } }
```

```
layers { type: DECONVOLUTION name: 'score4' bottom: 'score-fused'
  top: 'score4'
  blobs_lr: 1 weight_decay: 1
  convolution_param { bias_term: false kernel_size: 4 stride: 2 num_output: 21 } }
```

```
layers { type: DECONVOLUTION name: 'upsample'
  bottom: 'score-final' top: 'bigscore'
  blobs_lr: 0
  convolution_param { bias_term: false num_output: 21 kernel_size: 16 stride: 8 } }
```

deconvolution
for upsampling