



Image Analysis II

AlexNet, ZFNet, VGGNet, GoogLeNet, ResNet, ...

Radovan Fusek



CovNet Architectures

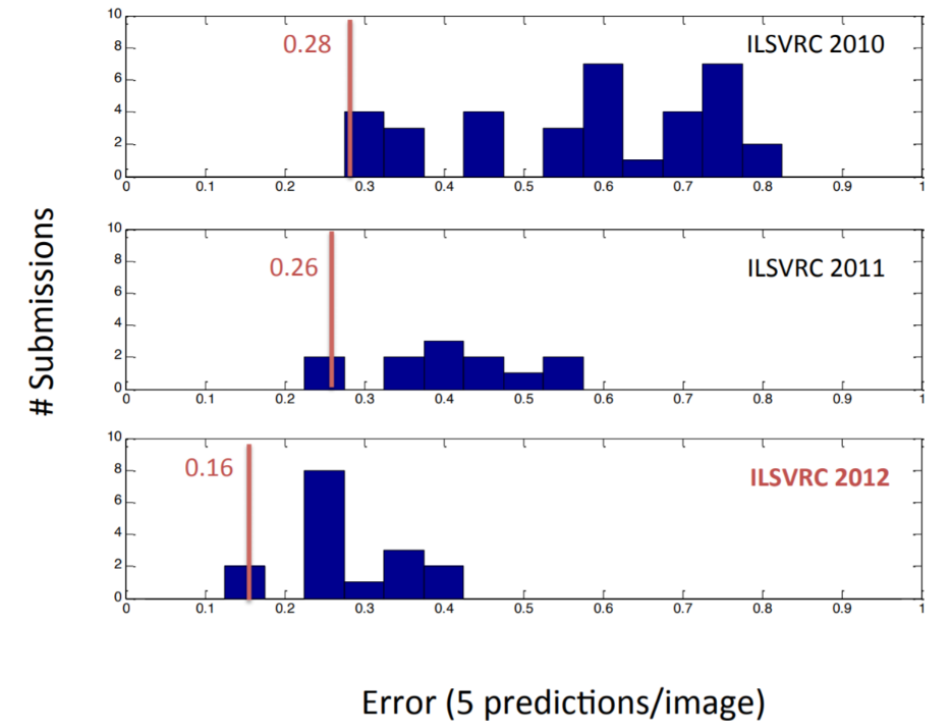
- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**



CovNet Architectures

2012

- ImageNet Database and competition
- <http://image-net.org/challenges/LSVRC/2012/>
- <http://www.image-net.org/>
- <http://image-net.org/challenges/LSVRC/2012/ilsvrc2012.pdf>



<https://en.wikipedia.org/wiki/ImageNet>

AlexNet - 2012

- similar architecture as LeNet but deeper (5 convolutional, 3 fully-connected)
- 1000 different classes (e.g. cats, dogs etc.)

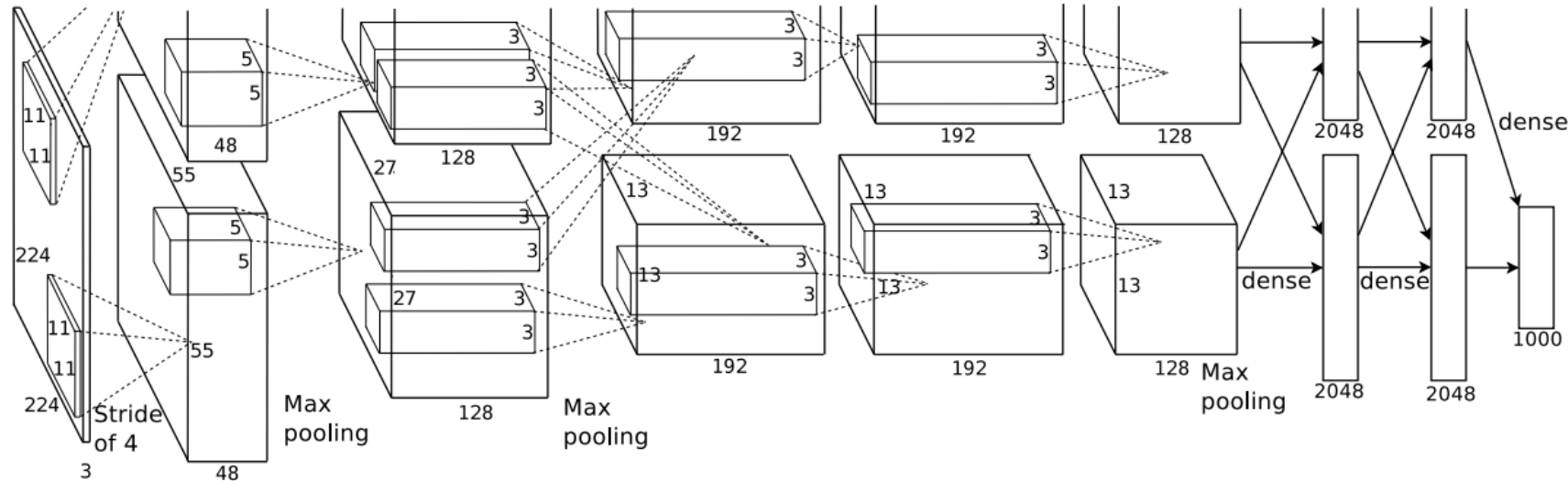
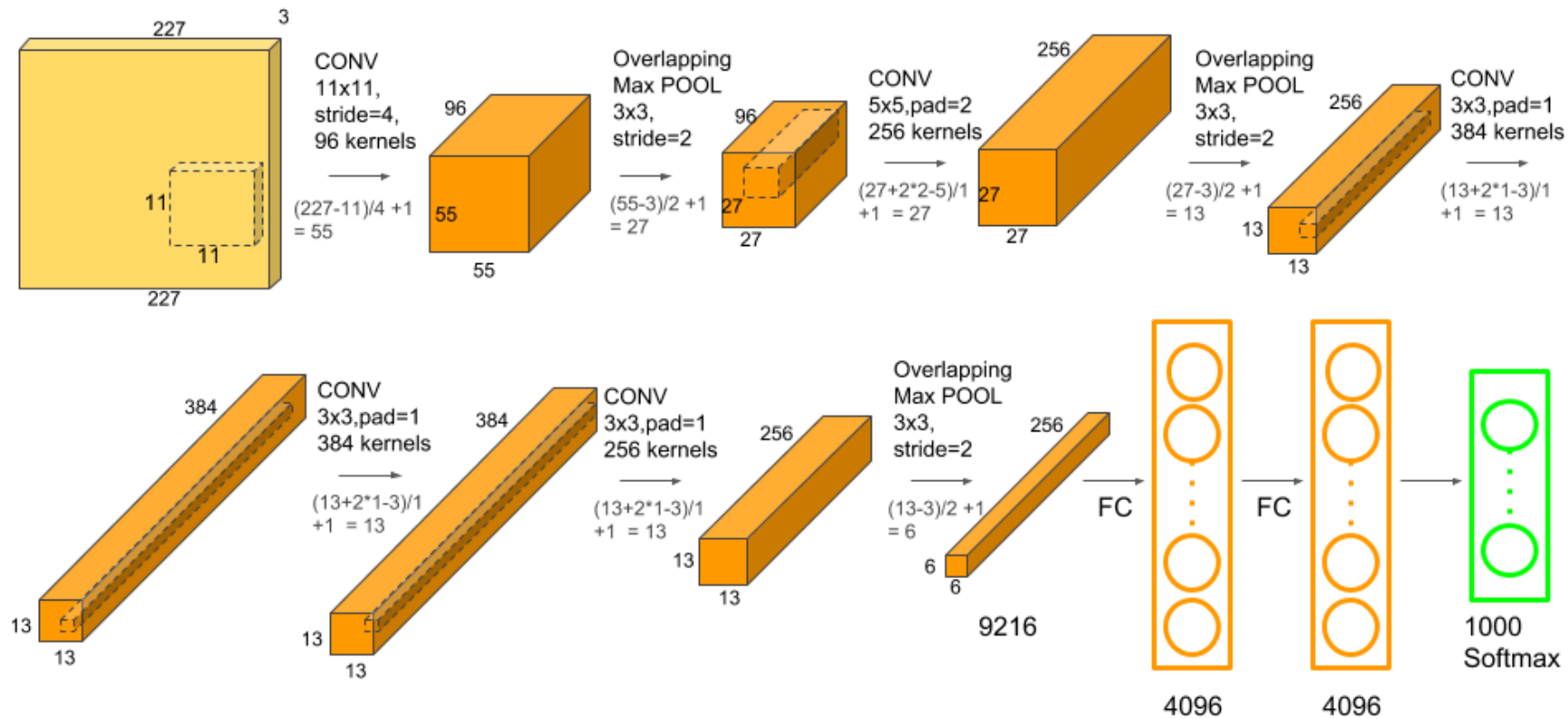


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet - 2012

- 5-6 days to train on two GTX 580 GPUs (90 epochs)
- Image augmentation (flip, color changes, ..)
- Dropout



<https://www.learnopencv.com/understanding-alexnet/>

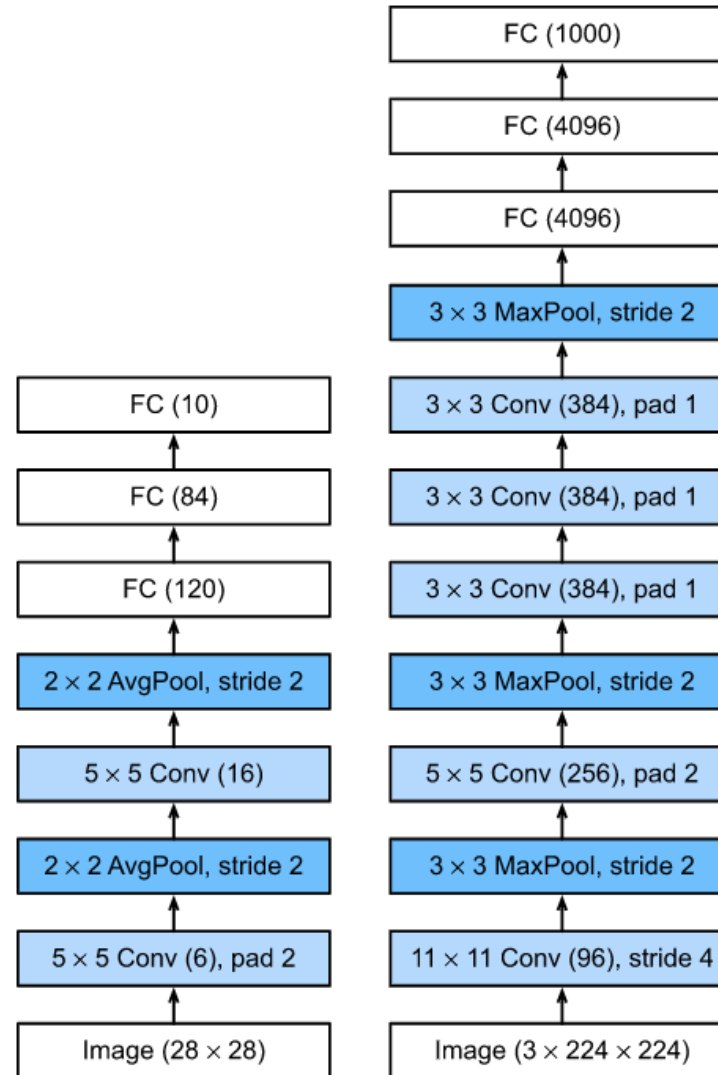
<https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848>

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012



http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

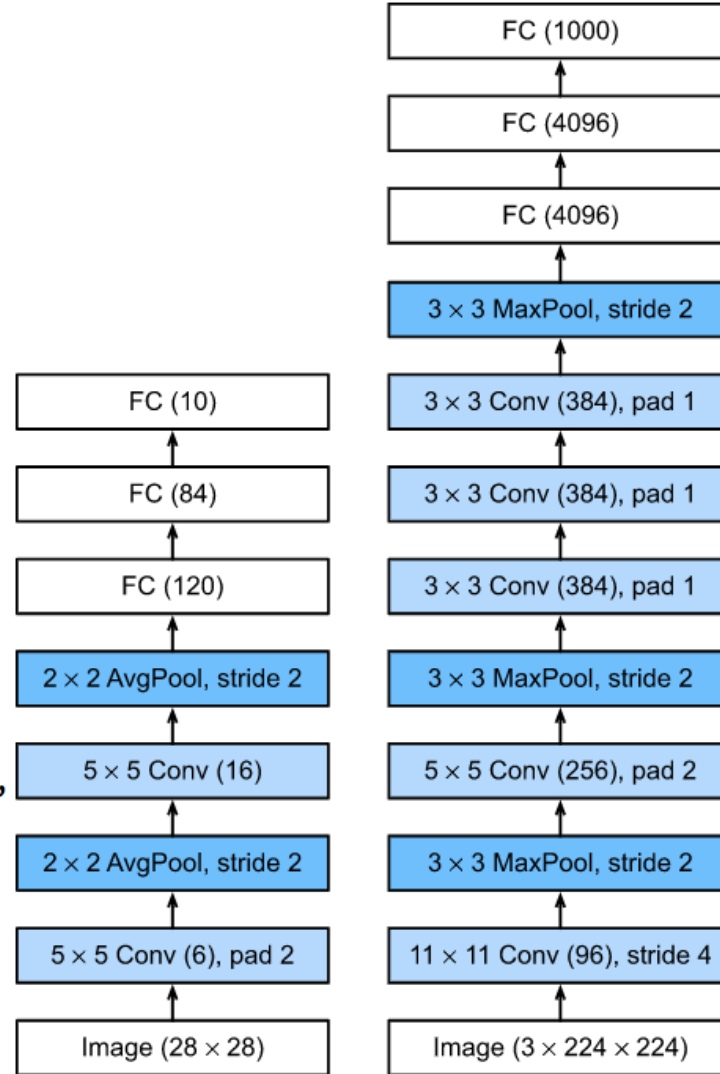
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

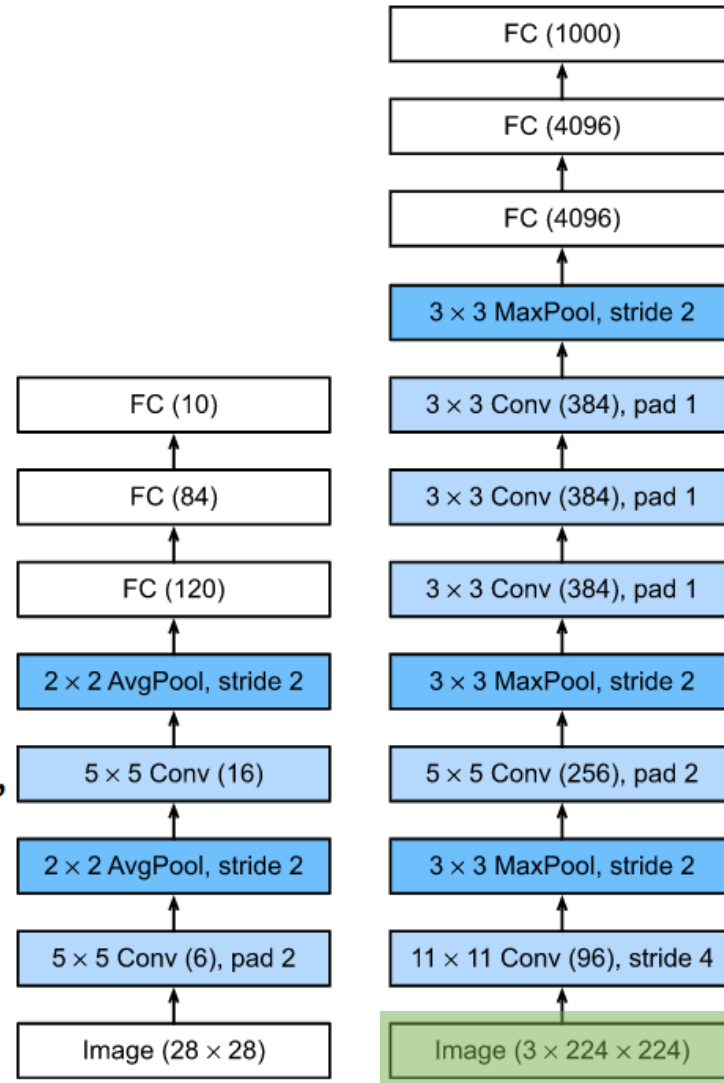
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`input<matrix<unsigned char>>`

http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

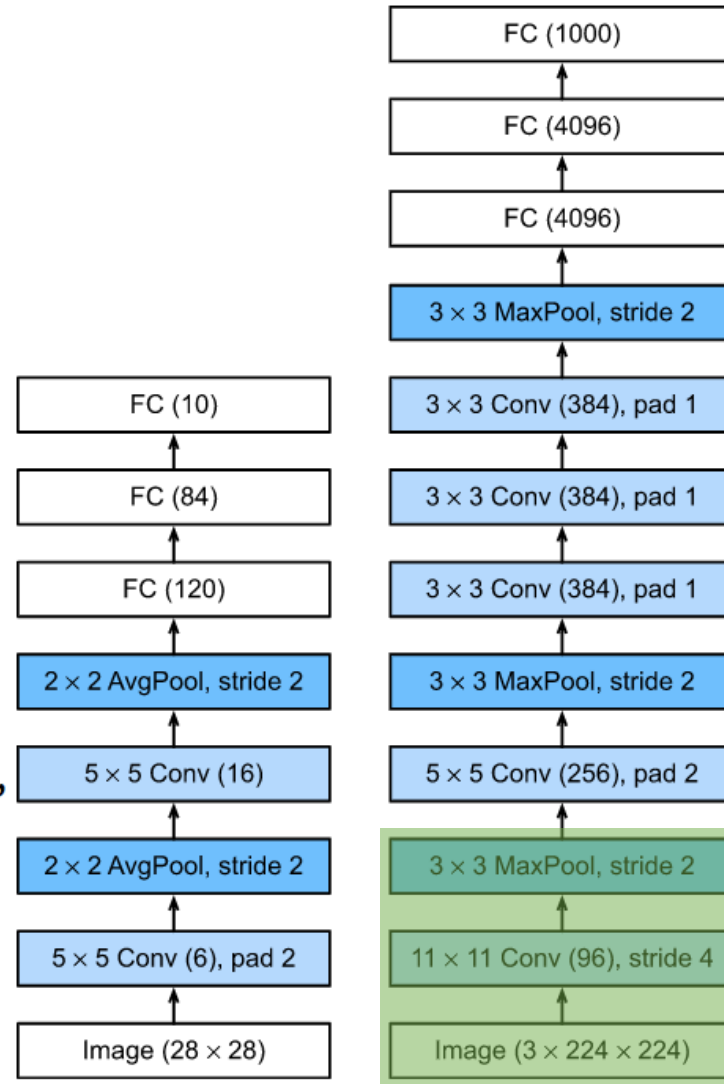
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`

http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

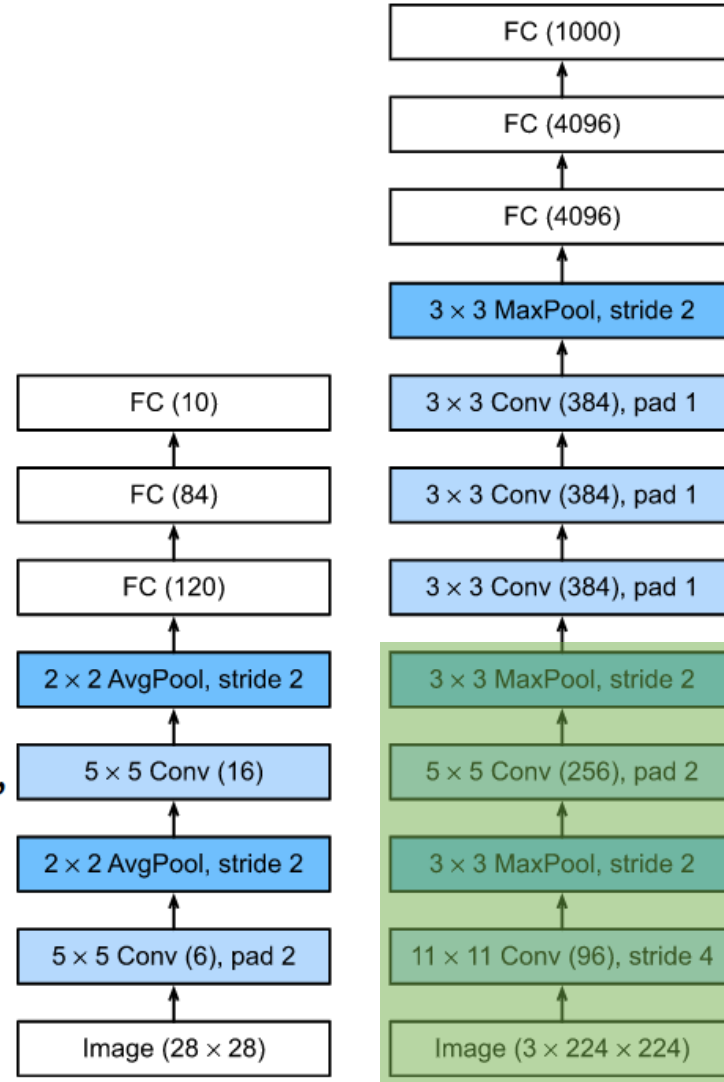
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`

http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

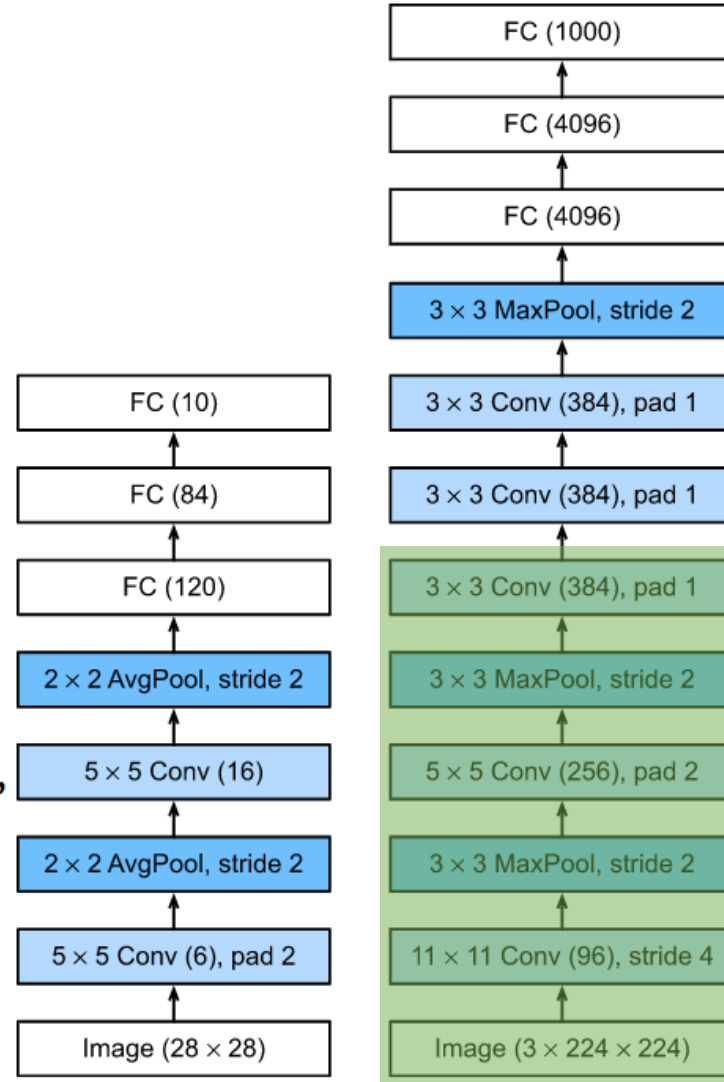
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`



AlexNet – 2012 (Dlib/C++)

`fc<10,`

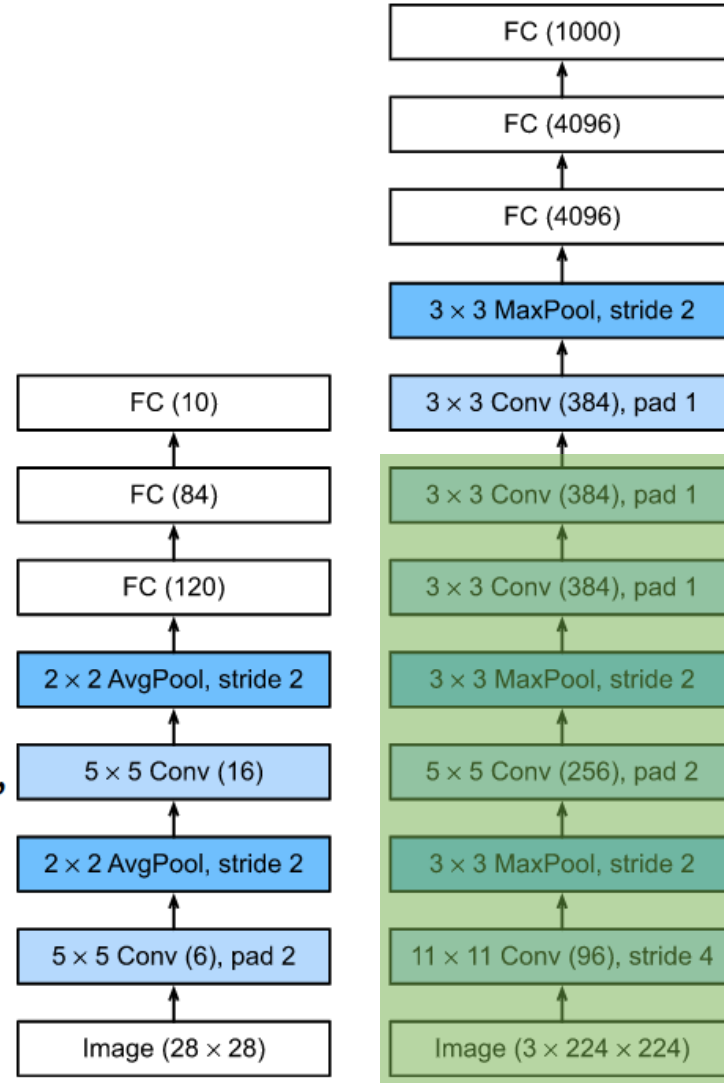
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`



AlexNet – 2012 (Dlib/C++)

`fc<10,`

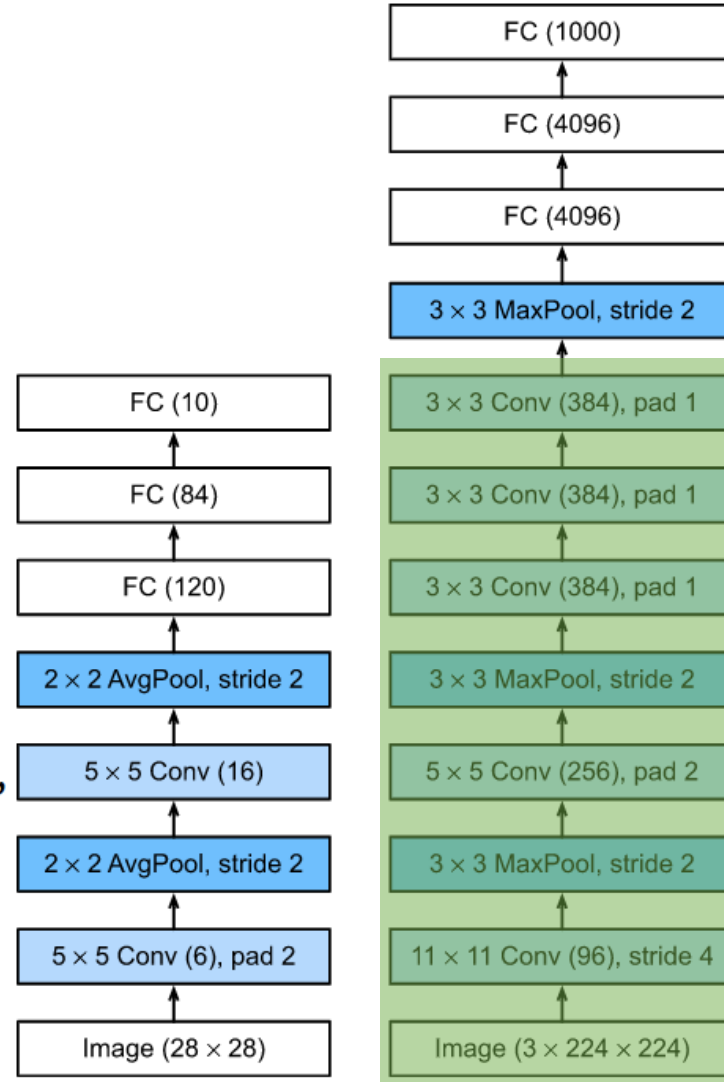
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`



AlexNet – 2012 (Dlib/C++)

`fc<10,`

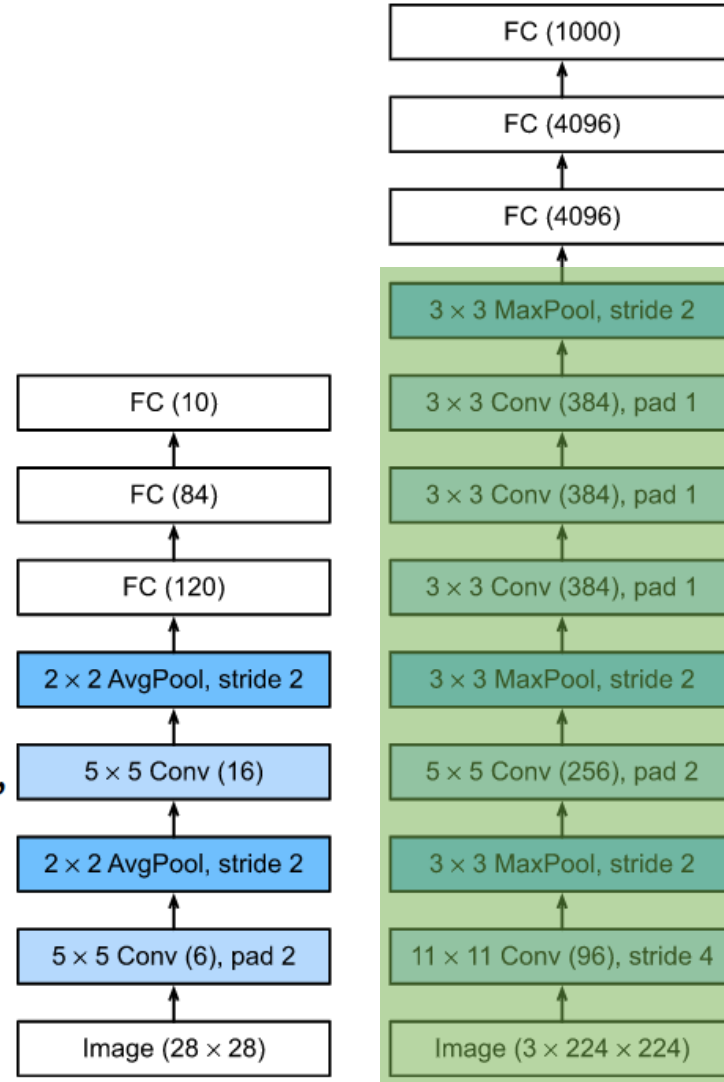
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`max_pool<3,3,2,2,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`



AlexNet – 2012 (Dlib/C++)

`fc<10,`

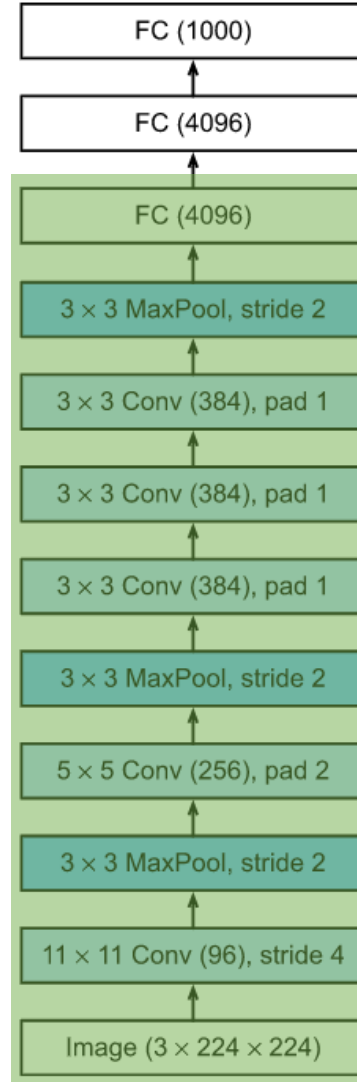
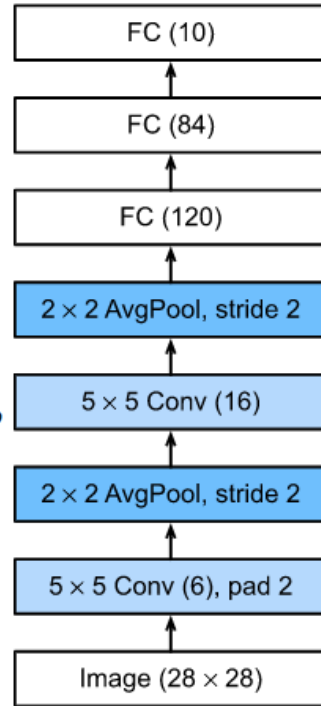
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`dropout<relu<fc<4096,`

`max_pool<3,3,2,2,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`



AlexNet – 2012 (Dlib/C++)

`fc<10,`

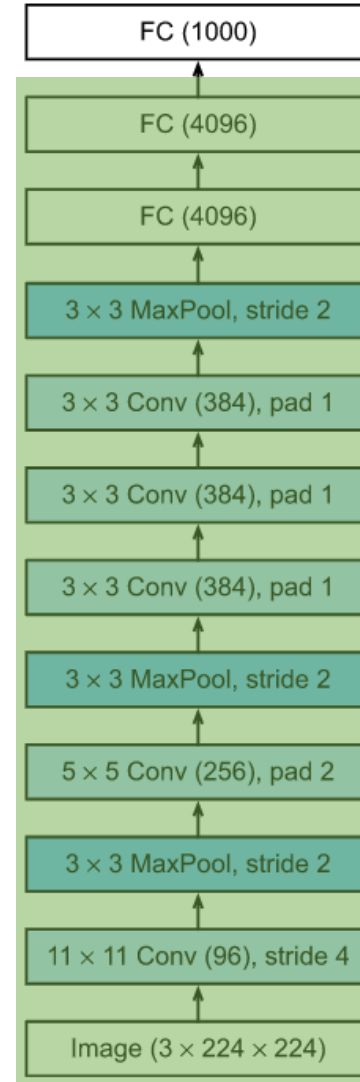
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`dropout<relu<fc<4096,`

`dropout<relu<fc<4096,`

`max_pool<3,3,2,2,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`

http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

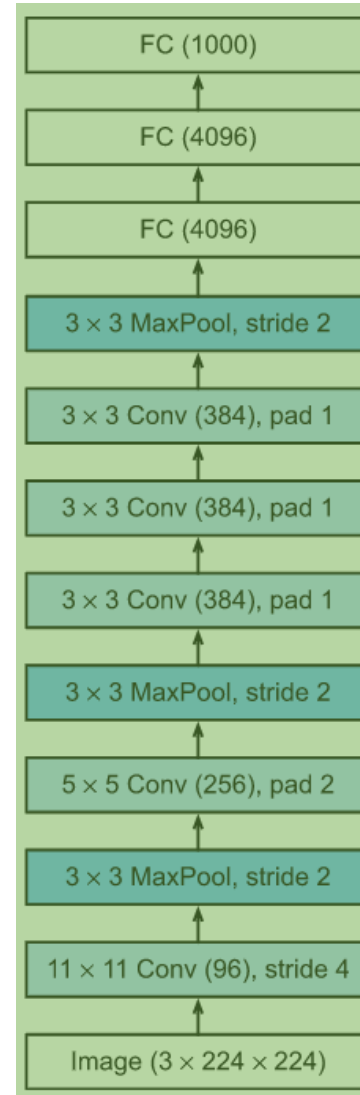
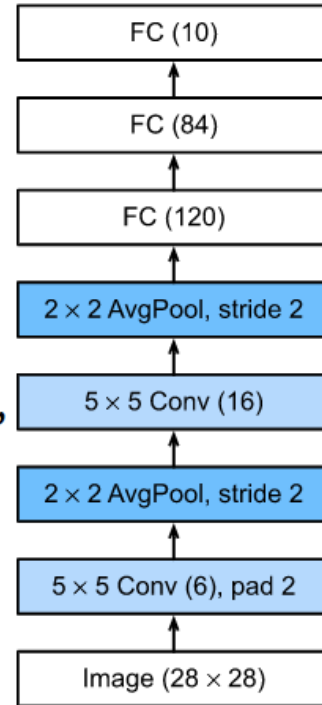
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`fc<1000,`

`dropout<relu<fc<4096,`

`dropout<relu<fc<4096,`

`max_pool<3,3,2,2,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`

http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (Dlib/C++)

`fc<10,`

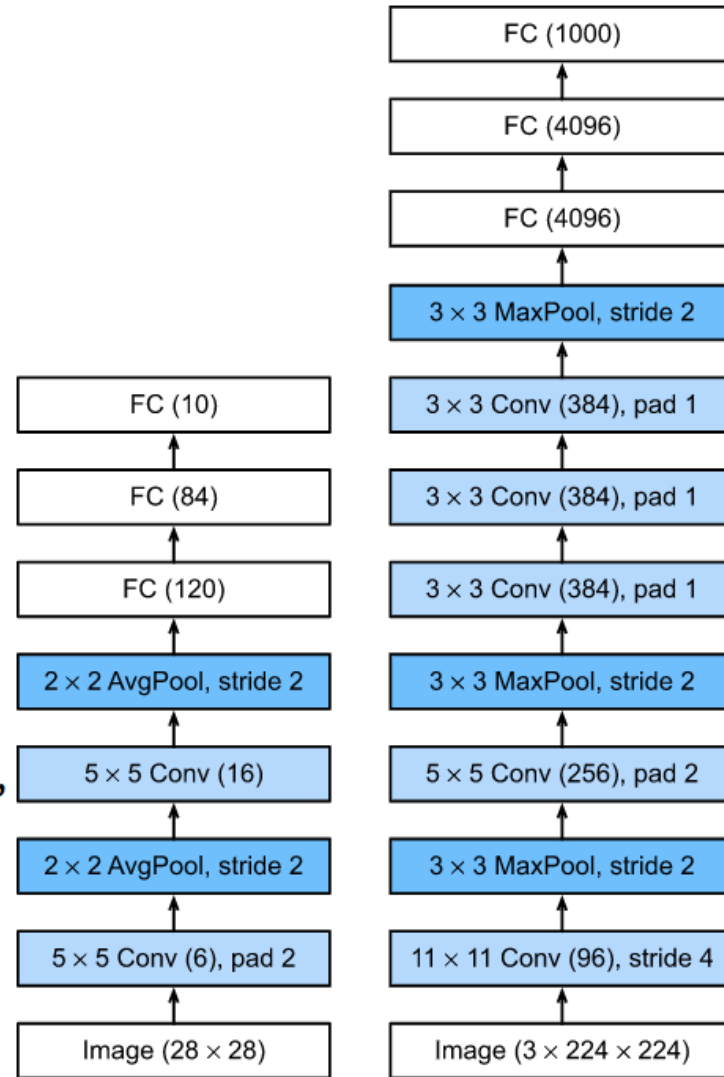
`relu<fc<84,`

`relu<fc<120,`

`max_pool<2,2,2,2,relu<con<16,5,5,1,1,`

`max_pool<2,2,2,2,relu<con<6,5,5,1,1,`

`input<matrix<unsigned char>>`



`fc<1000,`

`dropout<relu<fc<4096,`

`dropout<relu<fc<4096,`

`max_pool<3,3,2,2,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`relu<con<384,3,3,1,1,`

`max_pool<3,3,2,2,relu<con<256,5,5,1,1,`

`max_pool<3,3,2,2,relu<con<96,11,11,4,4,`

`input<matrix<unsigned char>>`

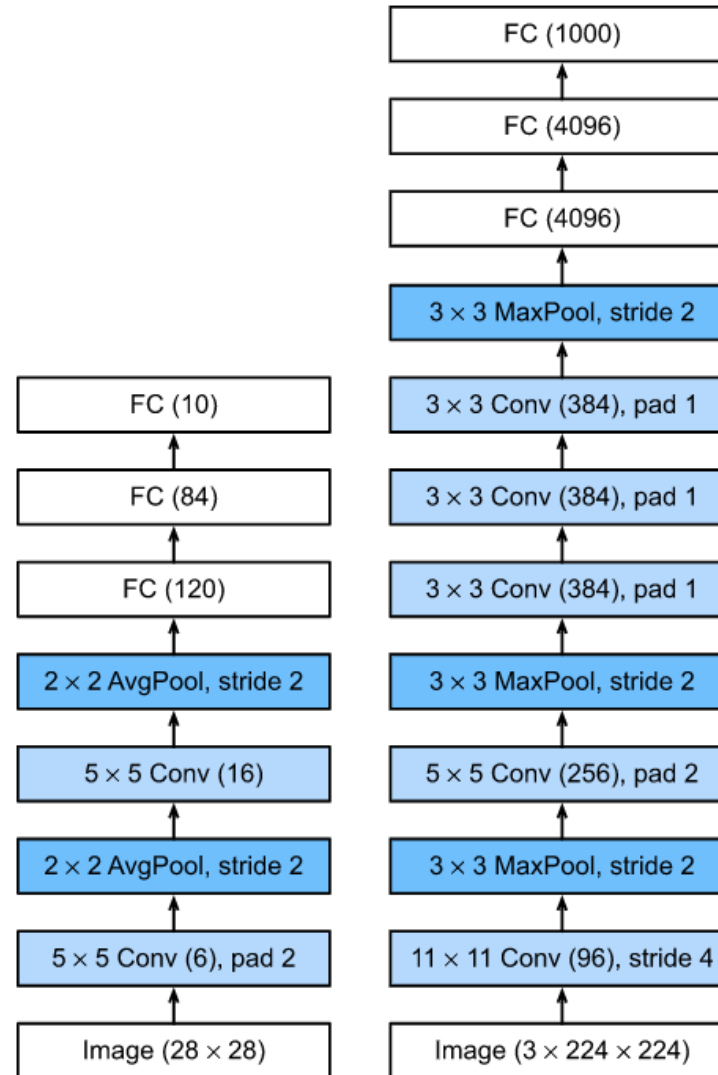
http://d2l.ai/chapter_convolutional-modern/alexnet.html

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet – 2012 (PyTorch)



http://d2l.ai/chapter_convolutional-modern/alexnet.html

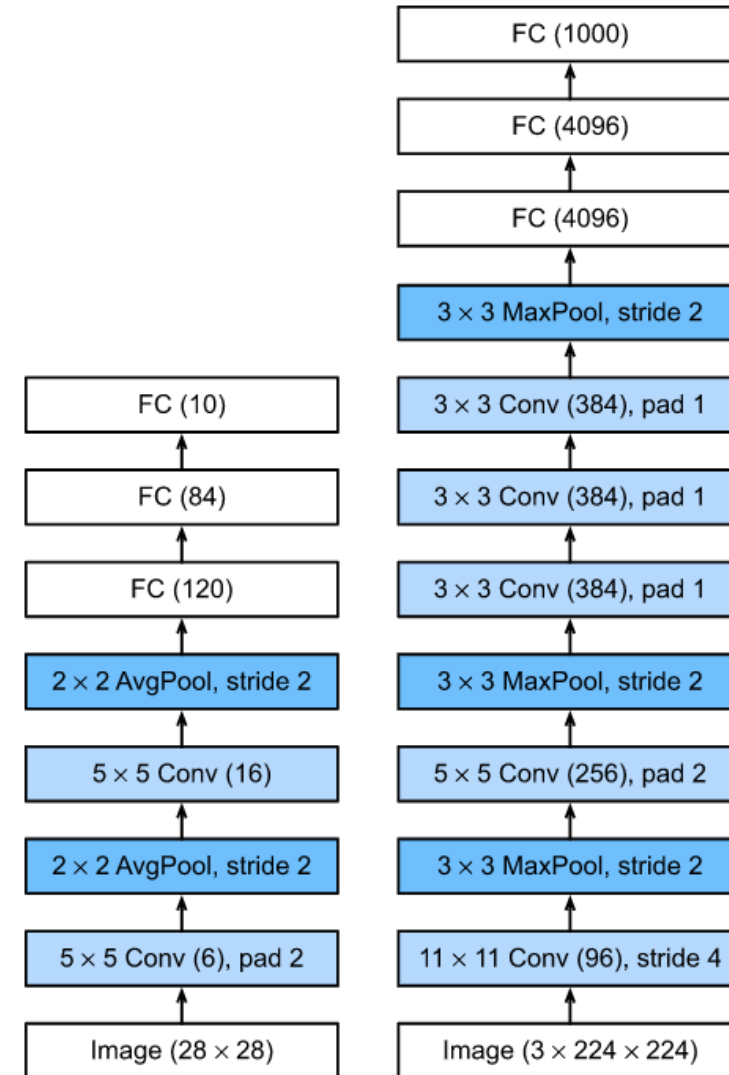
Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



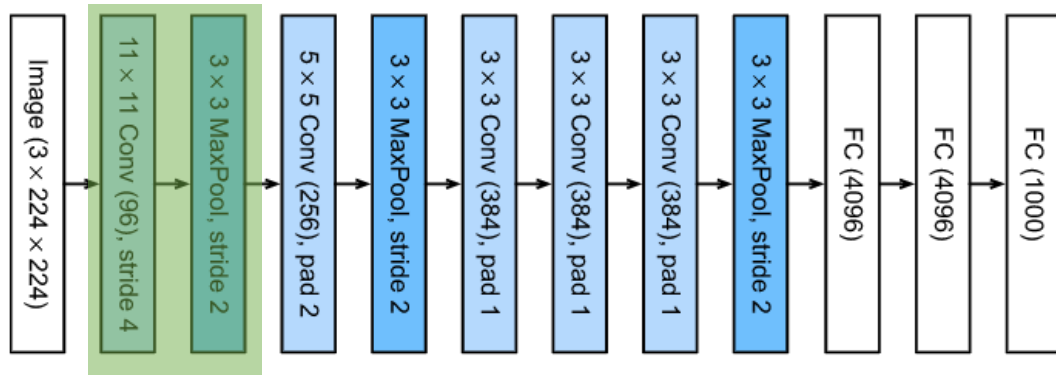
AlexNet – 2012 (PyTorch)

```
net = nn.Sequential(
    nn.Conv2d(3, 6, kernel_size=5, padding=0, stride=1),
    nn.ReLU(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5, padding=0, stride=1),
    nn.ReLU(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.LazyLinear(120),
    nn.ReLU(),
    nn.LazyLinear(84),
    nn.ReLU(),
    nn.LazyLinear(10),
    nn.ReLU(),
    nn.LazyLinear(84),
    nn.ReLU(),
    nn.LazyLinear(120),
    nn.ReLU(),
    nn.LazyLinear(1000),
    nn.ReLU(),
    nn.LazyLinear(4096),
    nn.ReLU(),
    nn.LazyLinear(4096),
    nn.ReLU(),
    nn.LazyLinear(1000)
)
```



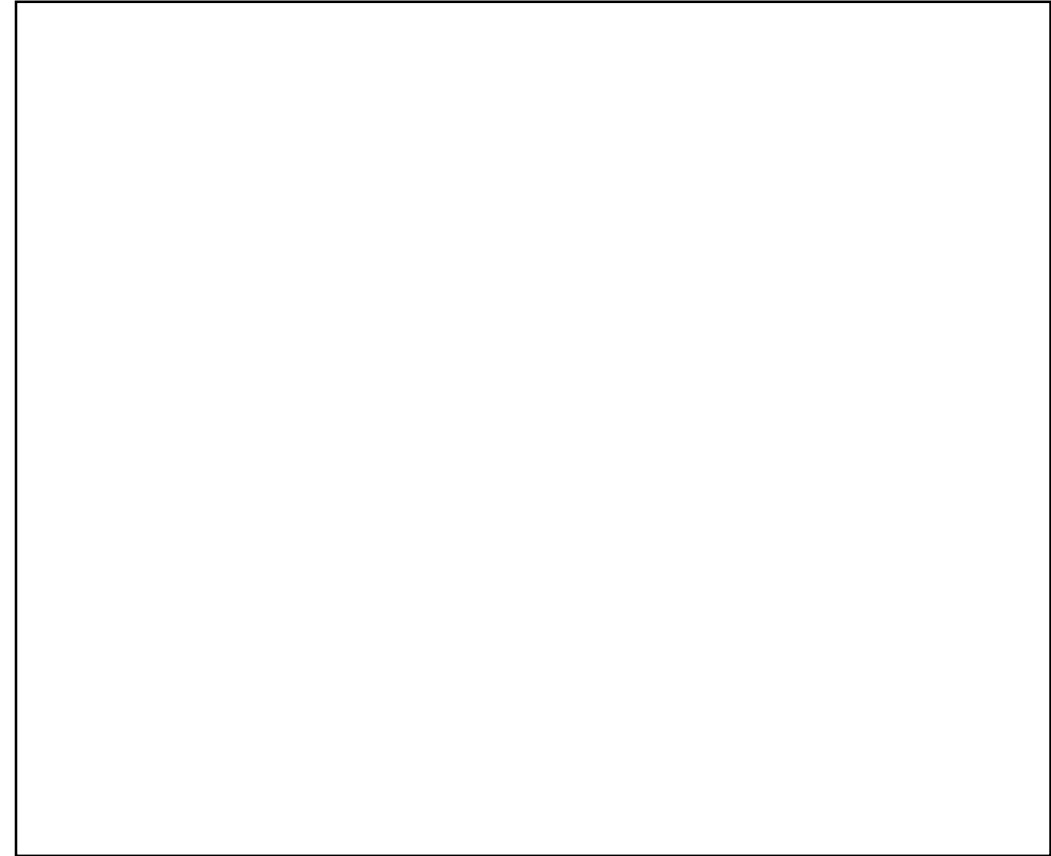


AlexNet – 2012 (PyTorch)



```
net = nn.Sequential(  

```

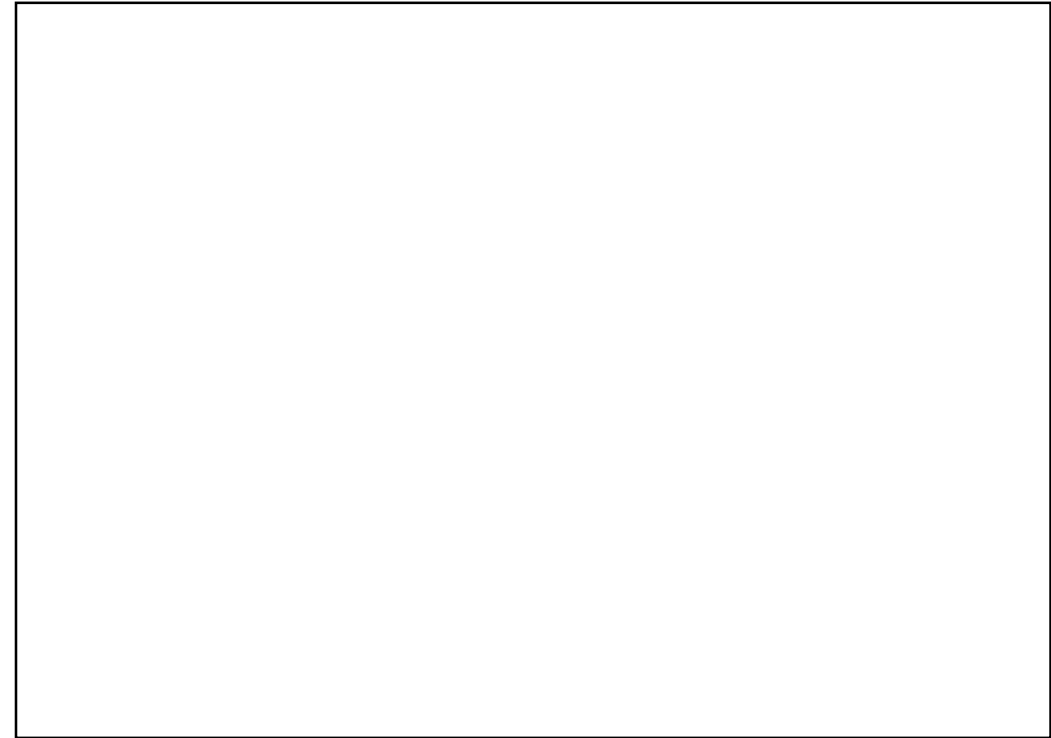
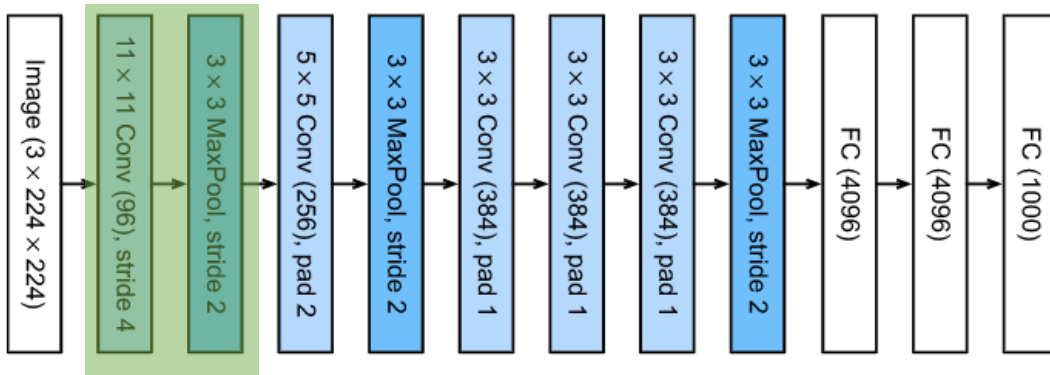


```
)
```



AlexNet – 2012 (PyTorch)

```
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
```

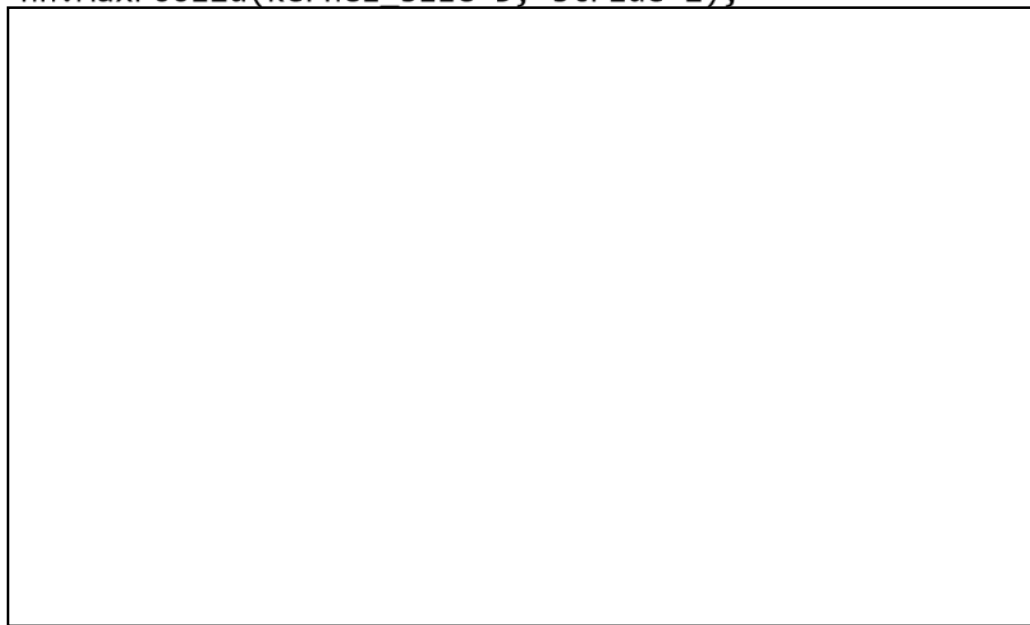
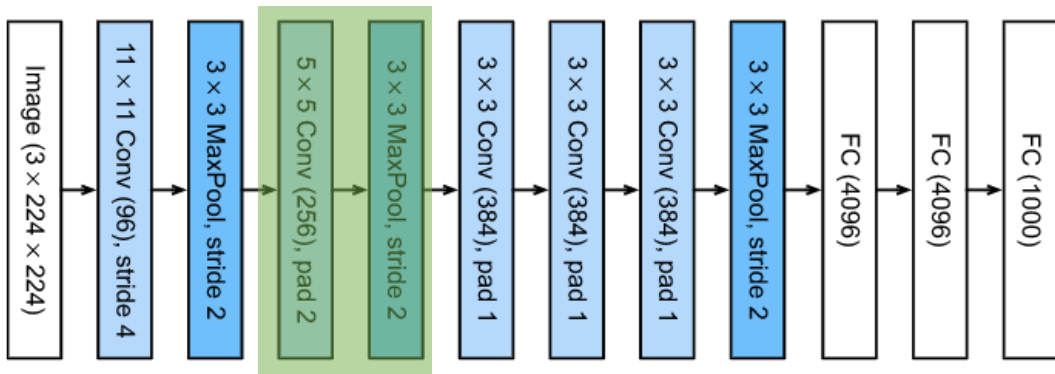


)



AlexNet – 2012 (PyTorch)

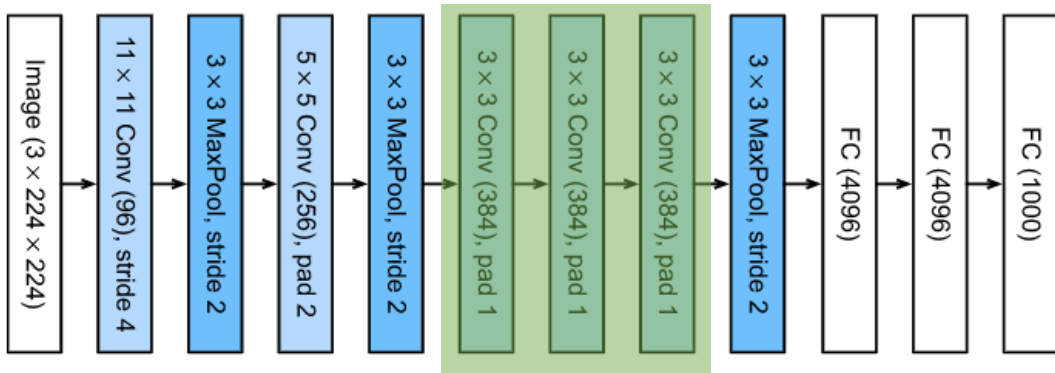
```
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(96, 256, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
```



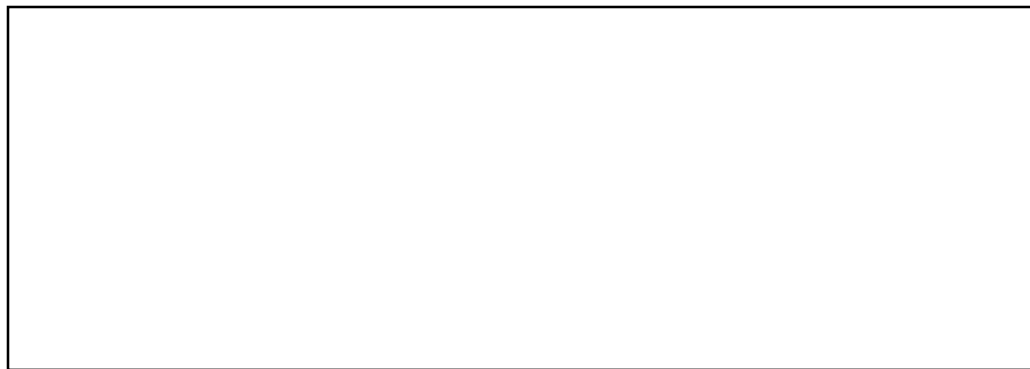
)



AlexNet – 2012 (PyTorch)



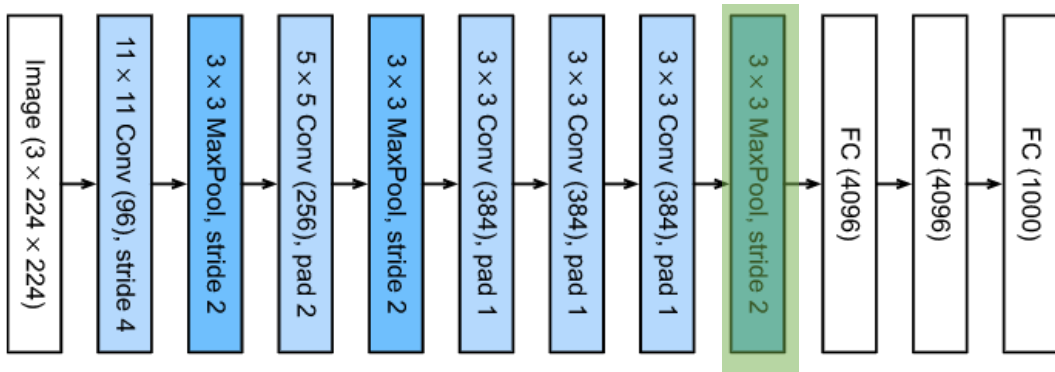
```
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(96, 256, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(256, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(),
```



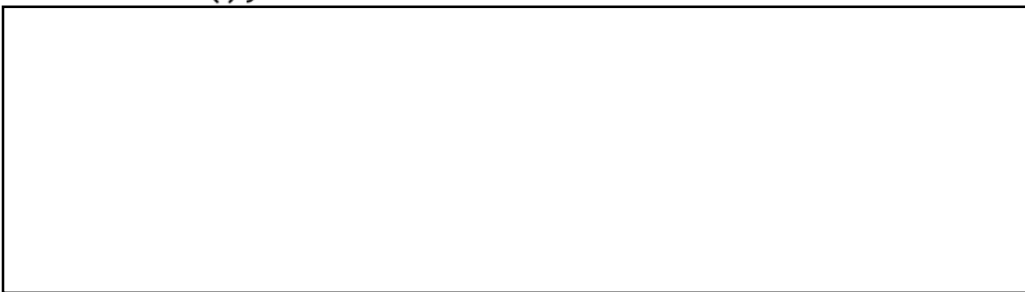
)



AlexNet – 2012 (PyTorch)



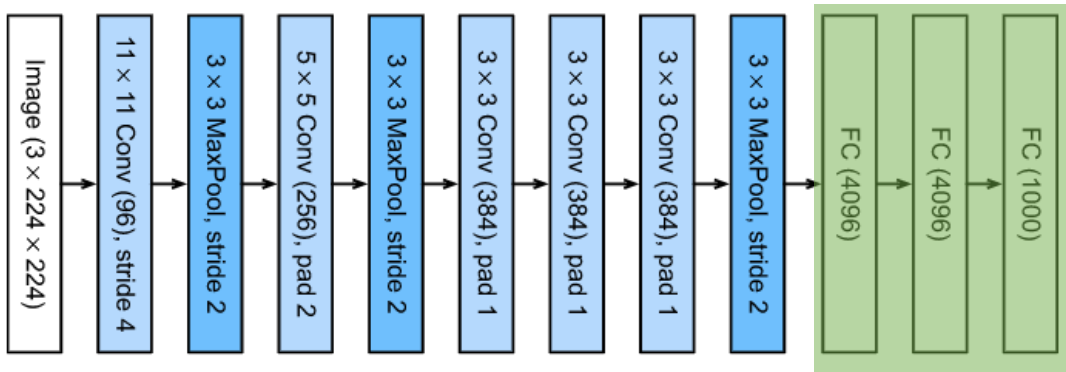
```
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(96, 256, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(256, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
```



)



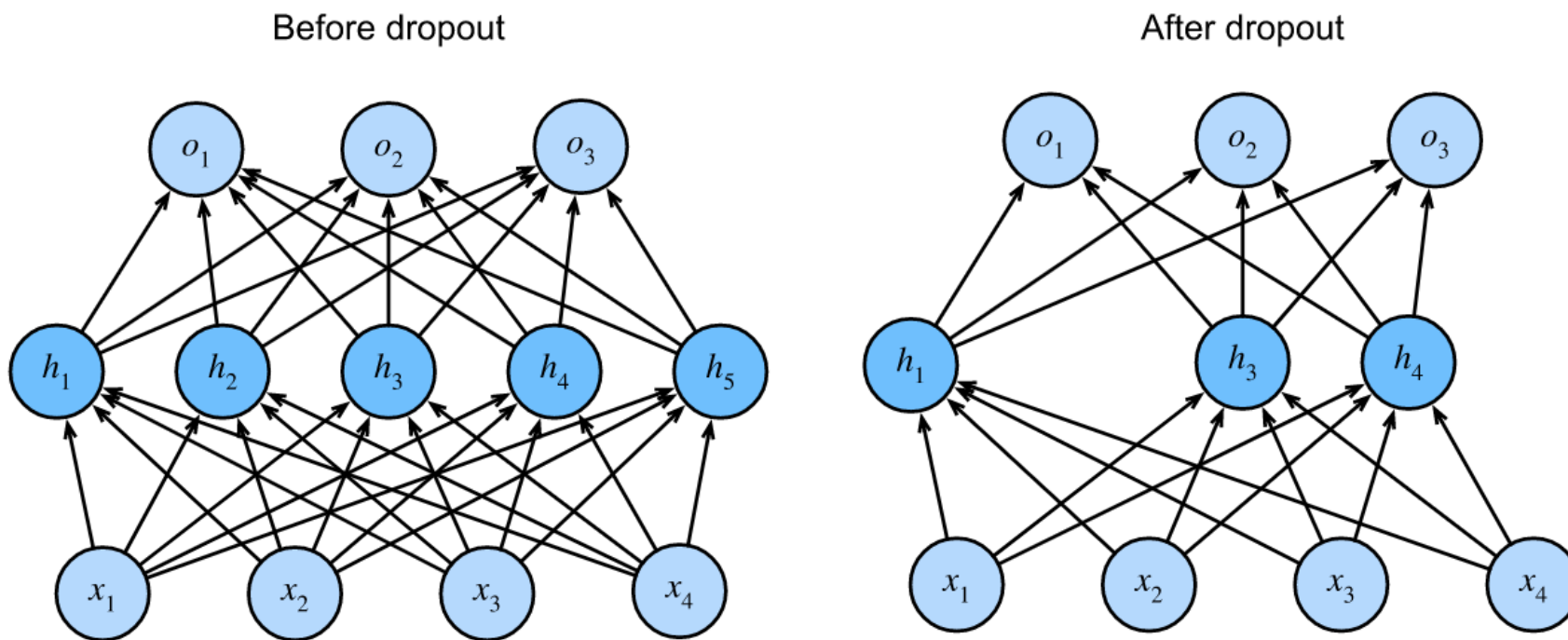
AlexNet – 2012 (PyTorch)



```
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(96, 256, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(256, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
    nn.Linear(6400, 4096),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 4096),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 2)
)
```

DROPOUT

- in some situations, the model is larger than we need – the model can be modified manually or using dropout technique
- **drop out** some neurons during training to avoid overfitting
- randomly **dropping out neurons**
 - neuron is dropped from the network with a probability of 0.5



Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov: Improving neural networks by preventing co-adaptation of feature detectors. CoRR abs/1207.0580 (2012)

<https://www.learnopencv.com/understanding-alexnet/>

http://d2l.ai/chapter_multilayer-perceptrons/dropout.html#sec-dropout

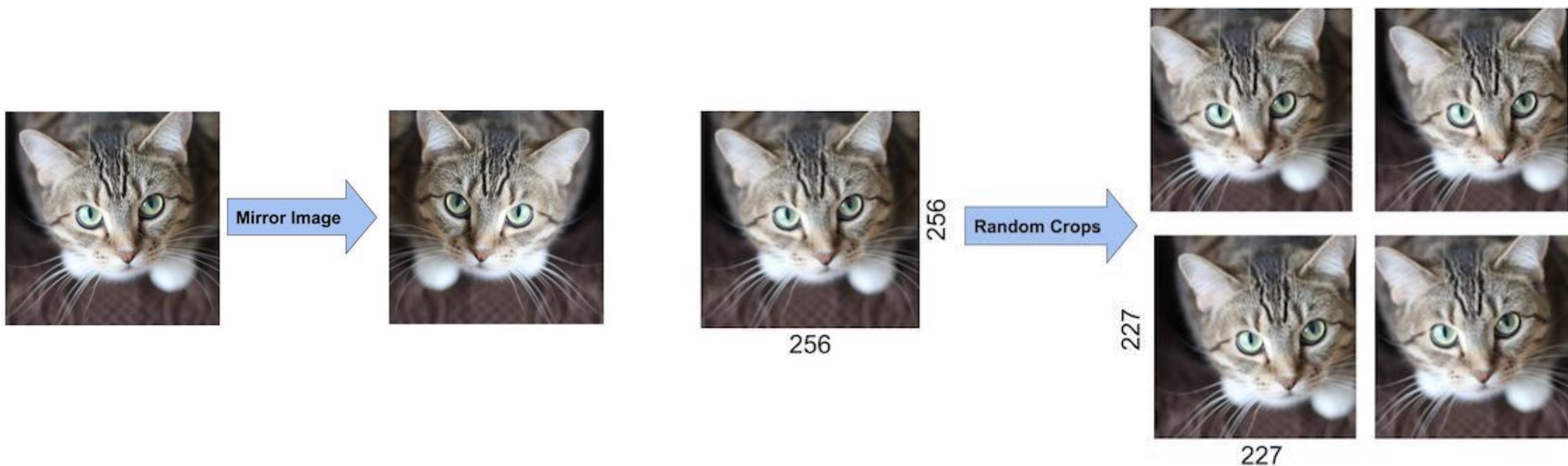
Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



AlexNet - 2012

Data Augmentation



<https://www.learnopencv.com/understanding-alexnet/>

http://d2l.ai/chapter_multilayer-perceptrons/dropout.html#sec-dropout

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.

AlexNet - 2012

ReLU (Rectified Linear Unit)

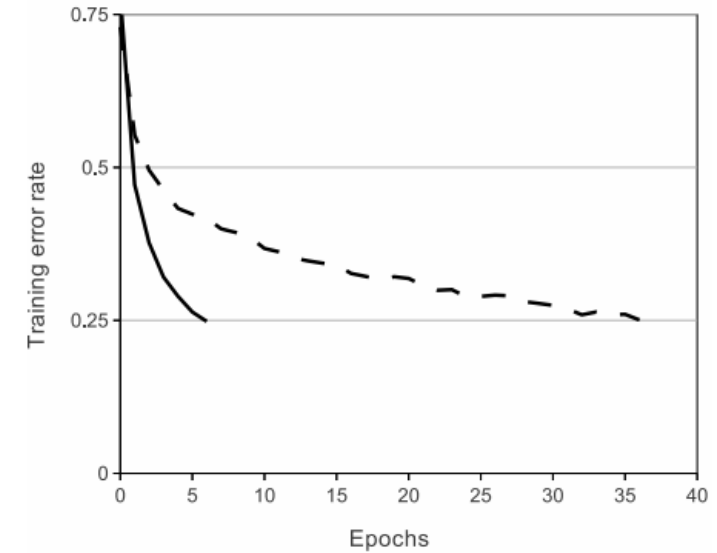
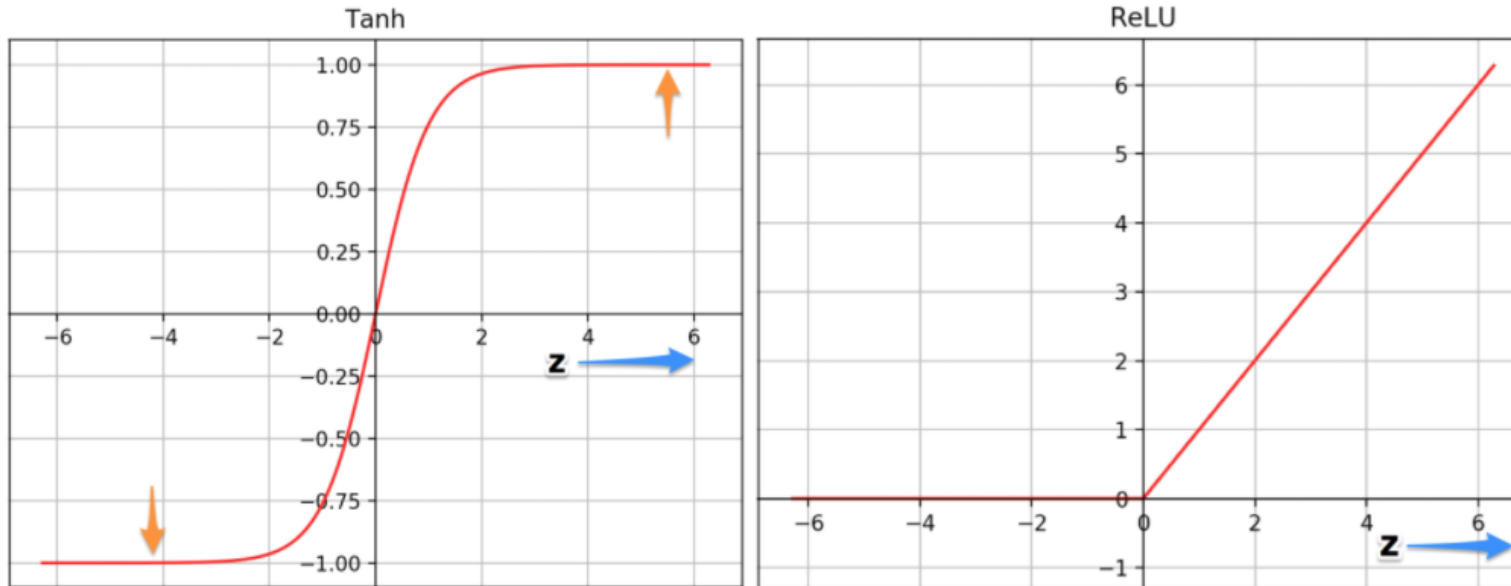


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

<https://www.learnopencv.com/understanding-alexnet/>
http://d2l.ai/chapter_multilayer-perceptrons/dropout.html#sec-dropout

Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25.

10.1145/3065386.



CovNet Architectures

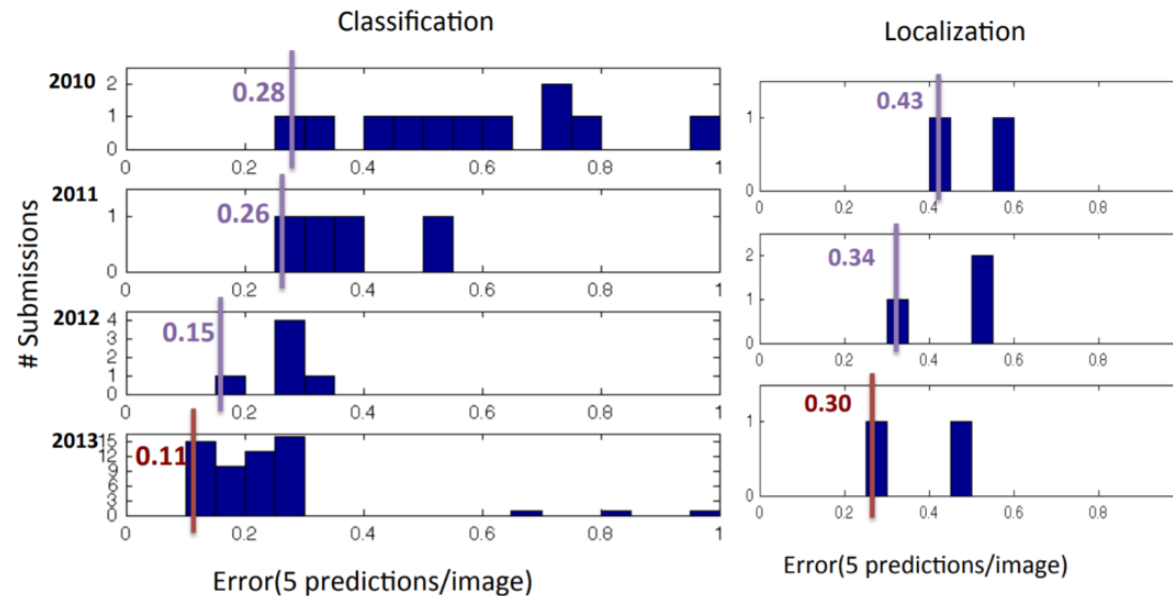
- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**



2013

- <http://image-net.org/challenges/LSVRC/2013/>
- http://www.image-net.org/challenges/LSVRC/2013/slides/ILSVRC2013_12_7_13_clsloc.pdf

ILSVRC over the years



- Similar architecture to AlexNet
- 11x11 vs. 7x7 filters in the first layer
- Number of filters is increasing
- Visualization of features - Deconvolutional Network

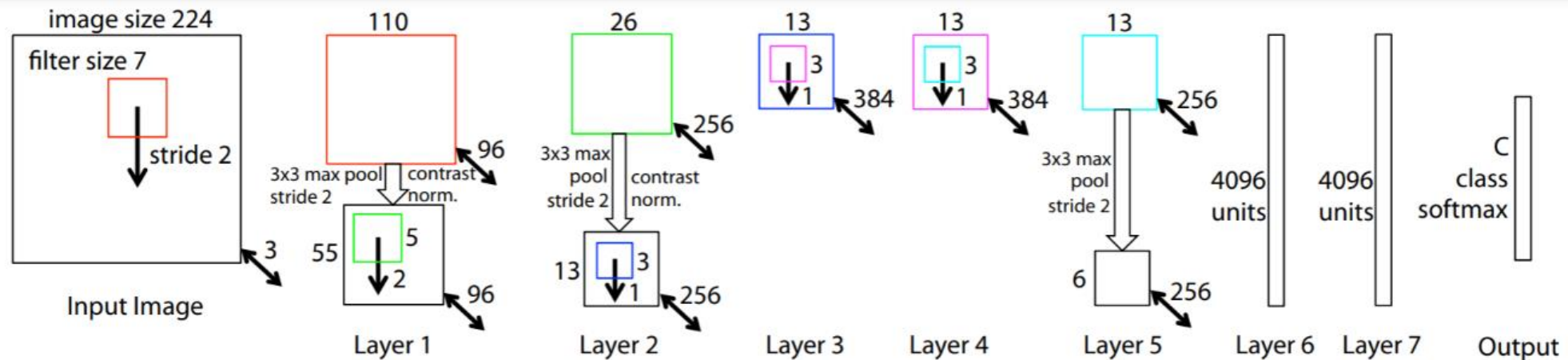
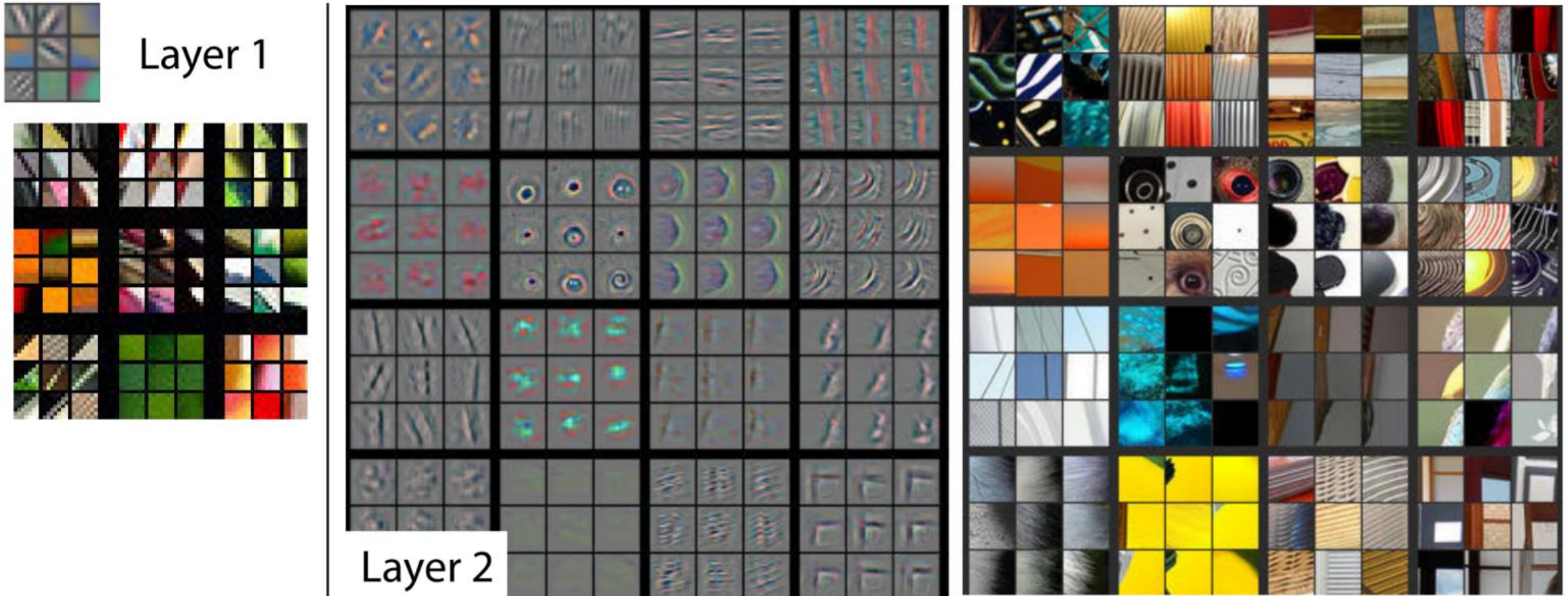


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

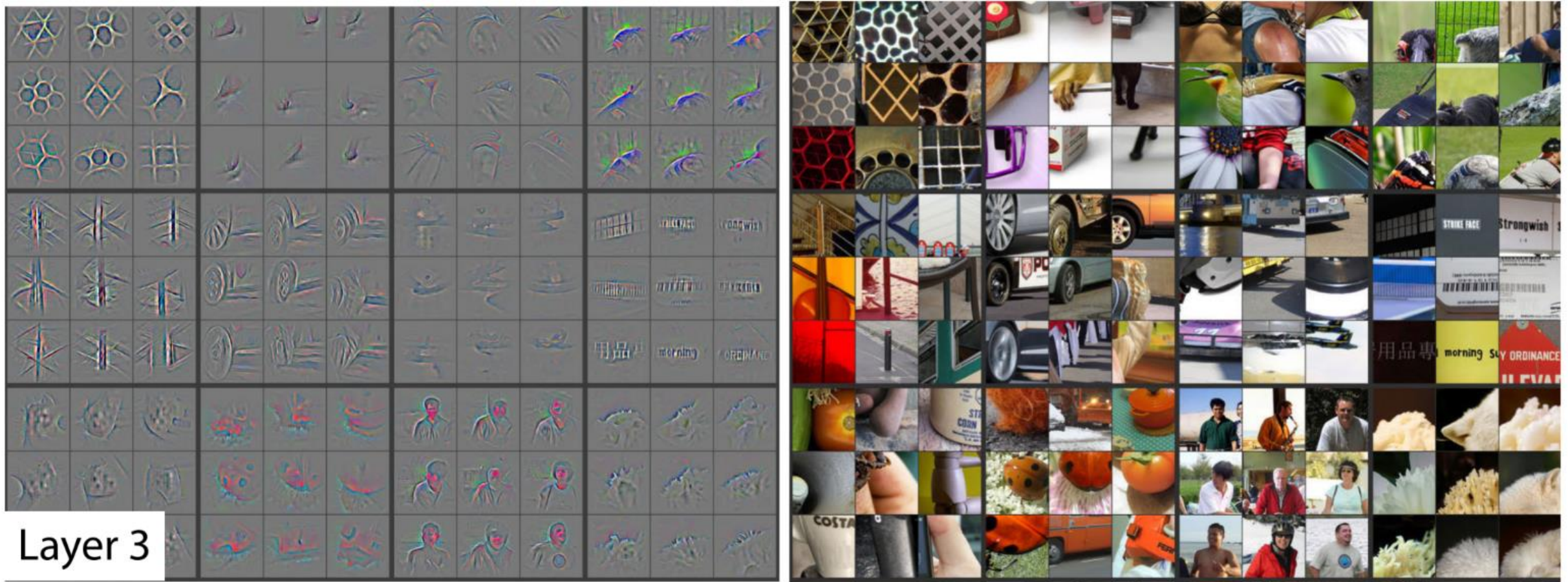


First Layers - low level feature detector (edge detector, circular features)





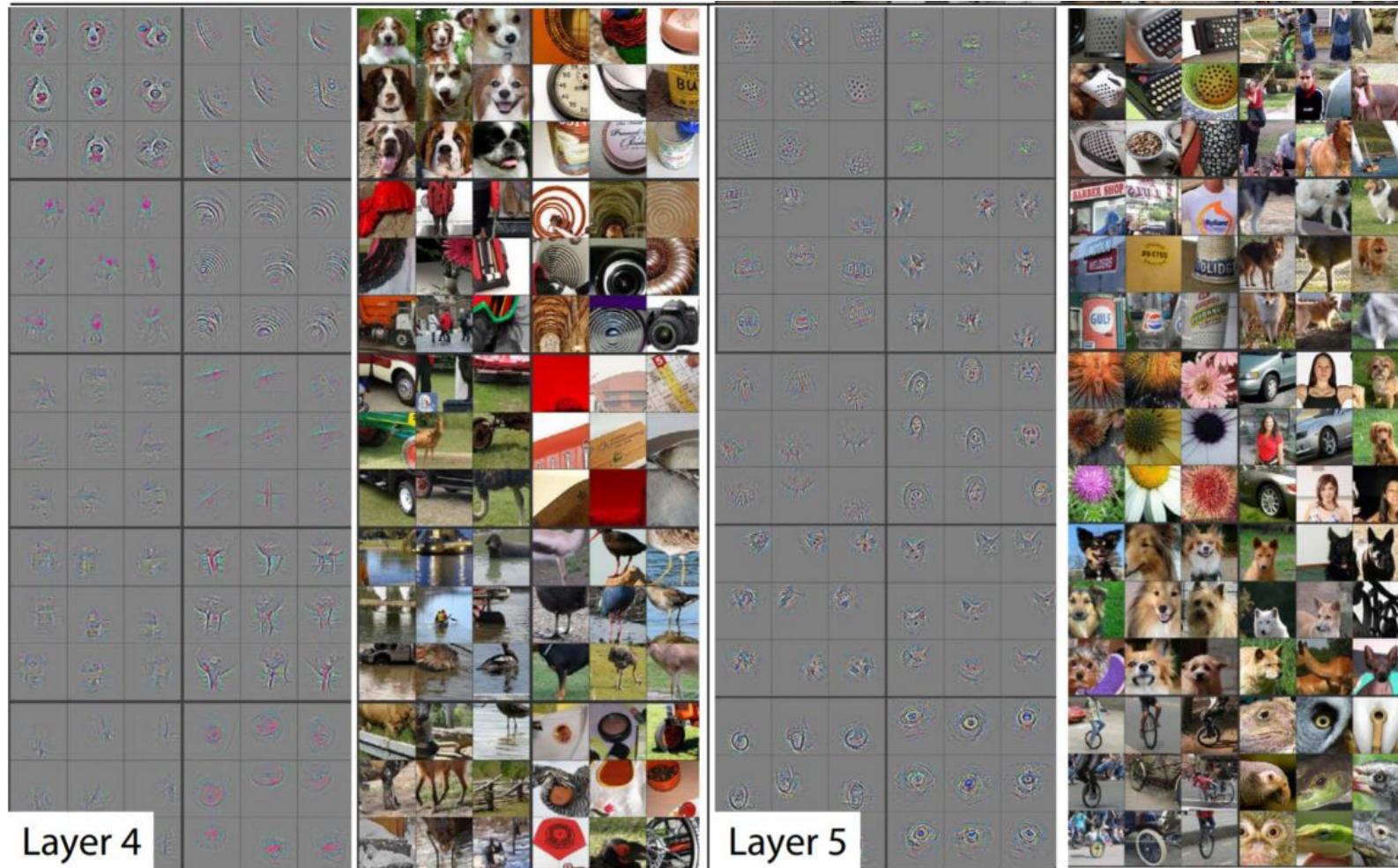
Deep Layers - higher level features such as dogs faces or flowers



Layer 3



Deep Layers - higher level features such as dogs faces or flowers

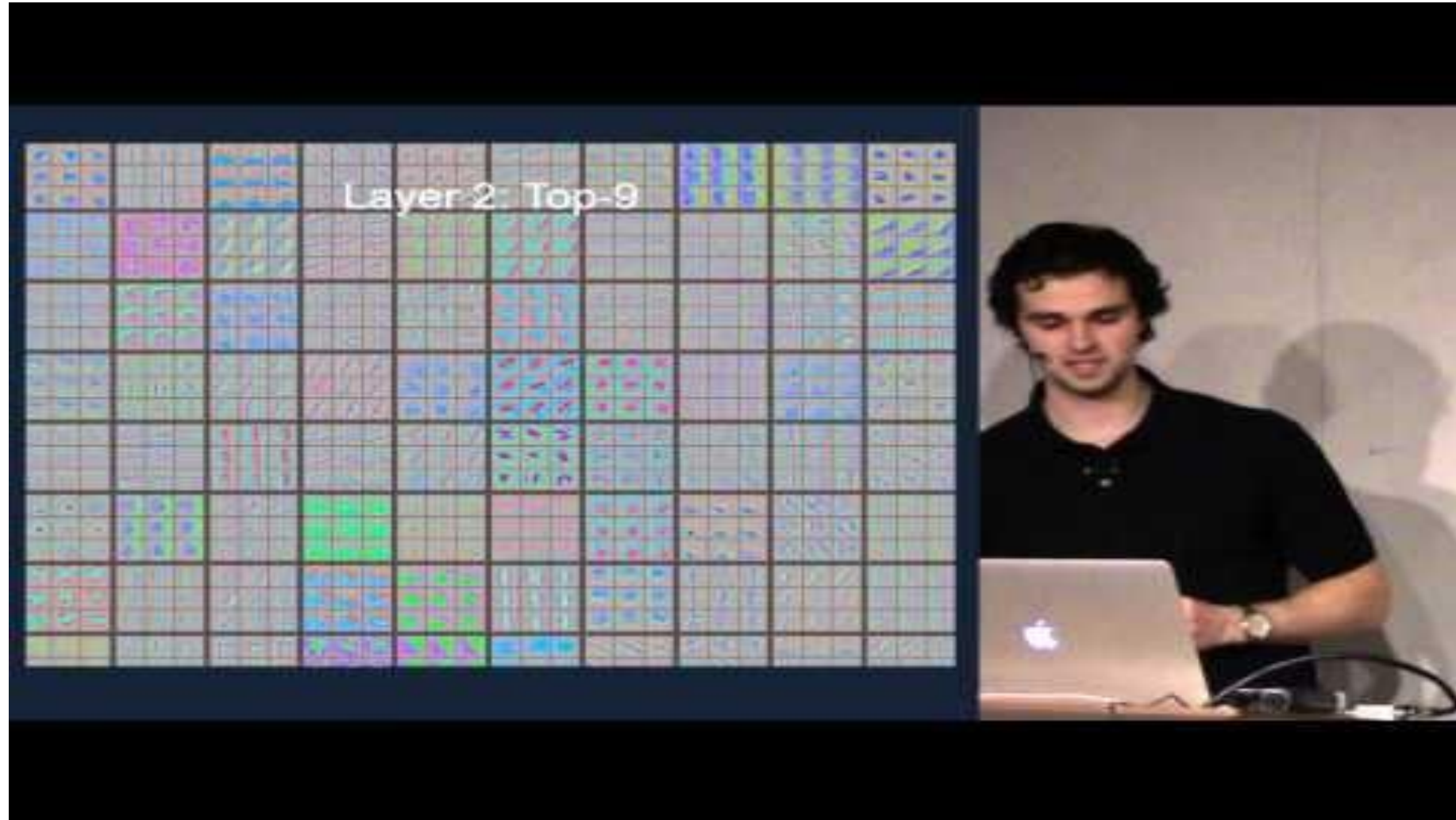


<https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

Matthew D. Zeiler, Rob Fergus: Visualizing and Understanding Convolutional Networks



<https://www.youtube.com/watch?reload=9&v=ghEmQSxT6tw>



<https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>

Matthew D. Zeiler, Rob Fergus: Visualizing and Understanding Convolutional Networks



CovNet Architectures

- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**



2014

- ImageNet Challenge <http://www.image-net.org/challenges/LSVRC/2014/>
- <http://www.image-net.org/challenges/LSVRC/2014/results>
- **Idea:**
 - stack the convolutional layers with increasing filter sizes
 - 3 x 3 filter size stride of 1
 - MaxPooling 2 x 2 stride of 2
 - **VGG Block**
 - Dropout, MaxPooling, ReLu
 - On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2–3 weeks depending on the architecture

http://d2l.ai/chapter_convolutional-modern/vgg.html <https://arxiv.org/pdf/1409.1556v6.pdf>

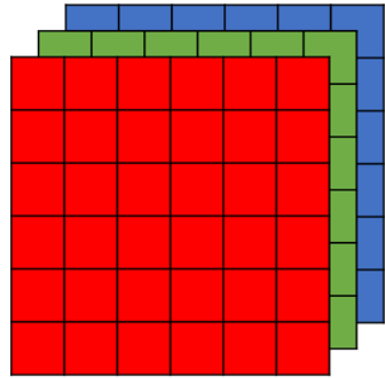
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



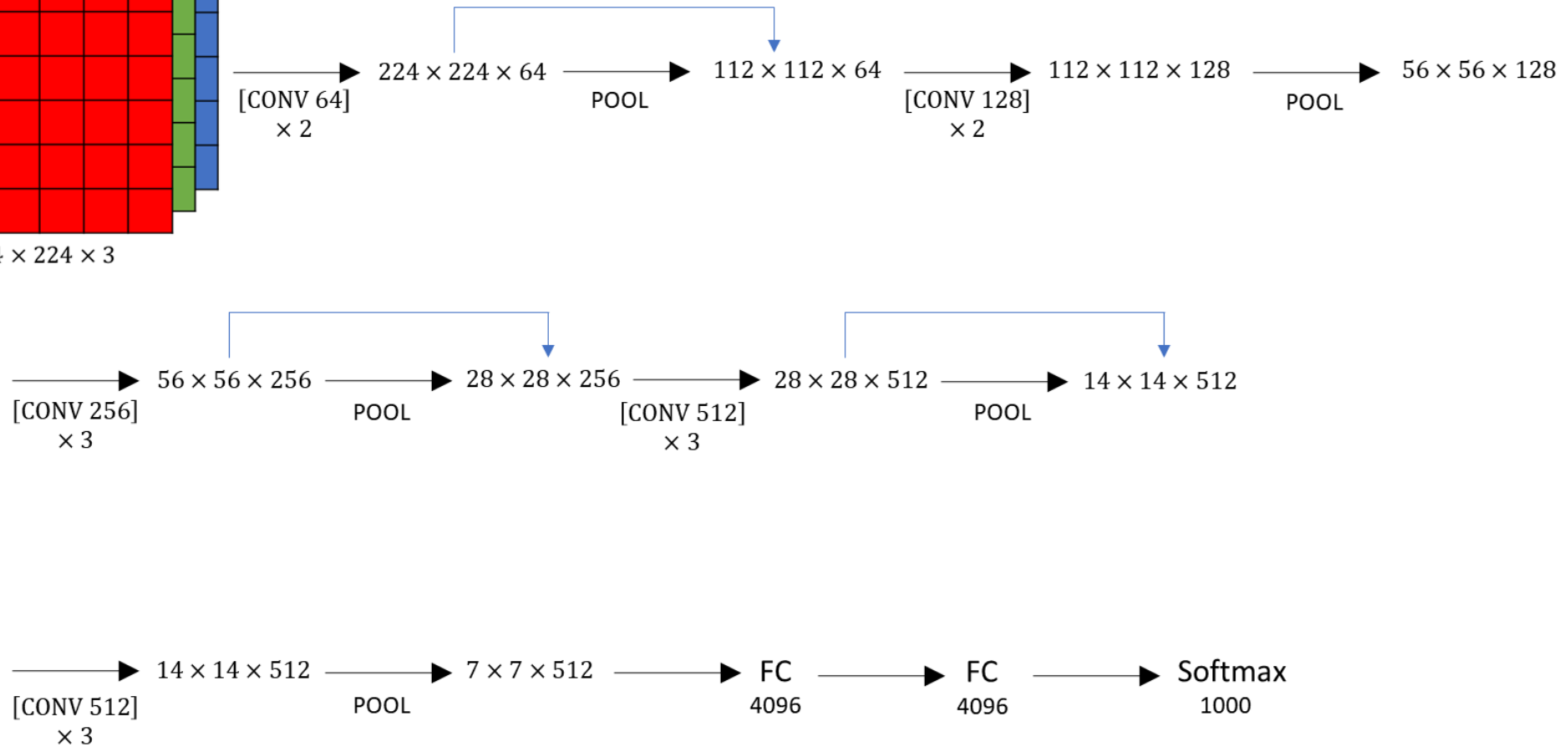
VGG - 2014

CONV = 3 x 3 filter, s=1, same

MAX-POOL = 2 x 2, s=2



224 x 224 x 3





VGG - 2014

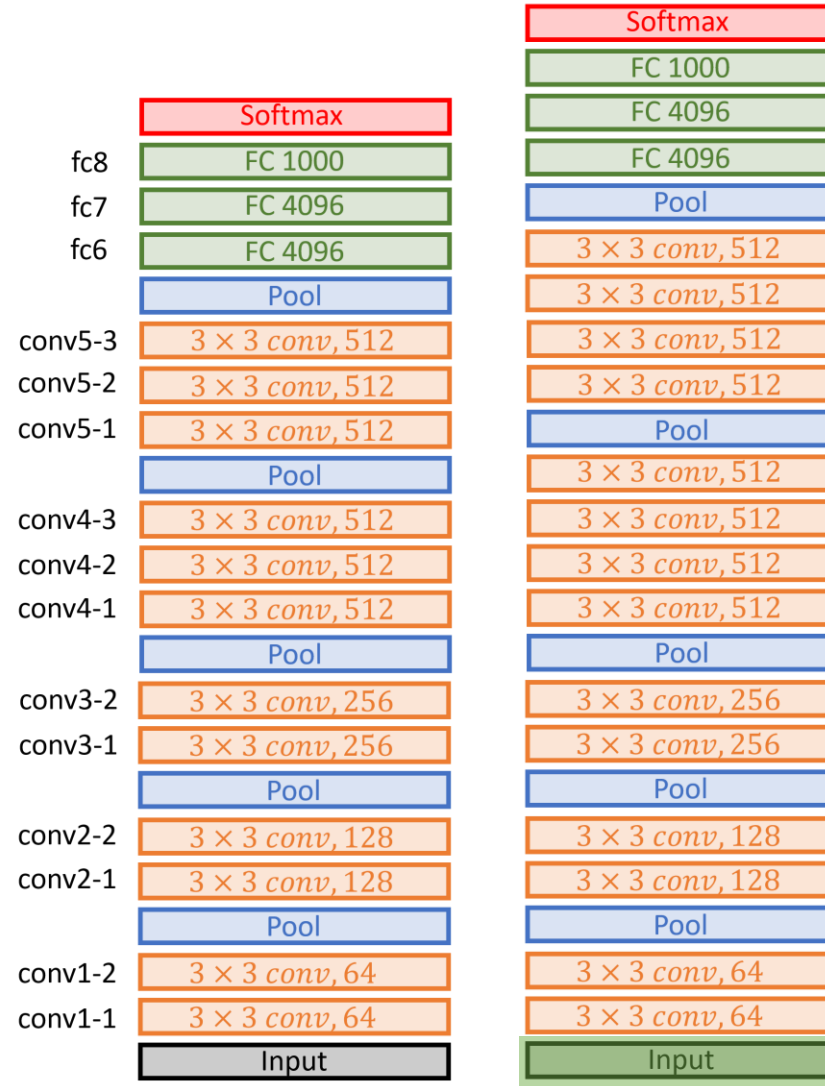
6 VGGNet Architectures

Architecture	Number of Parameters (millions)	Top-5 Error Rate (%)
<p>VGG-11</p>	133	10.4
<p>VGG-11 (LRN)</p>	133	10.5
<p>VGG-13</p>	133	9.9
<p>VGG-16 (Conv1)</p>	134	9.4
<p>VGG-16</p>	138	8.8
<p>VGG-19</p>	144	9.0

<https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

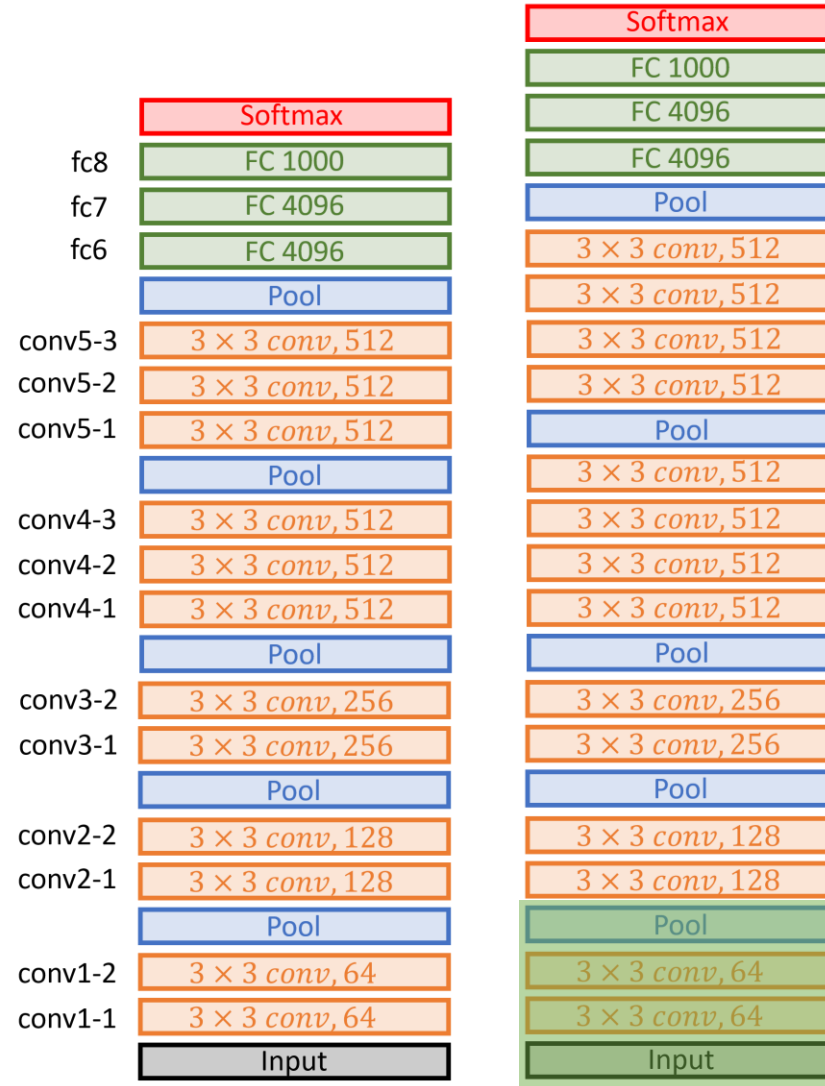
VGG19

input<matrix<unsigned char>>

<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

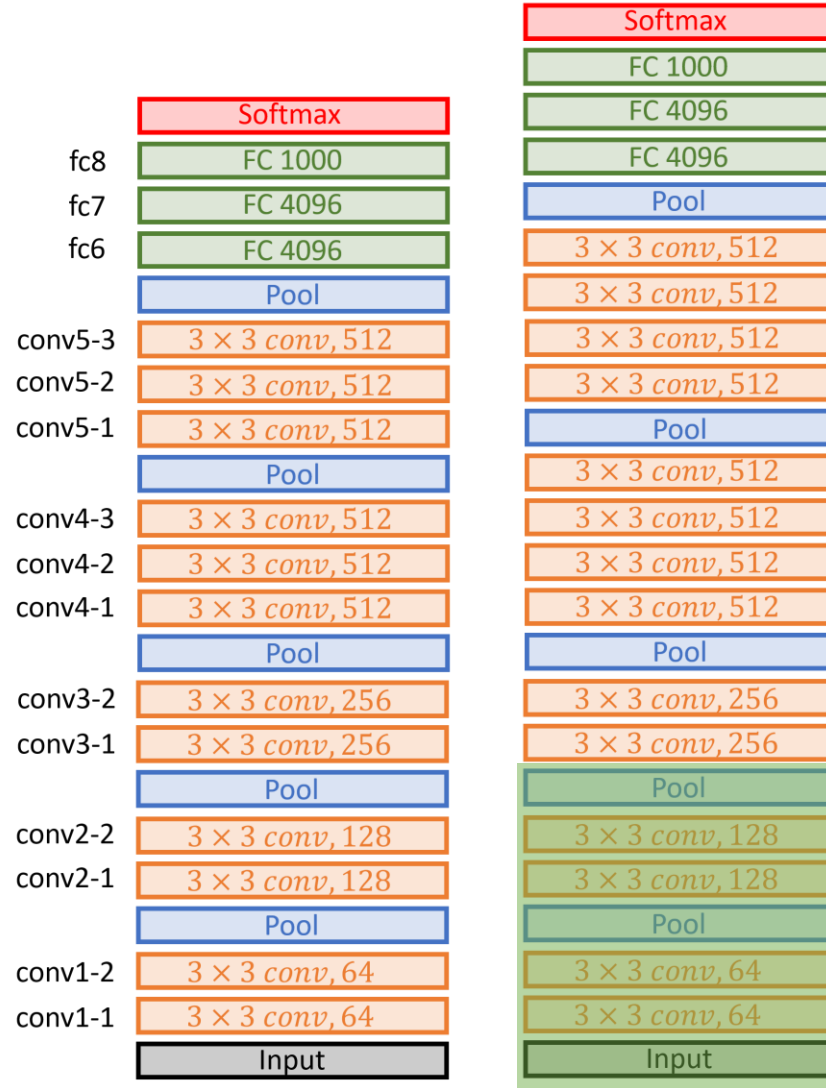
VGG19

```
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
```

<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

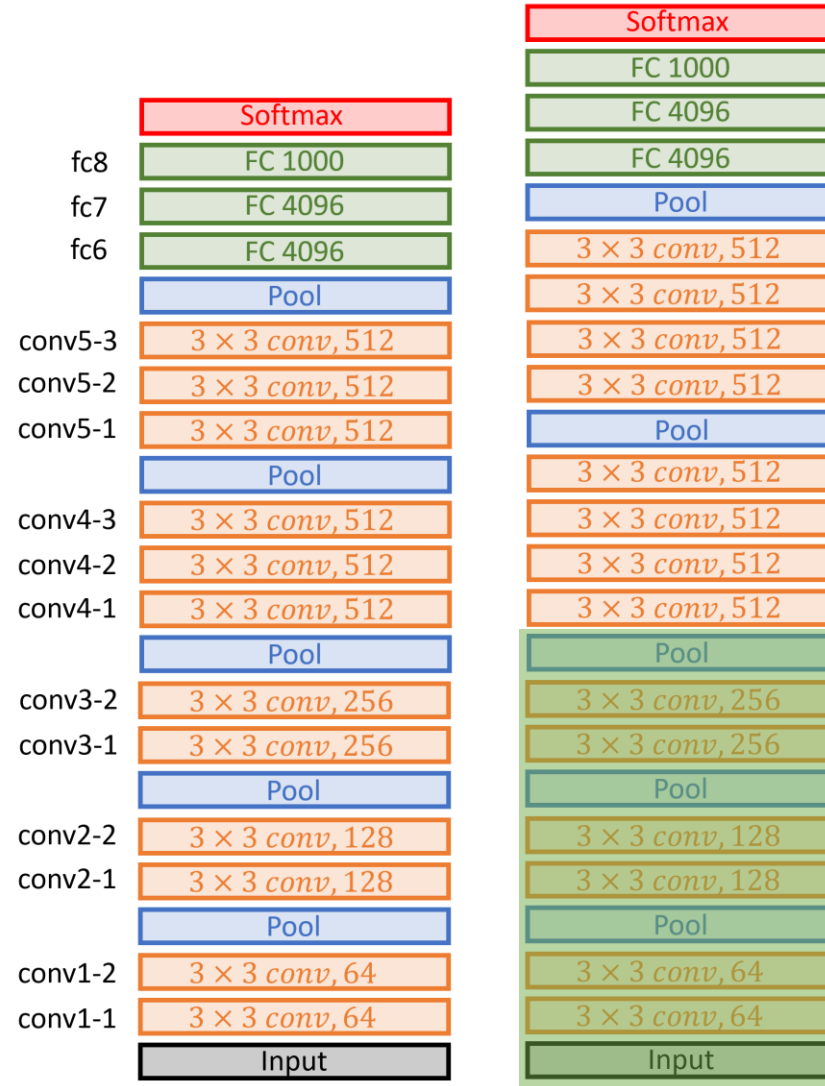
VGG19

```
max_pool<2,2,2,2,
relu<con<128,3,3,1,1,
relu<con<128,3,3,1,1,
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
```

<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

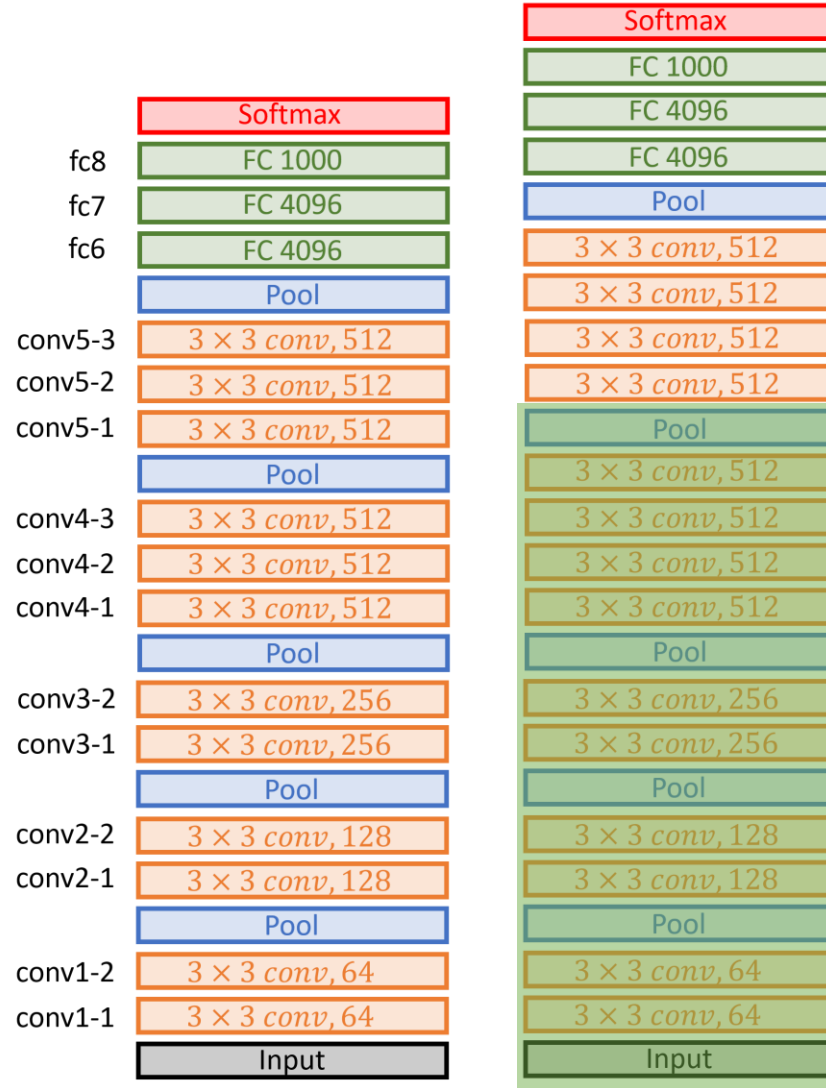
VGG19

```
max_pool<2,2,2,2,
relu<con<256,3,3,1,1,
relu<con<256,3,3,1,1,
max_pool<2,2,2,2,
relu<con<128,3,3,1,1,
relu<con<128,3,3,1,1,
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
```

<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

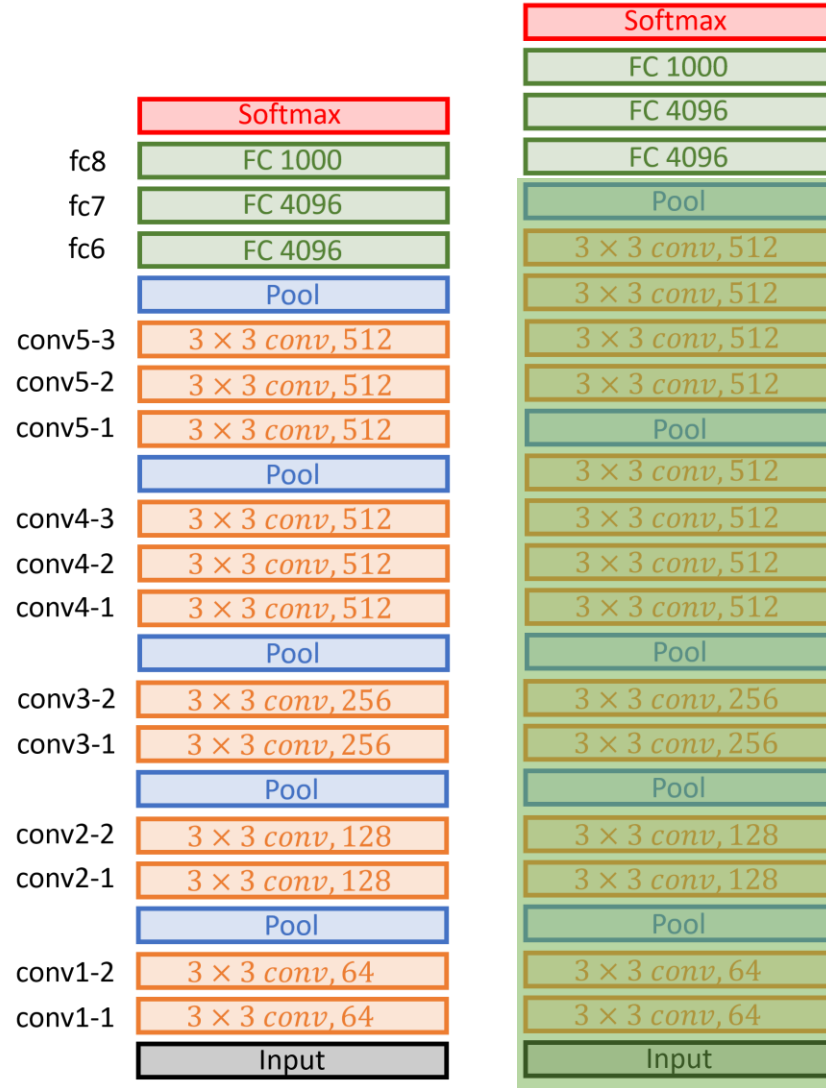
VGG19

```
max_pool<2,2,2,2,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
max_pool<2,2,2,2,
relu<con<256,3,3,1,1,
relu<con<256,3,3,1,1,
max_pool<2,2,2,2,
relu<con<128,3,3,1,1,
relu<con<128,3,3,1,1,
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
```

<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16

VGG19

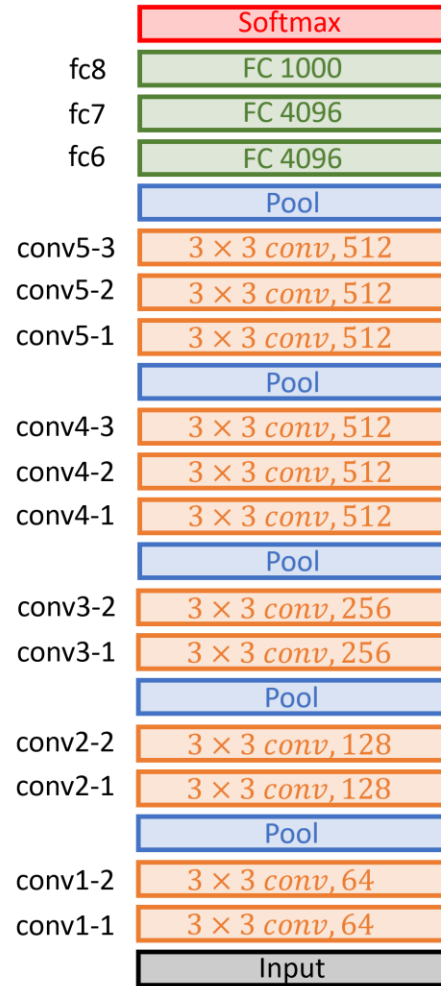
```

max_pool<2,2,2,2,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
max_pool<2,2,2,2,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
max_pool<2,2,2,2,
relu<con<256,3,3,1,1,
relu<con<256,3,3,1,1,
max_pool<2,2,2,2,
relu<con<128,3,3,1,1,
relu<con<128,3,3,1,1,
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
  
```

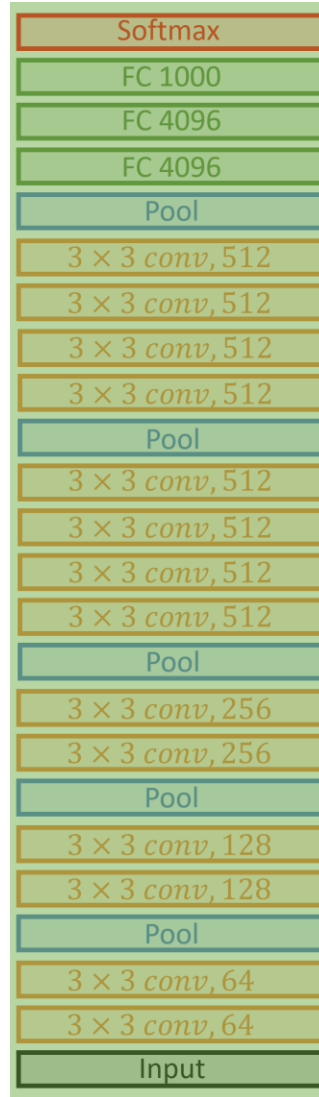
<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG16



VGG19

```
fc<1000,
dropout<relu<fc<4096,
dropout<relu<fc<4096,
max_pool<2,2,2,2,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
max_pool<2,2,2,2,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
relu<con<512,3,3,1,1,
max_pool<2,2,2,2,
relu<con<256,3,3,1,1,
relu<con<256,3,3,1,1,
max_pool<2,2,2,2,
relu<con<128,3,3,1,1,
relu<con<128,3,3,1,1,
max_pool<2,2,2,2,
relu<con<64,3,3,1,1,
relu<con<64,3,3,1,1,
input<matrix<unsigned char>>
```

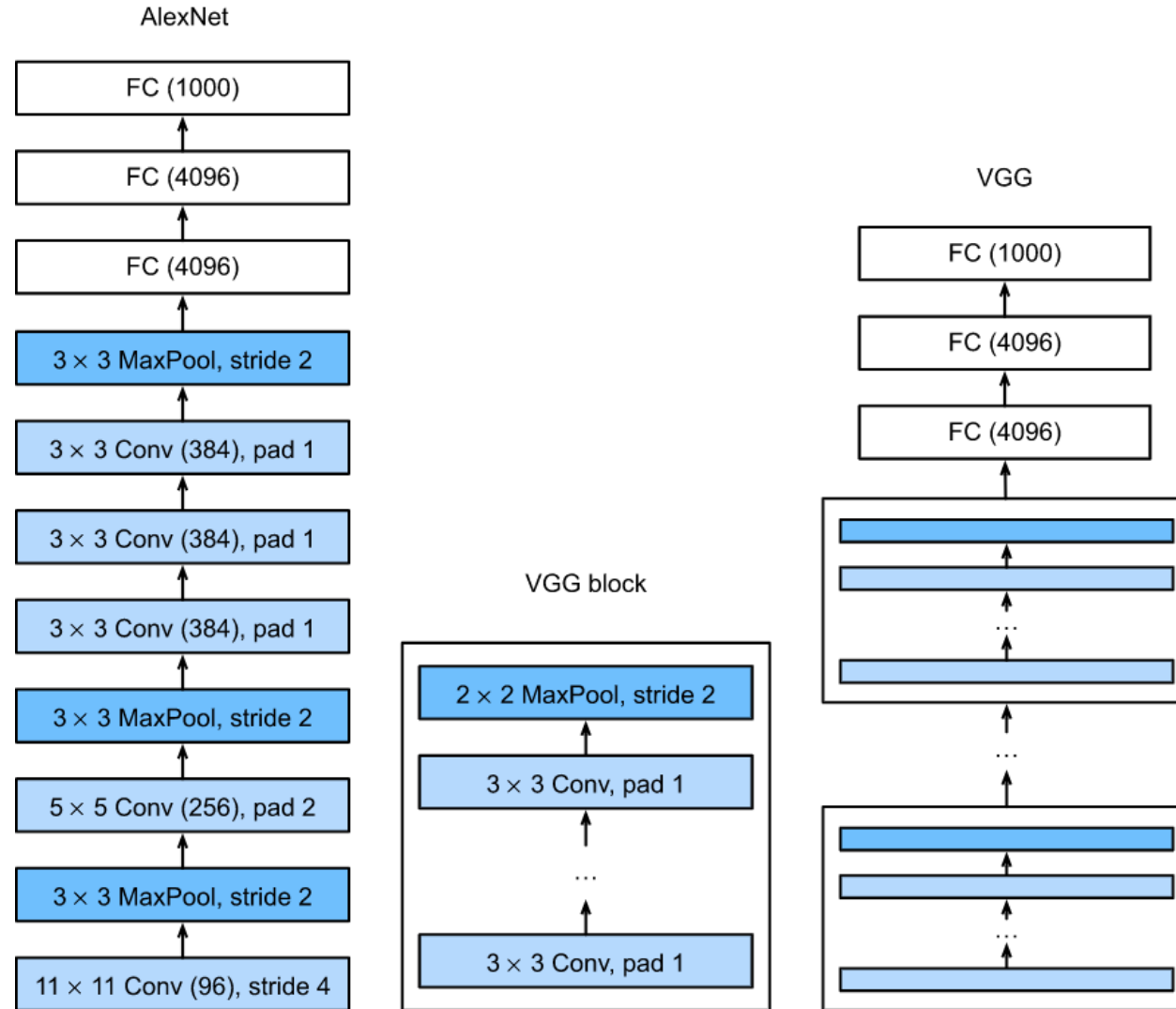
<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG - 2014



<http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/>

http://d2l.ai/chapter_convolutional-modern/vgg.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.



VGG – 2014 Pytorch

```

VGGNet = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(256, 512, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(4096),
    nn.ReLU(),
    nn.Dropout2d(0.5),
    nn.Linear(4096),
    nn.ReLU(),
    nn.Dropout2d(0.5),
    nn.Linear(2)
)

```

									Number of Parameters (millions)	Top-5 Error Rate (%)																		
	Image	Conv3-64	Max pool	Conv3-128	Max pool	Conv3-256	Conv3-256	Max pool	Conv3-512	Conv3-512	Max pool	Conv3-512	Conv3-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	133	10.4								
	VGG-11																											
	Image	Conv3-64	LRN	Max pool	Conv3-128	Max pool	Conv3-256	Conv3-256	Max pool	Conv3-512	Conv3-512	Max pool	Conv3-512	Conv3-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	133	10.5							
	VGG-11 (LRN)																											
	Image	Conv3-64	Conv3-64	Max pool	Conv3-128	Conv3-128	Max pool	Conv3-256	Conv3-256	Max pool	Conv3-512	Conv3-512	Max pool	Conv3-512	Conv3-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	133	9.9						
	VGG-13																											
	Image	Conv3-64	Conv3-64	Max pool	Conv3-128	Conv3-128	Max pool	Conv3-256	Conv3-256	Conv1-256	Max pool	Conv3-512	Conv3-512	Conv1-512	Max pool	Conv3-512	Conv3-512	Conv1-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	134	9.4			
	VGG-16 (Conv1)																											
	Image	Conv3-64	Conv3-64	Max pool	Conv3-128	Conv3-128	Max pool	Conv3-256	Conv3-256	Conv3-256	Max pool	Conv3-512	Conv3-512	Conv3-512	Max pool	Conv3-512	Conv3-512	Conv3-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	138	8.8			
	VGG-16																											
	Image	Conv3-64	Conv3-64	Max pool	Conv3-128	Conv3-128	Max pool	Conv3-256	Conv3-256	Conv3-256	Conv3-256	Max pool	Conv3-512	Conv3-512	Conv3-512	Conv3-512	Max pool	Conv3-512	Conv3-512	Conv3-512	Conv3-512	Max pool	FC-4096	FC-4096	FC-1000	Soft-max	144	9.0
	VGG-19																											



CovNet Architectures

- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**



GoogLeNet - 2014

2014

- ImageNet Challenge <http://www.image-net.org/challenges/LSVRC/2014/>
- <http://www.image-net.org/challenges/LSVRC/2014/results>
- **GoogLeNet > Google/LeNet**
- **1 x 1 convolution**
- Network In Network
- **Variously-sized kernels**
- **Inception Blocks**



GoogLeNet - 2014



všechny plakáty (98)

Počátek



Inception
Inception

(další názvy)

Thriller / Mysteriózní / Akční / Sci-Fi / Dobrodružný
USA / Velká Británie, 2010, 148 min

Režie: Christopher Nolan

Scénář: Christopher Nolan

Kamera: Wally Pfister

Hudba: Hans Zimmer

Hrají: Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Tom Hardy, Ken Watanabe, Dileep Rao, Cillian Murphy, Tom Berenger, Marion Cotillard, Pete Postlethwaite, Michael Caine, Lukas Haas, Tai-Li Lee, Claire Geare, Taylor Geare, Johnathan Geare, Tohoru Masamune, Júdži Okumoto, Earl Cameron, Ryan Hayward, Tim Kelleher, Talulah Riley, S... (více)

(další profese)





GoogLeNet - 2014



inception we need to go deeper



Q Vše [Obrázky](#) [Videa](#) [Nákupy](#) [Zprávy](#) [Více](#) [Nastavení](#) [Nástroje](#)

gif

meme

leonardo dicaprio

di caprio

neural networks

leo dicaprio

deep

inception meme generator

deep learning

machine learning



We Need To Go Deeper | Know Your Meme
knowyourmeme.com



We Need To Go Deeper Inception GIFs | Tenor
tenor.com



That's not enough We have to go deeper - Incep...
quickmeme.com



Beta inception we need to go deeper ...
memegenerator.net



Create meme "We need to go deep...
meme-arsenal.com



Inception - We need to ...
9gag.com



we need to go deeper - I...
memegenerator.net



we-need-to-go-deeper-yoga-leonardo-di...
pinterest.com



We Need To Go Deeper | ...
knowyourmeme.com



This conversation is ge...
quickmeme.com



Inception still slaps 10 ...
vox.com



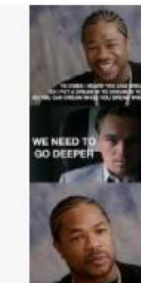
A Simple Guide to the Versions of the Inception Netw...
towardsdatascience.com



go deeper - Imgflip
imgflip.com



We need to go deeper... | Inception | Know Y...
knowyourmeme.com



We need to go deep
theshiznit.co.uk



GoogLeNet - 2014

- **Deep Network > overfitting**
- **What kernel sizes is right?**
 - **3 x 3 (VGGNet)**
 - **5 x 5 (LeNet)**
 - **7 x 7 (ZFNet)**
 - **11 x 11 (AlexNet)**

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- **Multiple filter sizes** in on the same level
- **“Wider”** rather than “Deeper”

<http://datahacker.rs/building-inception-network/>

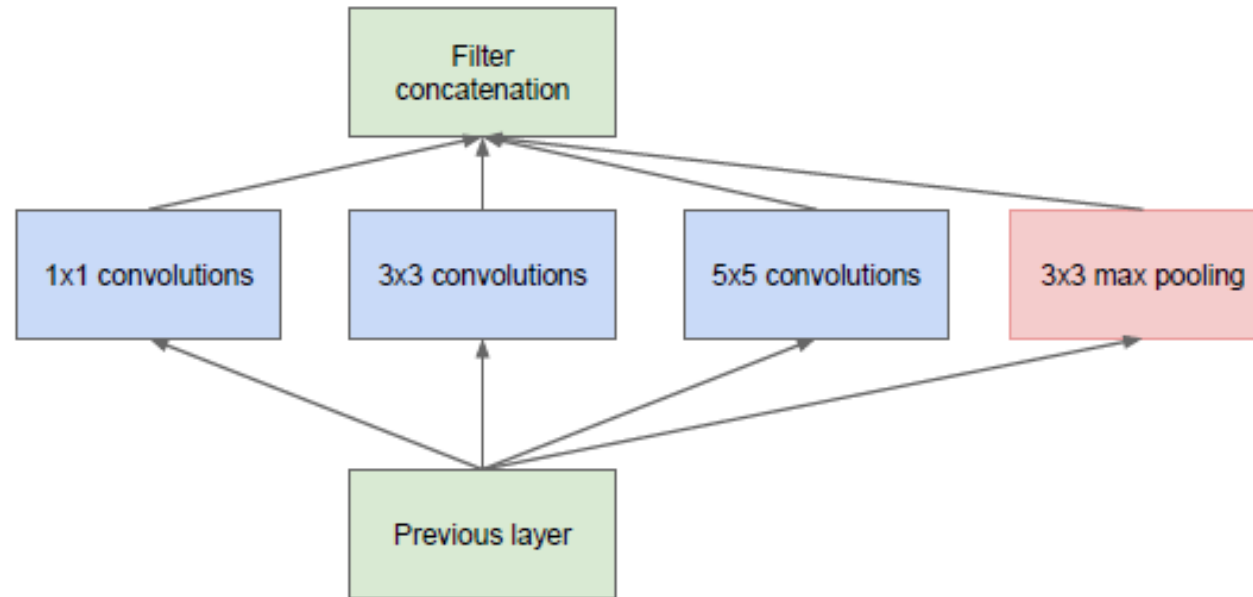
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- **Multiple filter sizes** in on the same level
- **“Wider”** rather than **“Deeper”** - **3 different filters**



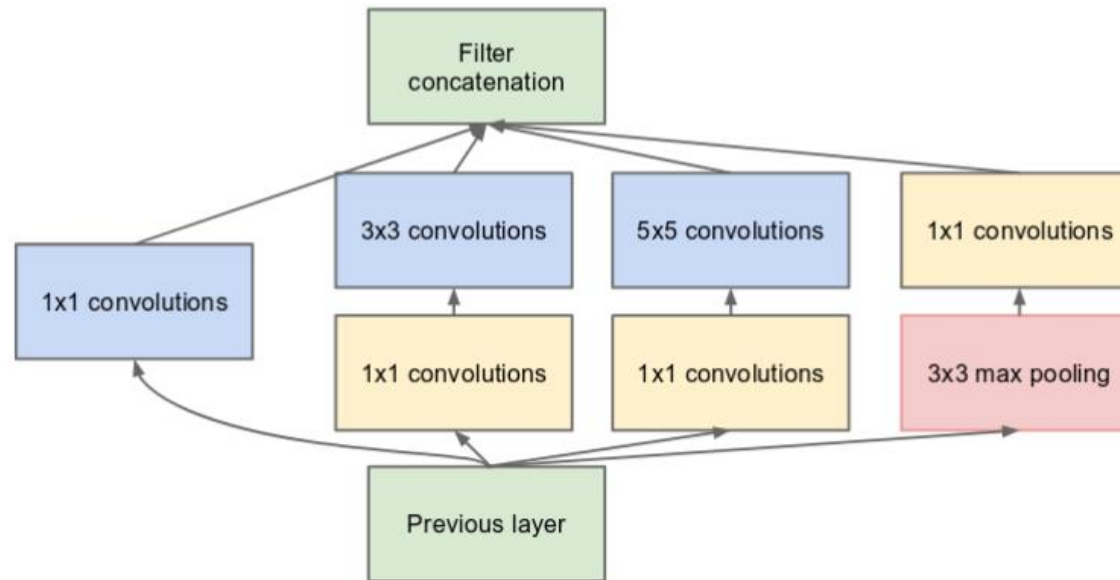
(a) Inception module, naïve version

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

- **1 x 1 filters for dimension reduction**
- **“Wider”** rather than “Deeper” - **3 different filters**



(b) Inception module with dimension reductions

<http://datahacker.rs/building-inception-network/>

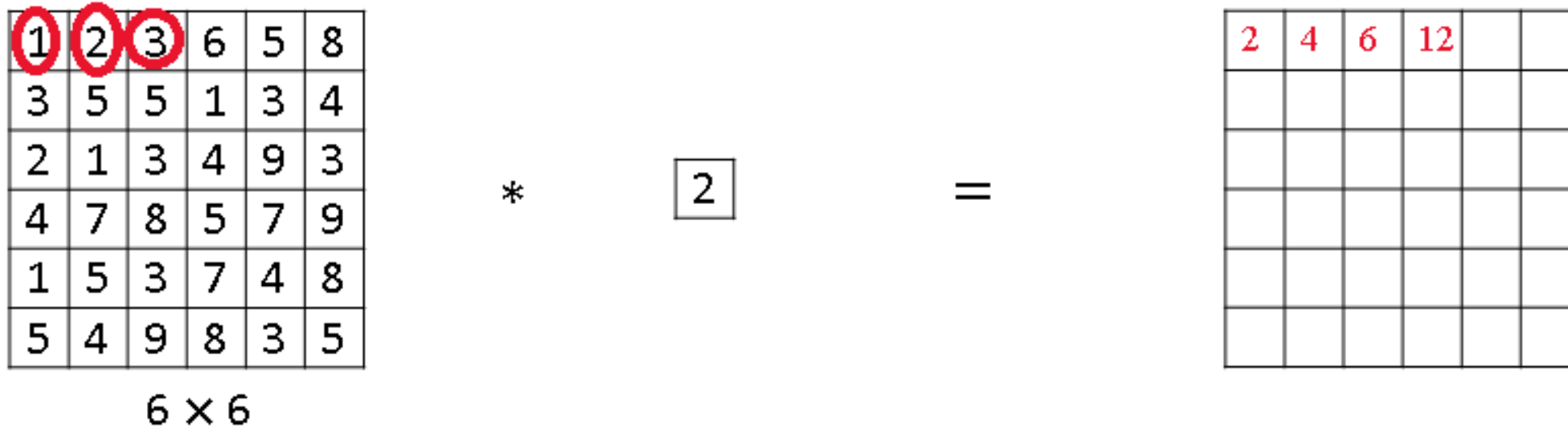
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- **1 x 1 filters in 2D**



<https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html>

<http://datahacker.rs/building-inception-network/>

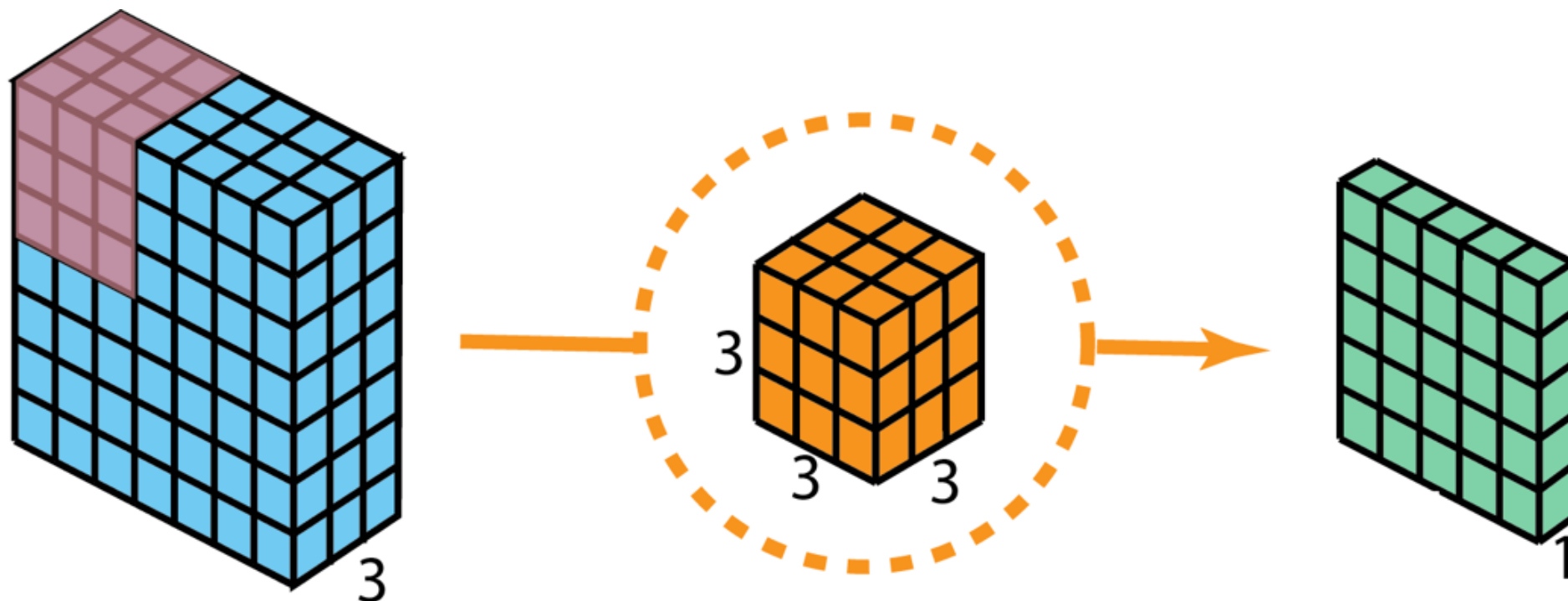
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- filter in 3D



<https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578>

<http://datahacker.rs/building-inception-network/>

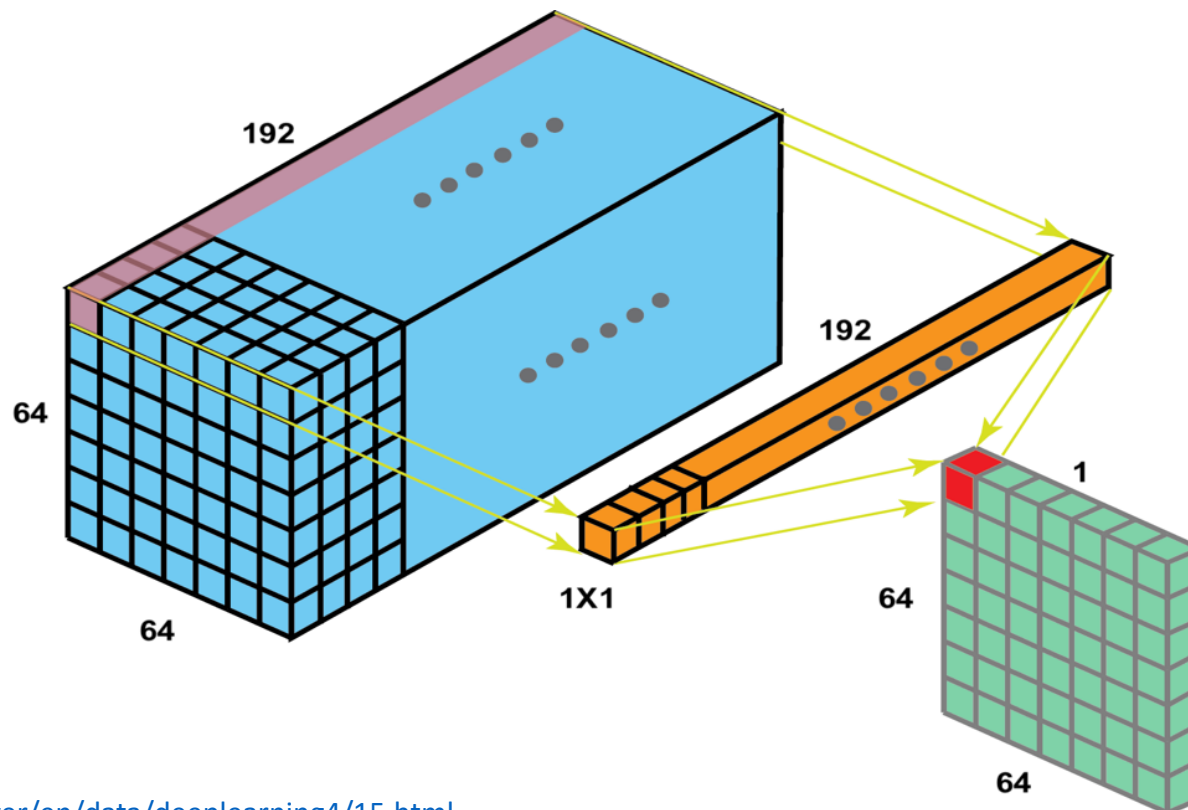
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- **1 x 1 filters** in large number of channels - **192**



<https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html>

<https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578>

<http://datahacker.rs/building-inception-network/>

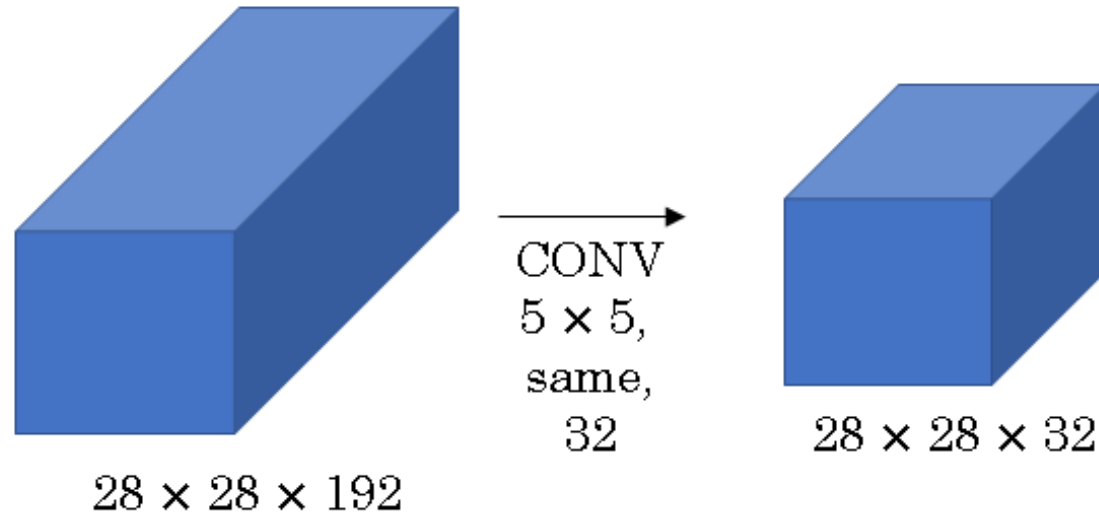
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- convolve $28 \times 28 \times 192$ input feature maps with $5 \times 5 \times 32$ filters.
- **This will result in aprox. 120 Million operations**



<https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html>

<https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578>

<http://datahacker.rs/building-inception-network/>

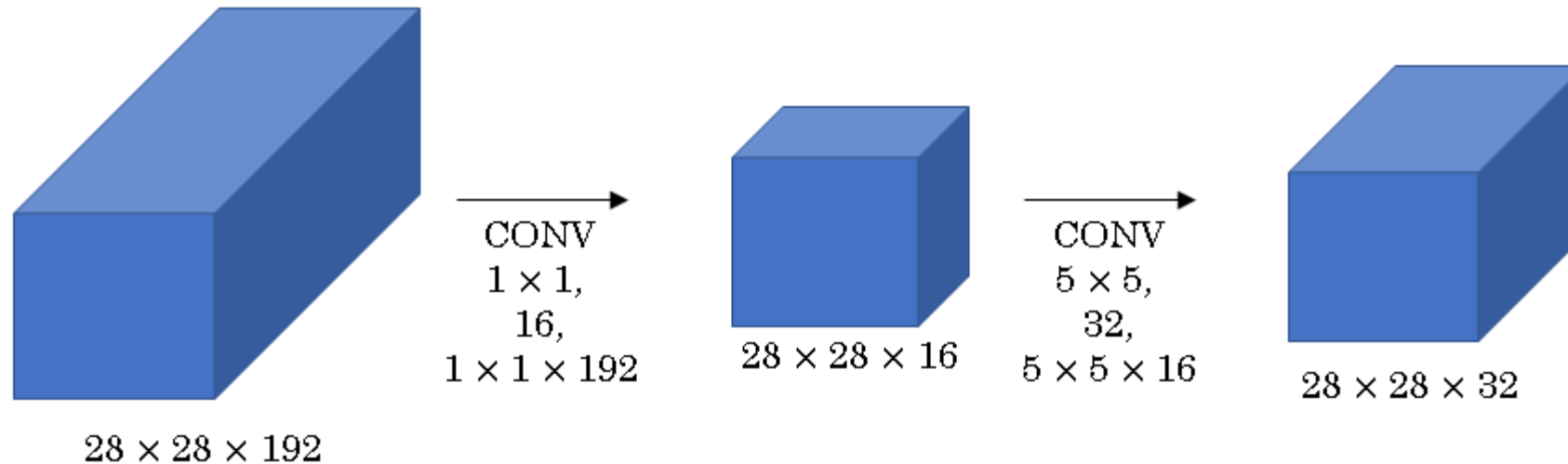
http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014

- Using 1 X 1 filters - **aprox. 12 Million operations**
- “Bottleneck layer”



<https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html>

<https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578>

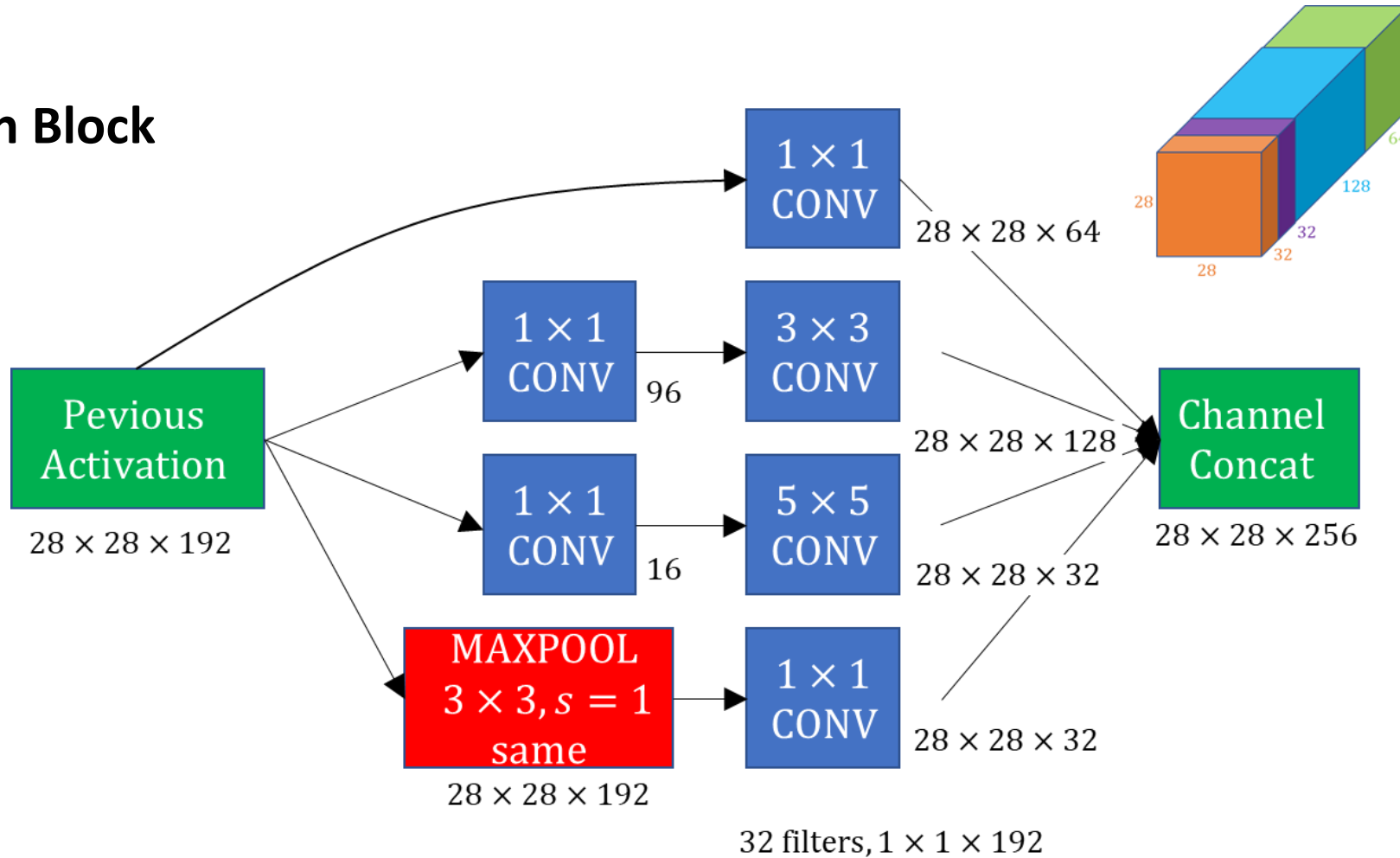
<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

GoogLeNet - 2014

- Inception Block



<https://upscfever.com/upsc-fever/en/data/deeplearning4/15.html>

<https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578>

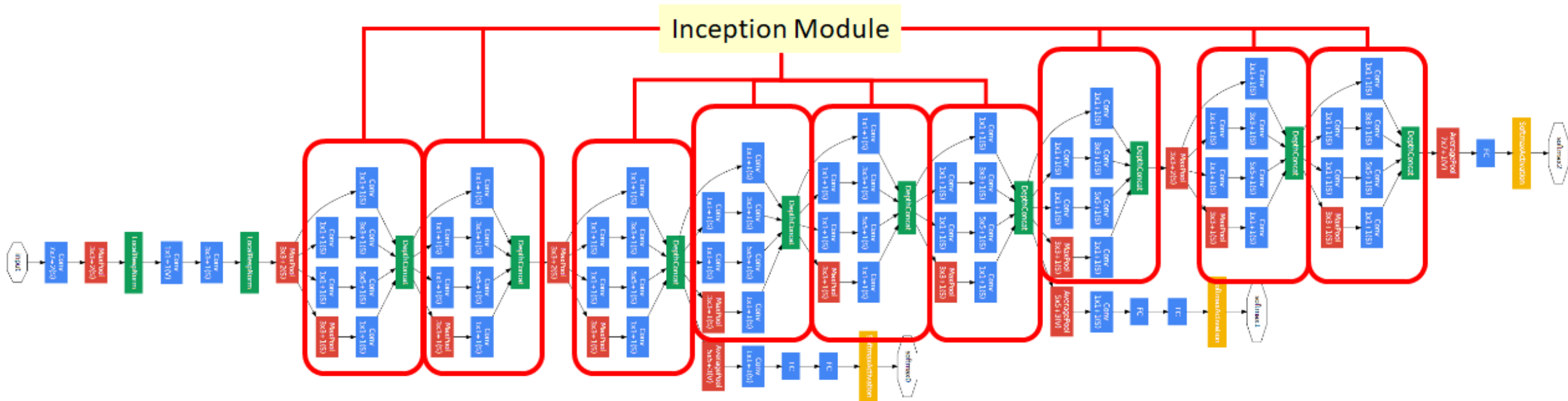
<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet - 2014



<https://towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc>

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



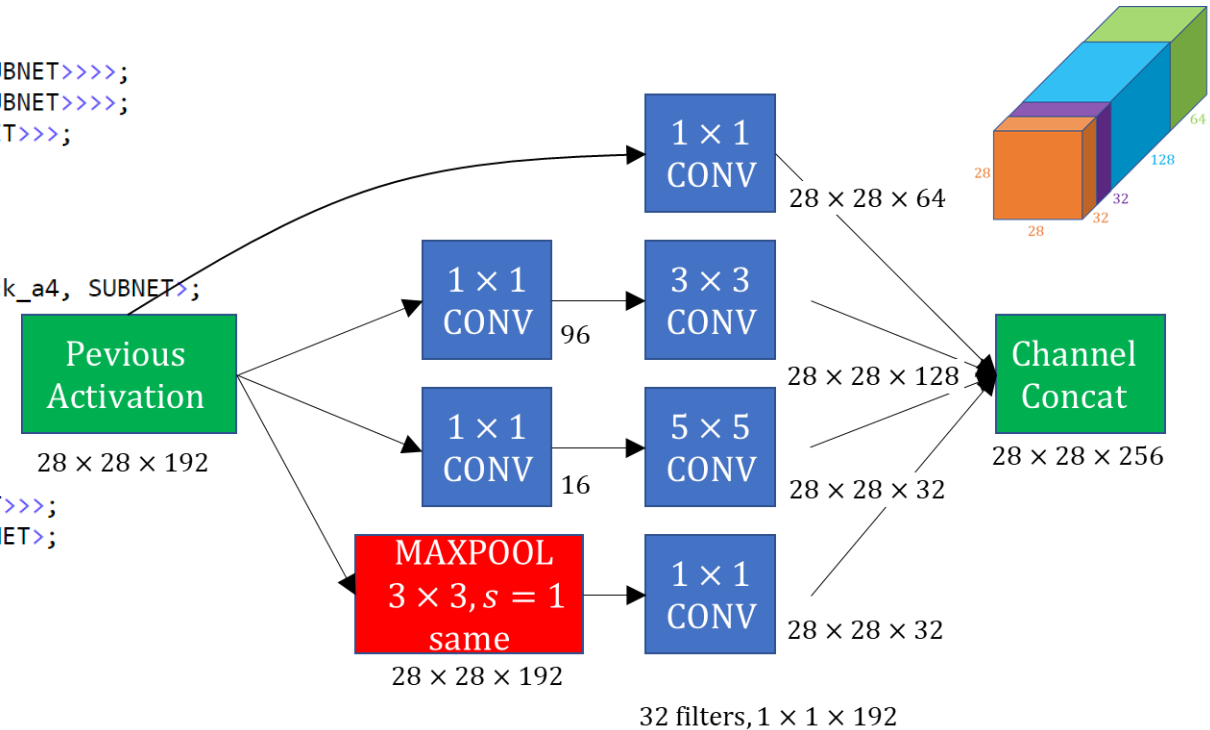
GoogLeNet – 2014 Dlib

```
// Inception layer has some different convolutions inside. Here we define
// blocks as convolutions with different kernel size that we will use in
// inception layer block.
template <typename SUBNET> using block_a1 = relu<con<10,1,1,1,1,SUBNET>>;
template <typename SUBNET> using block_a2 = relu<con<10,3,3,1,1,relu<con<16,1,1,1,1,SUBNET>>>>;
template <typename SUBNET> using block_a3 = relu<con<10,5,5,1,1,relu<con<16,1,1,1,1,SUBNET>>>>;
template <typename SUBNET> using block_a4 = relu<con<10,1,1,1,1,max_pool<3,3,1,1,SUBNET>>>>;

// Here is inception layer definition. It uses different blocks to process input
// and returns combined output. Dlib includes a number of these inceptionN
// layer types which are themselves created using concat layers.
template <typename SUBNET> using incept_a = inception4<block_a1,block_a2,block_a3,block_a4, SUBNET>;

// Network can have inception layers of different structure. It will work
// properly so long as all the sub-blocks inside a particular inception block
// output tensors with the same number of rows and columns.
template <typename SUBNET> using block_b1 = relu<con<4,1,1,1,1,SUBNET>>;
template <typename SUBNET> using block_b2 = relu<con<4,3,3,1,1,SUBNET>>;
template <typename SUBNET> using block_b3 = relu<con<4,1,1,1,1,max_pool<3,3,1,1,SUBNET>>>>;
template <typename SUBNET> using incept_b = inception3<block_b1,block_b2,block_b3,SUBNET>;

// Now we can define a simple network for classifying MNIST digits. We will
// train and test this network in the code below.
using net_type = loss_multiclass_log<
    fc<10,
    relu<fc<32,
    max_pool<2,2,2,2,incept_b<
    max_pool<2,2,2,2,incept_a<
    input<matrix<unsigned_char>>
    >>>>>>>;
```



http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet – 2014 - Pytorch

```
class Inception(nn.Block):
    # `c1`--`c4` are the number of output channels for each path
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)

    # Path 1 is a single 1 x 1 convolutional layer
    self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')
```

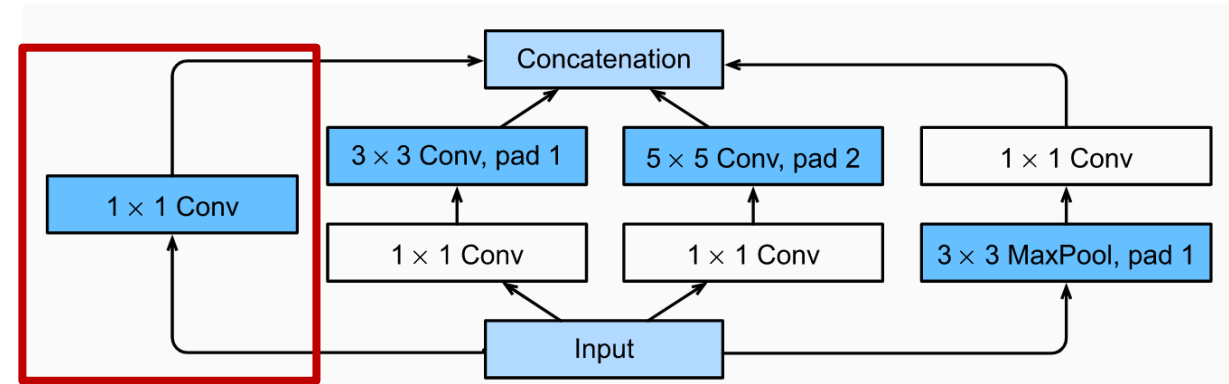


Fig. 7.4.1 Structure of the Inception block.

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet – 2014 - Pytorch

```
class Inception(nn.Block):
    # `c1`--`c4` are the number of output channels for each path
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)

        # Path 1 is a single 1 x 1 convolutional layer
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')

        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3
        # convolutional layer
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1, activation='relu')
```

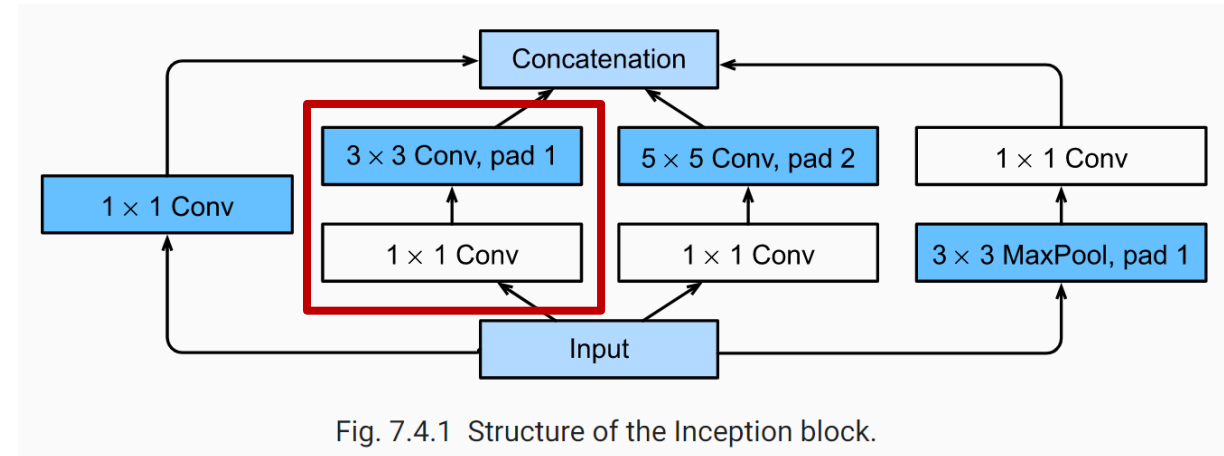


Fig. 7.4.1 Structure of the Inception block.

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet – 2014 - Pytorch

```
class Inception(nn.Block):
    # `c1`--`c4` are the number of output channels for each path
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)

        # Path 1 is a single 1 x 1 convolutional layer
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')

        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3
        # convolutional layer
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1, activation='relu')

        # Path 3 is a 1 x 1 convolutional layer followed by a 5 x 5
        # convolutional layer
        self.p3_1 = nn.Conv2D(c3[0], kernel_size=1, activation='relu')
        self.p3_2 = nn.Conv2D(c3[1], kernel_size=5, padding=2, activation='relu')
```

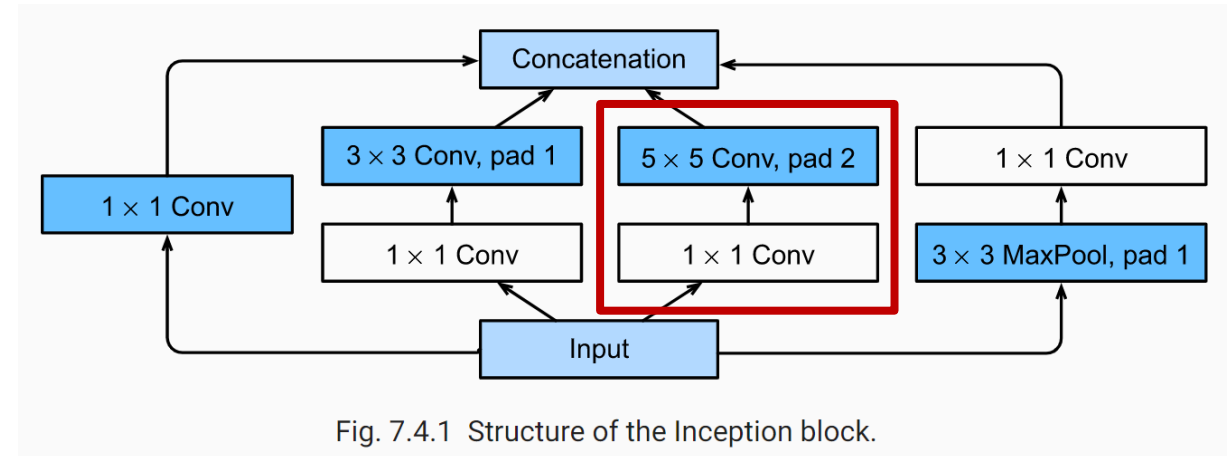


Fig. 7.4.1 Structure of the Inception block.

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

GoogLeNet – 2014 - PyTorch

```
class Inception(nn.Block):  
    # `c1`--`c4` are the number of output channels for each path  
    def __init__(self, c1, c2, c3, c4, **kwargs):  
        super(Inception, self).__init__(**kwargs)  
  
        # Path 1 is a single 1 x 1 convolutional layer  
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')  
  
        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3  
        # convolutional layer  
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')  
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1, activation='relu')  
  
        # Path 3 is a 1 x 1 convolutional layer followed by a 5 x 5  
        # convolutional layer  
        self.p3_1 = nn.Conv2D(c3[0], kernel_size=1, activation='relu')  
        self.p3_2 = nn.Conv2D(c3[1], kernel_size=5, padding=2, activation='relu')  
  
        # Path 4 is a 3 x 3 maximum pooling layer followed by a 1 x 1  
        # convolutional layer  
        self.p4_1 = nn.MaxPool2D(pool_size=3, strides=1, padding=1)  
        self.p4_2 = nn.Conv2D(c4, kernel_size=1, activation='relu')
```

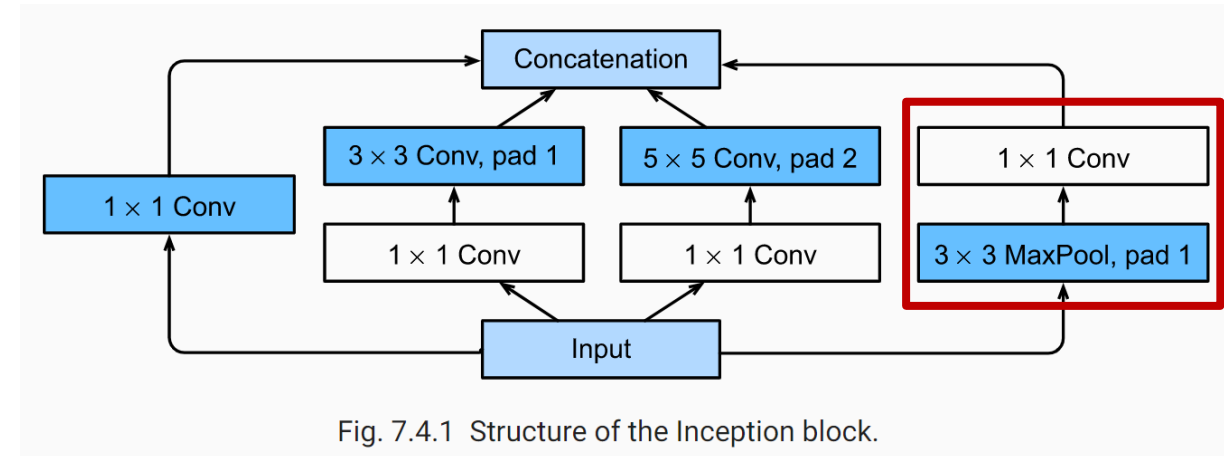


Fig. 7.4.1 Structure of the Inception block.

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>



GoogLeNet – 2014 - PyTorch

```
class Inception(nn.Module):
    # `c1`--`c4` are the number of output channels for each path
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)

        # Path 1 is a single 1 x 1 convolutional layer
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')

        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3
        # convolutional layer
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1, activation='relu')

        # Path 3 is a 1 x 1 convolutional layer followed by a 5 x 5
        # convolutional layer
        self.p3_1 = nn.Conv2D(c3[0], kernel_size=1, activation='relu')
        self.p3_2 = nn.Conv2D(c3[1], kernel_size=5, padding=2, activation='relu')

        # Path 4 is a 3 x 3 maximum pooling layer followed by a 1 x 1
        # convolutional layer
        self.p4_1 = nn.MaxPool2D(pool_size=3, strides=1, padding=1)
        self.p4_2 = nn.Conv2D(c4, kernel_size=1, activation='relu')

    def forward(self, x):
        p1 = self.p1_1(x)
        p2 = self.p2_2(self.p2_1(x))
        p3 = self.p3_2(self.p3_1(x))
        p4 = self.p4_2(self.p4_1(x))

        # Concatenate the outputs on the channel dimension
        return np.concatenate((p1, p2, p3, p4), axis=1)
```

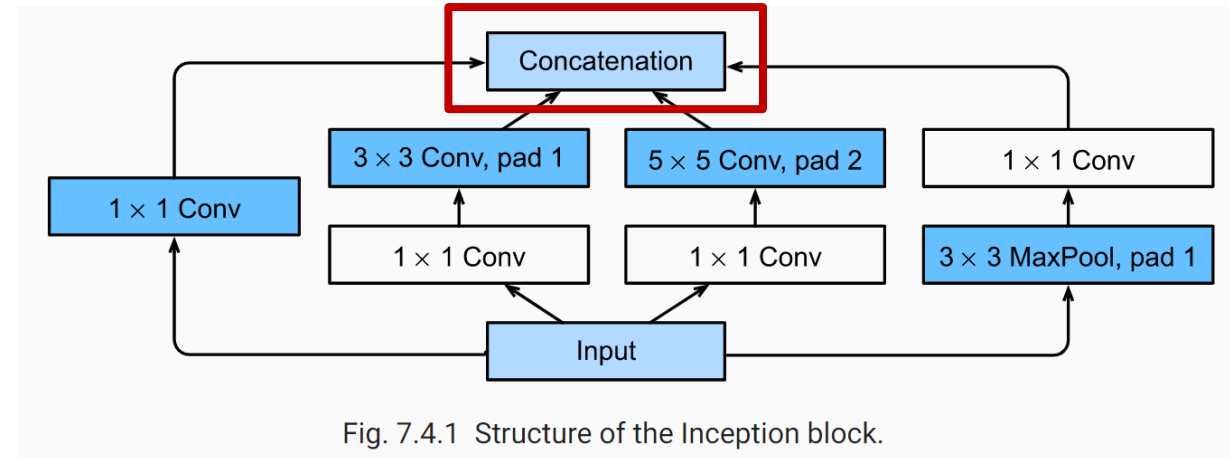


Fig. 7.4.1 Structure of the Inception block.

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

GoogLeNet – 2014 - PyTorch

```
class Inception(nn.Block):  
    # `c1`--`c4` are the number of output channels for each path  
    def __init__(self, c1, c2, c3, c4, **kwargs):  
        super(Inception, self).__init__(**kwargs)  
  
        # Path 1 is a single 1 x 1 convolutional layer  
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')  
  
        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3  
        # convolutional layer  
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')  
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1, activation='relu')  
  
        # Path 3 is a 1 x 1 convolutional layer followed by a 5 x 5  
        # convolutional layer  
        self.p3_1 = nn.Conv2D(c3[0], kernel_size=1, activation='relu')  
        self.p3_2 = nn.Conv2D(c3[1], kernel_size=5, padding=2, activation='relu')  
  
        # Path 4 is a 3 x 3 maximum pooling layer followed by a 1 x 1  
        # convolutional layer  
        self.p4_1 = nn.MaxPool2D(pool_size=3, strides=1, padding=1)  
        self.p4_2 = nn.Conv2D(c4, kernel_size=1, activation='relu')  
  
    def forward(self, x):  
        p1 = self.p1_1(x)  
        p2 = self.p2_2(self.p2_1(x))  
        p3 = self.p3_2(self.p3_1(x))  
        p4 = self.p4_2(self.p4_1(x))  
        # Concatenate the outputs on the channel dimension  
        return np.concatenate((p1, p2, p3, p4), axis=1)
```

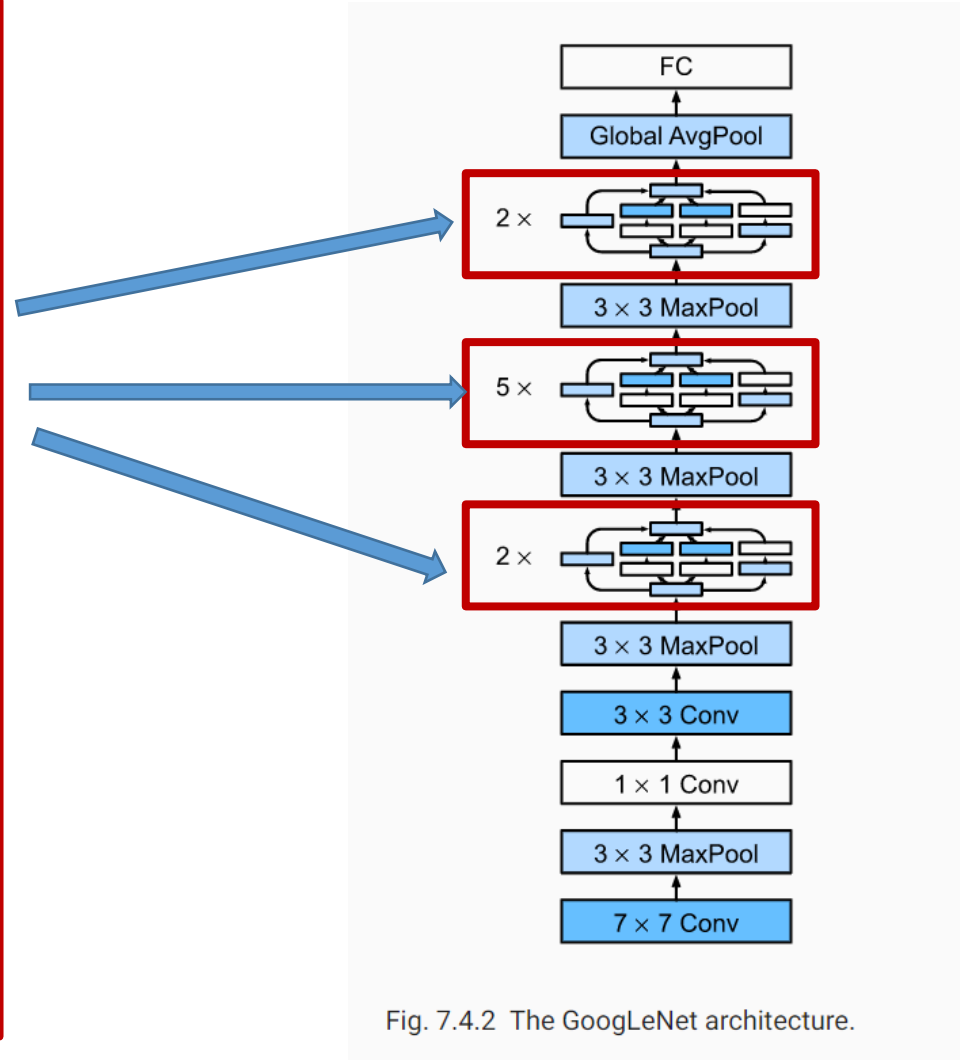


Fig. 7.4.2 The GoogLeNet architecture.

- http://dlib.net/dnn_inception_ex.cpp.html
- <http://datahacker.rs/building-inception-network/>
- http://d2l.ai/chapter_convolutional-modern/googlenet.html
- <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html

GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),  
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),  
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),  
                  Inception(256, 128, (128, 192), (32, 96), 64),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),  
                  Inception(512, 160, (112, 224), (24, 64), 64),  
                  Inception(512, 128, (128, 256), (24, 64), 64),  
                  Inception(512, 112, (144, 288), (32, 64), 64),  
                  Inception(528, 256, (160, 320), (32, 128), 128),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),  
                  Inception(832, 384, (192, 384), (48, 128), 128),  
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

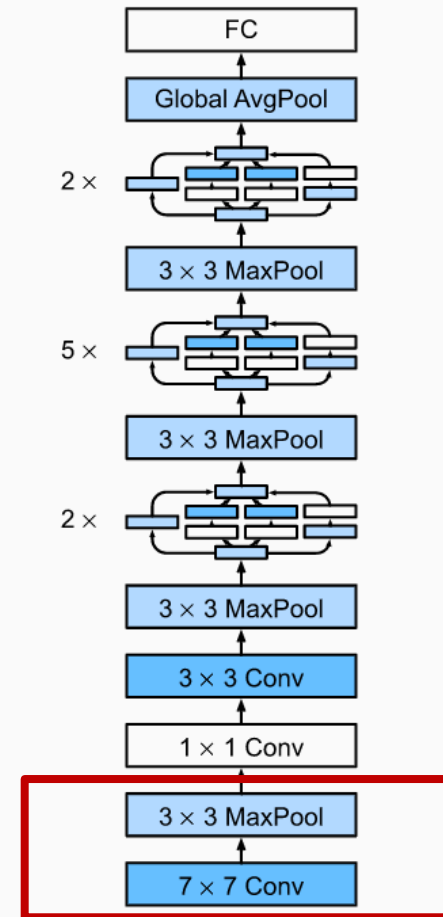


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),
                  Inception(256, 128, (128, 192), (32, 96), 64),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),
                  Inception(512, 160, (112, 224), (24, 64), 64),
                  Inception(512, 128, (128, 256), (24, 64), 64),
                  Inception(512, 112, (144, 288), (32, 64), 64),
                  Inception(528, 256, (160, 320), (32, 128), 128),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),
                  Inception(832, 384, (192, 384), (48, 128), 128),
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

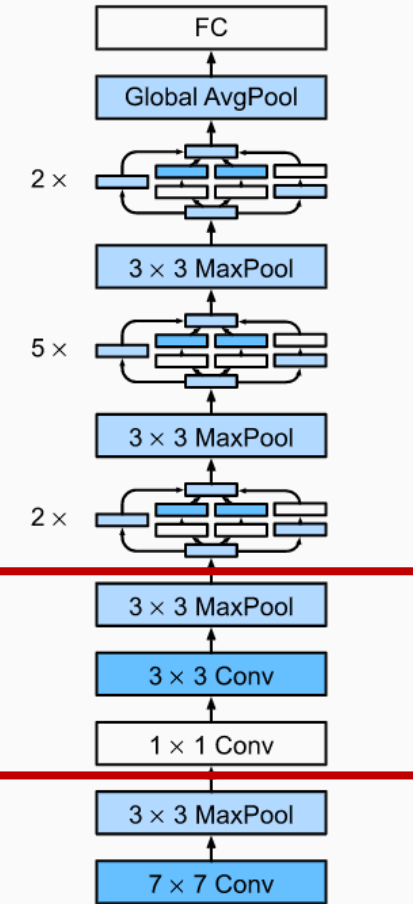


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),  
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),  
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),  
                  Inception(256, 128, (128, 192), (32, 96), 64),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),  
                  Inception(512, 160, (112, 224), (24, 64), 64),  
                  Inception(512, 128, (128, 256), (24, 64), 64),  
                  Inception(512, 112, (144, 288), (32, 64), 64),  
                  Inception(528, 256, (160, 320), (32, 128), 128),  
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),  
                  Inception(832, 384, (192, 384), (48, 128), 128),  
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

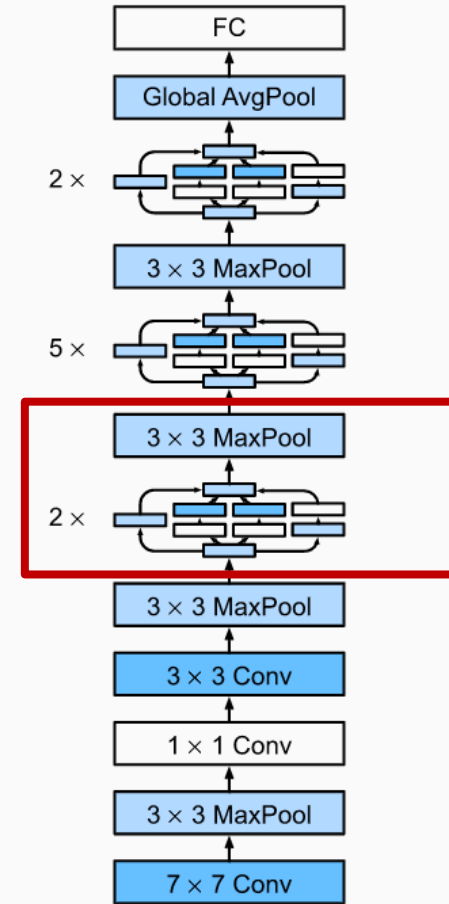


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),
                  Inception(256, 128, (128, 192), (32, 96), 64),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),
                  Inception(512, 160, (112, 224), (24, 64), 64),
                  Inception(512, 128, (128, 256), (24, 64), 64),
                  Inception(512, 112, (144, 288), (32, 64), 64),
                  Inception(528, 256, (160, 320), (32, 128), 128),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),
                  Inception(832, 384, (192, 384), (48, 128), 128),
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

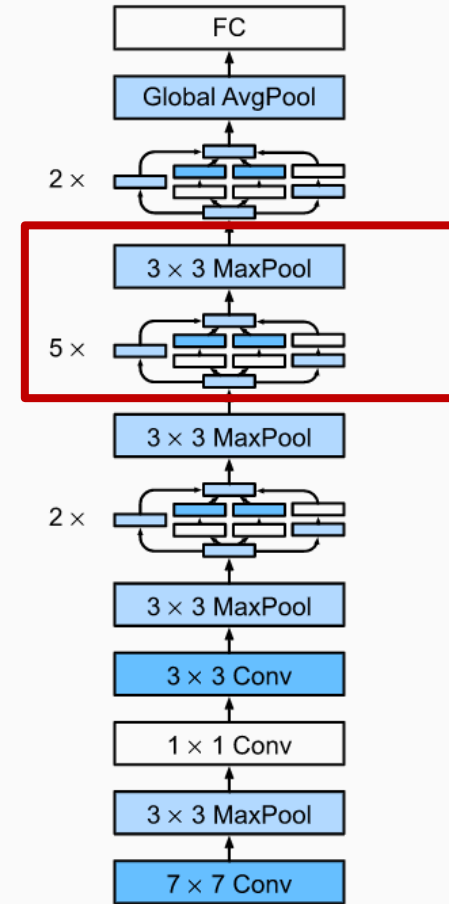


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),
                  Inception(256, 128, (128, 192), (32, 96), 64),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),
                  Inception(512, 160, (112, 224), (24, 64), 64),
                  Inception(512, 128, (128, 256), (24, 64), 64),
                  Inception(512, 112, (144, 288), (32, 64), 64),
                  Inception(528, 256, (160, 320), (32, 128), 128),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),
                  Inception(832, 384, (192, 384), (48, 128), 128),
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

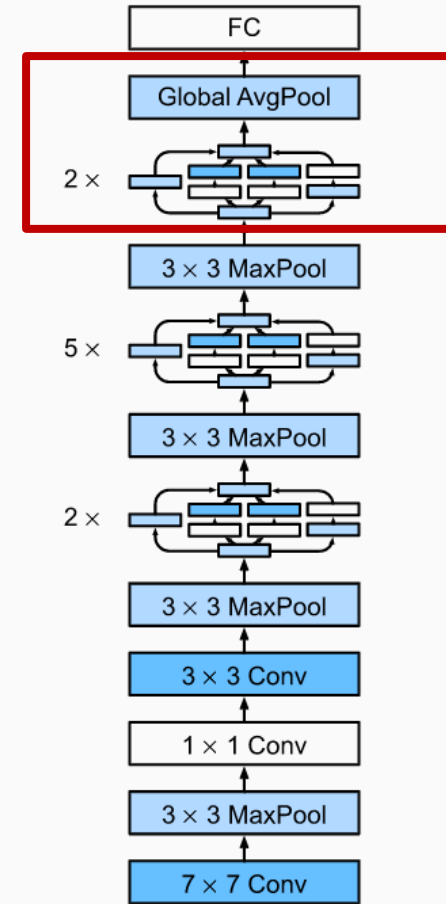


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



GoogLeNet – 2014 - PyTorch

```
b1 = nn.Sequential(nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
                  nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b2 = nn.Sequential(nn.Conv2d(64, 64, kernel_size=1), nn.ReLU(),
                  nn.Conv2d(64, 192, kernel_size=3, padding=1), nn.ReLU(),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b3 = nn.Sequential(Inception(192, 64, (96, 128), (16, 32), 32),
                  Inception(256, 128, (128, 192), (32, 96), 64),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b4 = nn.Sequential(Inception(480, 192, (96, 208), (16, 48), 64),
                  Inception(512, 160, (112, 224), (24, 64), 64),
                  Inception(512, 128, (128, 256), (24, 64), 64),
                  Inception(512, 112, (144, 288), (32, 64), 64),
                  Inception(528, 256, (160, 320), (32, 128), 128),
                  nn.MaxPool2d(kernel_size=3, stride=2, padding=1))
```

```
b5 = nn.Sequential(Inception(832, 256, (160, 320), (32, 128), 128),
                  Inception(832, 384, (192, 384), (48, 128), 128),
                  nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten())
```

```
GoogLeNet = nn.Sequential(b1, b2, b3, b4, b5, nn.Linear(1024, 2))
```

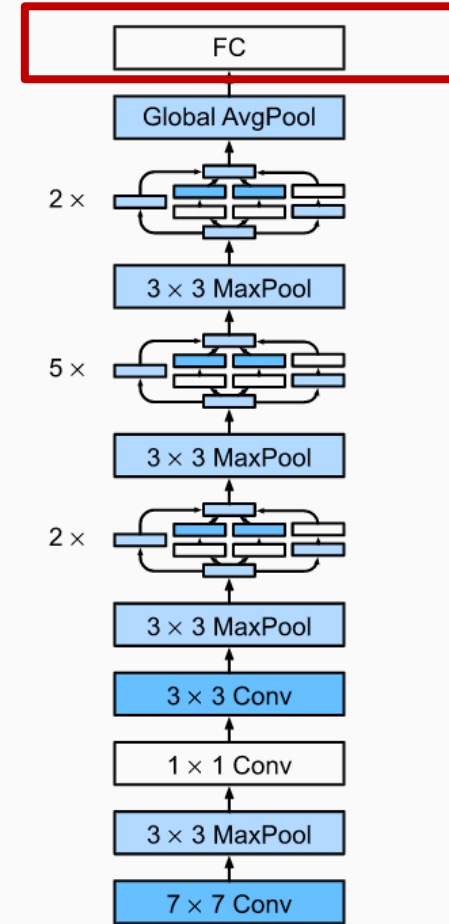


Fig. 7.4.2 The GoogLeNet architecture.

http://dlib.net/dnn_inception_ex.cpp.html

<http://datahacker.rs/building-inception-network/>

http://d2l.ai/chapter_convolutional-modern/googlenet.html

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

source: http://d2l.ai/chapter_convolutional-modern/googlenet.html



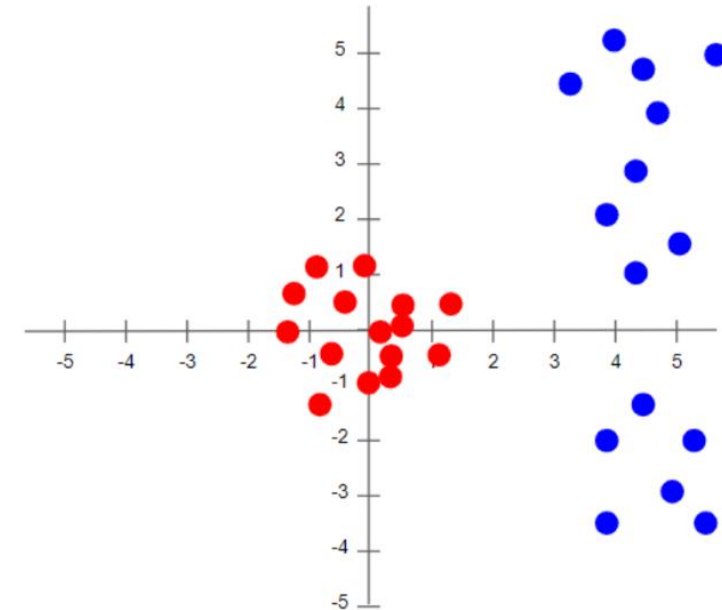
Batch normalization

- Additional layer that standardize the inputs to a network
- Helps to train very deep neural networks
- Accelerates training
- Can be used in many architectures of deep neural networks
- Solves the problem that the large feature can overshadow the small feature
- Uses mean and variance (průměr, rozptyl) of the current batch for a more stable distribution of values.
- <https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html>

<https://arxiv.org/pdf/1502.03167.pdf>

http://d2l.ai/chapter_convolutional-modern/batch-norm.html

In the picture below, we can see the effect of normalizing data. The original values (in blue) are now centered around zero (in red). This ensures that all the feature values are now on the same scale.



source: <https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739>



CovNet Architectures

- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**



2015

- ImageNet Challenge <http://www.image-net.org/challenges/LSVRC/2015/>
- <http://www.image-net.org/challenges/LSVRC/2015/results>
- http://image-net.org/challenges/talks/ILSVRC2015_12_17_15_clsloc.pdf
- **Deep networks are hard to train - Vanishing gradient issue**
- **Residual Blocks**
- **1 x 1 convolutions**

<http://datahacker.rs/deep-learning-residual-networks/>

He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

“Is learning better networks as easy as stacking more layers?”

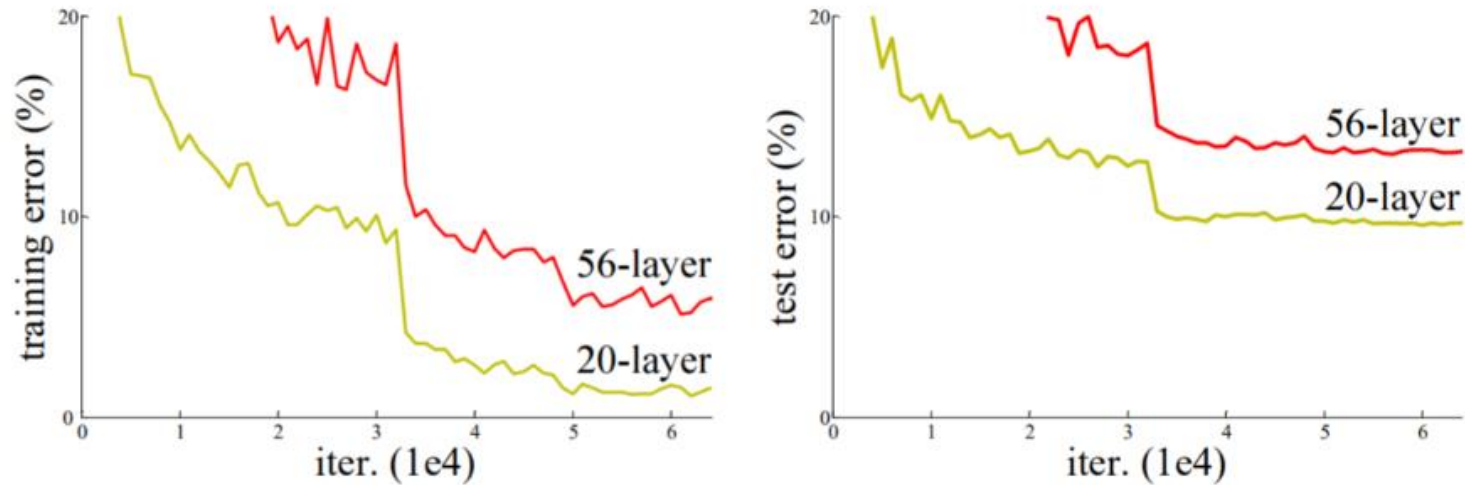


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNets - 2015

- in the **classical** way: the output of layer 1 is pass to layer 2
- in **ResNet**: information can go much deeper into the neural network
 - information (gradient) from higher layer can directly pass to the lower layer
 - via **shortcut** or **skip connection** (identity connection, addition operator)
 - stacking a lot of residual blocks together, we can build much deeper neural networks

“shortcut connections”

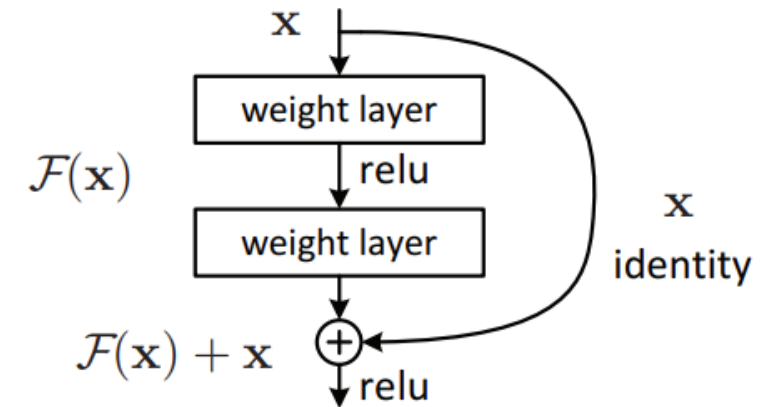


Figure 2. Residual learning: a building block.

ResNets - 2015

“shortcut connections”

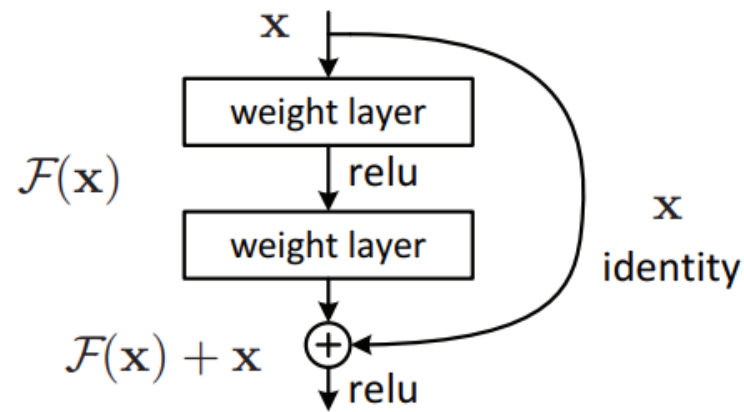


Figure 2. Residual learning: a building block.

“bottleneck”

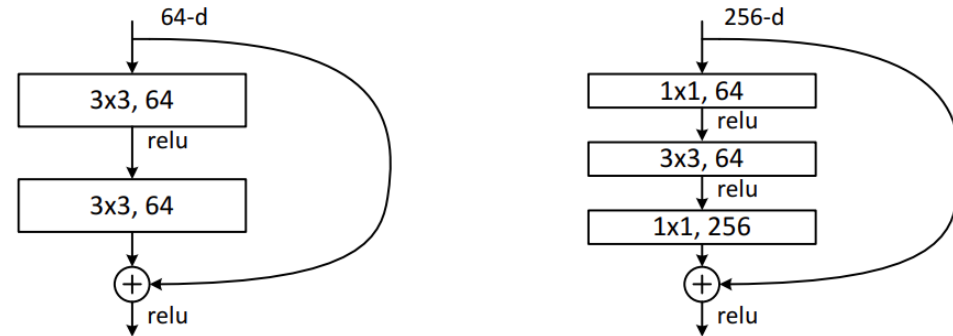


Figure 5. A deeper residual function \mathcal{F} for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.



ResNets - 2015

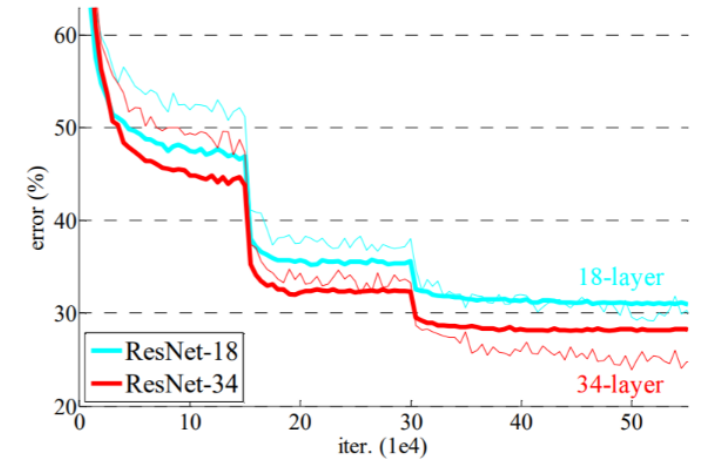
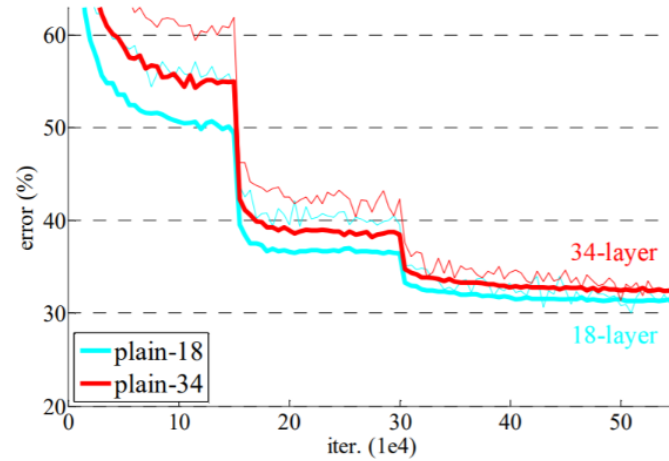
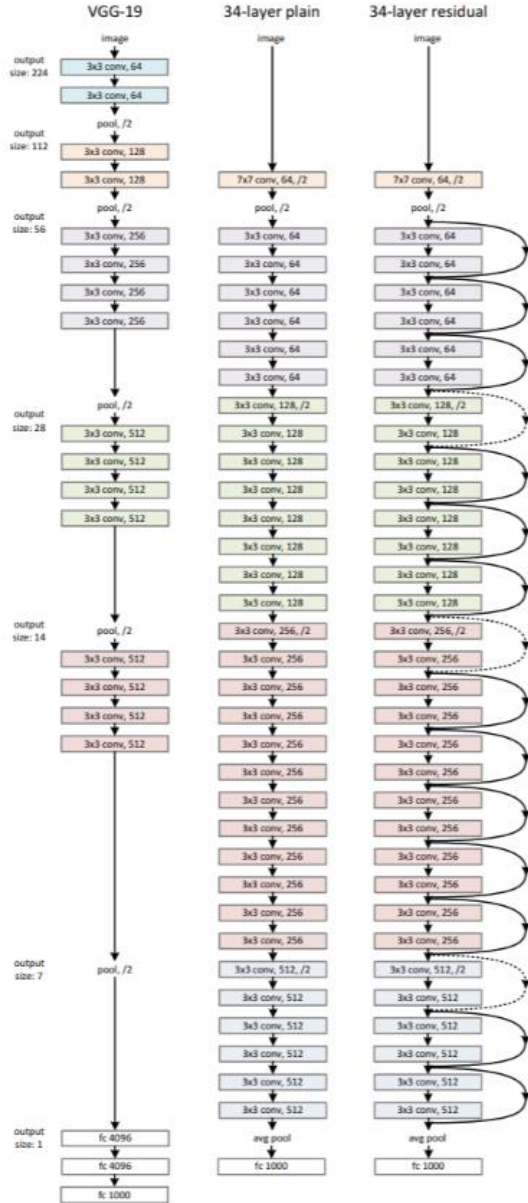


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

ResNets - 2015

Table 2. Classification error (%) on the CIFAR-10 test set using different activation functions.

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	6.37	5.46

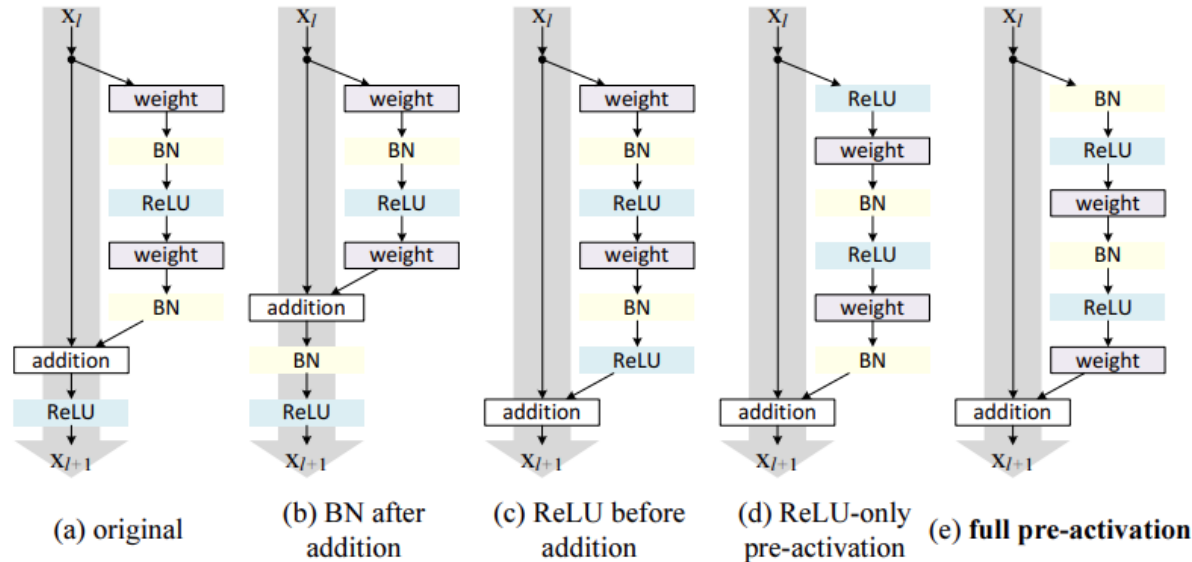


Figure 4. Various usages of activation in Table 2. All these units consist of the same components — only the orders are different.



Variations of ResNets:

- Inception-ResNet V1
- Inception-ResNet-v2
- ResNeXt - 1st Runner Up of ILSVRC classification task
 - <http://image-net.org/challenges/LSVRC/2016/results>

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
Winner of ILSVRC 2015 ResNet-101 [14]	22.0	6.0	-	-
Pre-Activation ResNet ResNet-200 [15]	21.7	5.8	20.1	4.8
1 st Runner-Up of ILSVRC 2015 Inception-v3 [39]	-	-	21.2	5.6
Better Than Inception-v3 Inception-v4 [37]	-	-	20.0	5.0
Inception-v4 + ResNet Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (64 × 4d)	20.4	5.3	19.1	4.4

<https://towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc>

<https://towardsdatascience.com/review-resnext-1st-runner-up-of-ilsvrc-2016-image-classification-15d7f17b42ac>

S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 5987-5995, doi: 10.1109/CVPR.2017.634.

He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Identity Mappings in Deep Residual Networks. 9908. 630-645. 10.1007/978-3-319-46493-0_38.

He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.



CovNet Architectures

- **LeNet (1990s)**
- **AlexNet (2012)**
- **ZF Net (2013)**
- **VGGNet (2014)**
- **GoogLeNet (2014)**
- **ResNets (2015)**
- **DenseNet (2017)**

DenseNet - 2017

- connects all layers directly with each other
- handle vanishing gradients problem
- addition vs. concatenation (ResNet vs. DenseNet)
- dense blocks

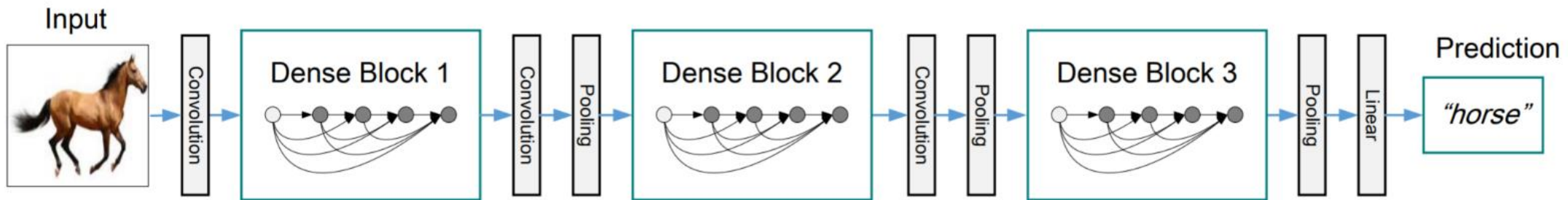
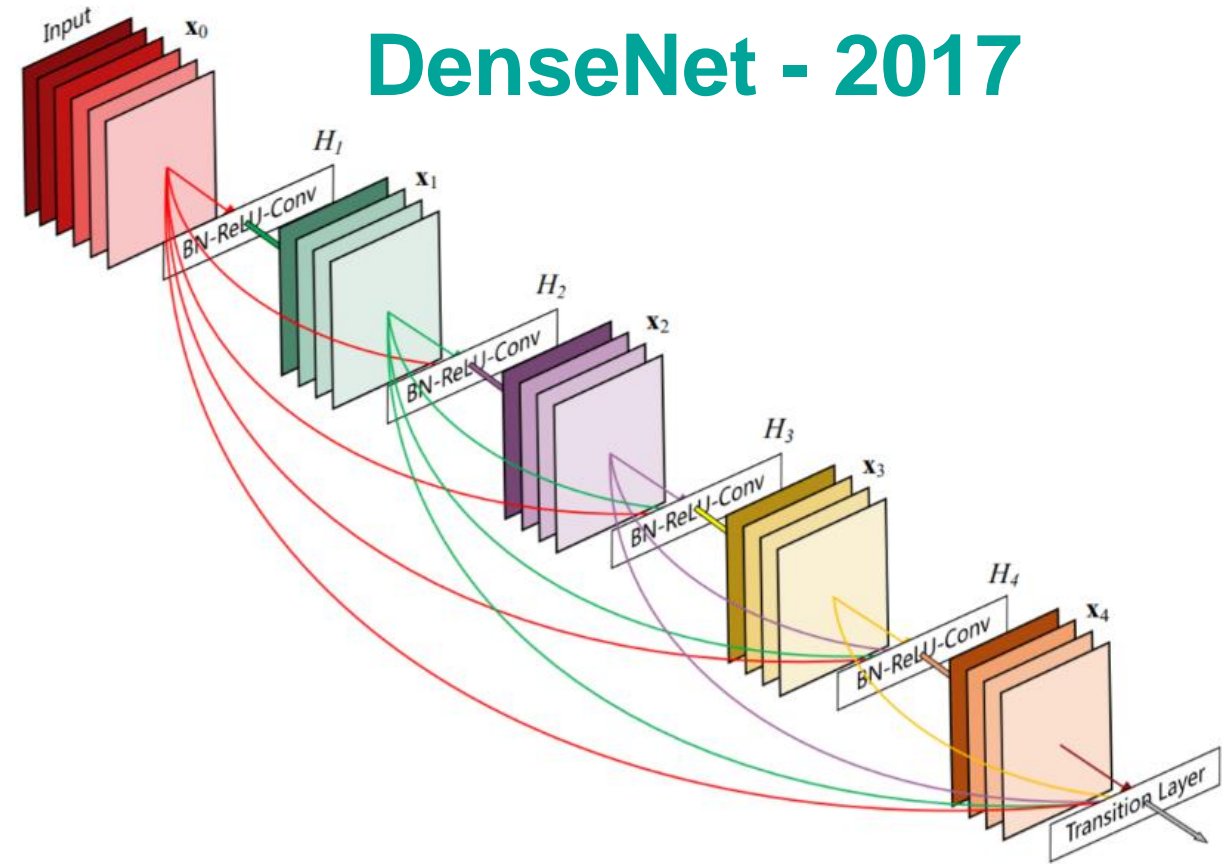


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.



Transfer Learning - PyTorch

- reuse the parameters of the trained network
 - for example on large dataset (ImageNet)
- modified the network with the use of freezing the parameters or changing the output layer
- **Option 1 - Freeze all layers except the last fc layer that needs to be modified due to the class number:**

```
resnet18 = models.resnet18(pretrained = True)

#print info
for name, child in resnet18.named_children():
    print("name: ", name)

# turn off gradient computation
for param in resnet18.parameters():
    param.requires_grad = False

# change number of out classes
num_ftrs = resnet18.fc.in_features
print("resnet18.fc ", resnet18.fc)
resnet18.fc = nn.Linear(num_ftrs, 2)
resnet18 = resnet18.to(device)

#optimize params in fc layer only
optimizer = optim.SGD(resnet18.fc.parameters(), lr=0.001, momentum=0.9)
```



Transfer Learning - PyTorch

- **Option 2 - Freeze and Unfreeze selected layers:**

```
resnet18 = models.resnet18(pretrained = True)

# freeze all layers in the body of the network
for param in resnet18.parameters():
    param.requires_grad = False

num_ftrs = resnet18.fc.in_features
# modified class number in fc layer
resnet18.fc = nn.Linear(num_ftrs, 2)

# unfreeze layer4 + fc
for name,param in resnet18.named_parameters():
    print("\t",name)
    if( ("fc" in name) or ("layer4" in name)):
        param.requires_grad = True

resnet18 = resnet18.to(device)
#optimize selected params
params_to_update = [param for param in resnet18.parameters() if param.requires_grad==True]
print(len(params_to_update))
optimizer = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
```



Transfer Learning - PyTorch

- **Option 3 – Add a new FC layer (similar to Option 1):**

```
# freeze all layers in the body of the network
for param in resnet18.parameters():
    param.requires_grad = False

num_ftrs = resnet18.fc.in_features
#define last layer with modified class number
myFC = nn.Sequential(
    nn.Linear(num_ftrs, 512),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(256, 2)
)

resnet18.fc = myFC
resnet18 = resnet18.to(device)

#optimize params in fc layer only
optimizer = optim.SGD(resnet18.fc.parameters(), lr=0.001, momentum=0.9)
```



Region-Based CNNs (R-CNNs)

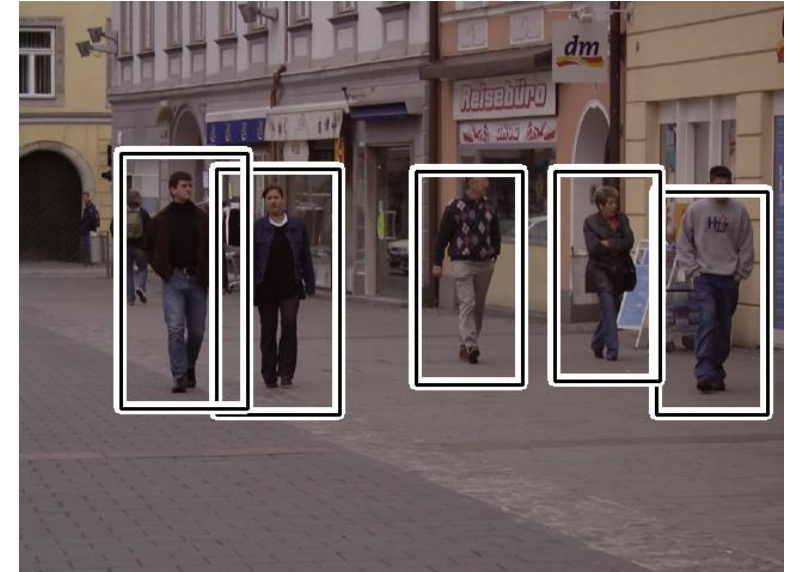
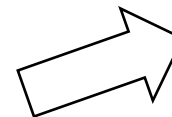
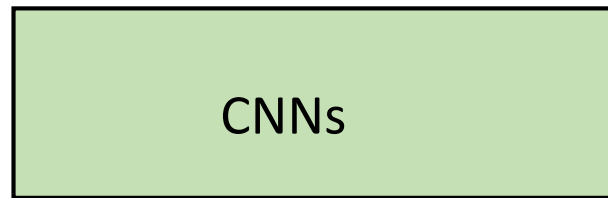
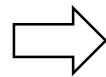
- Classical way (how to localize/detect object) is based on sliding window technique





Region-Based CNNs (R-CNNs)

- Disadvantages of sliding window with the use of very deep CNNs for object detection
 - many different image regions
 - each region is used as an input for CNNs
 - computational cost – overlapping regions (stride parameters)
 - duplicated operations





Region-Based CNNs (R-CNNs)

- **R-CNN**

- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580–587).

- **Fast R-CNN**

- Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision* (pp. 1440–1448).

- **Faster R-CNN**

- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: towards real-time object detection with region proposal networks. *Advances in neural information processing systems* (pp. 91–99).

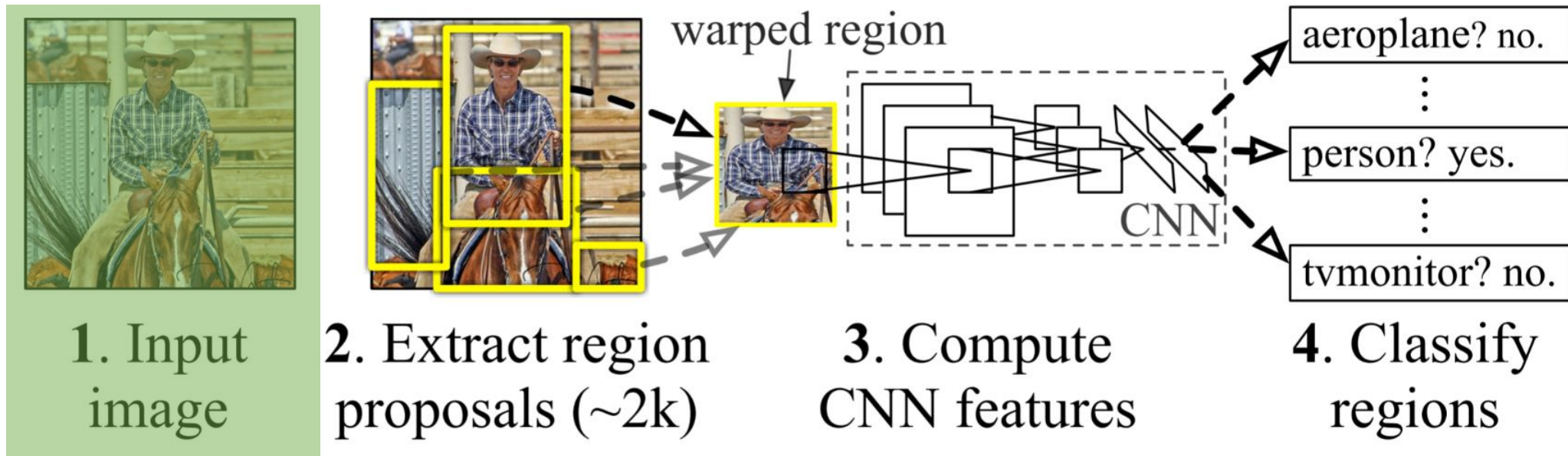
- **Mask R-CNN**

- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. *Proceedings of the IEEE international conference on computer vision* (pp. 2961–2969).



Region-Based CNNs (R-CNNs)

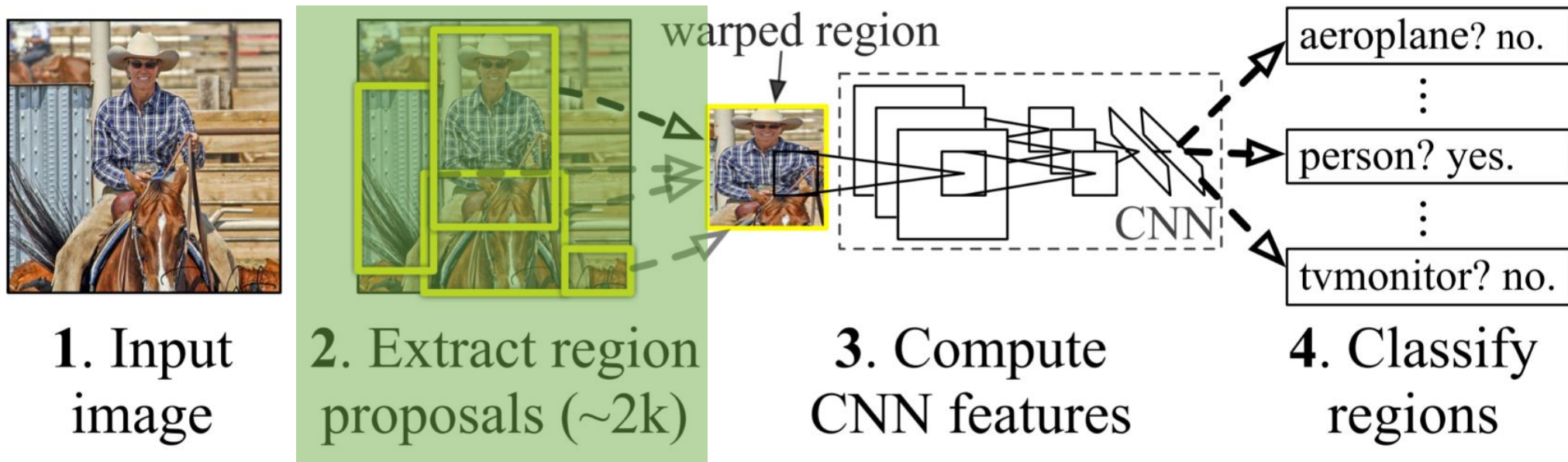
- R-CNN - 2014
- (1) takes an input image





Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**
- (1) takes an input image
- **(2) extracts around 2000 bottom-up regions using selective search**
 - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013

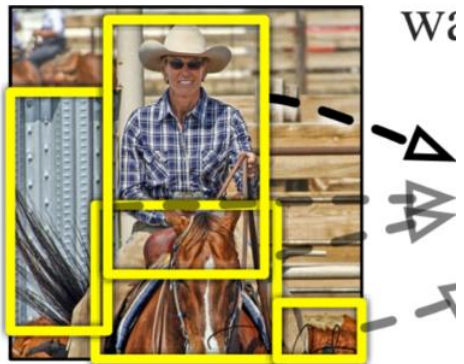




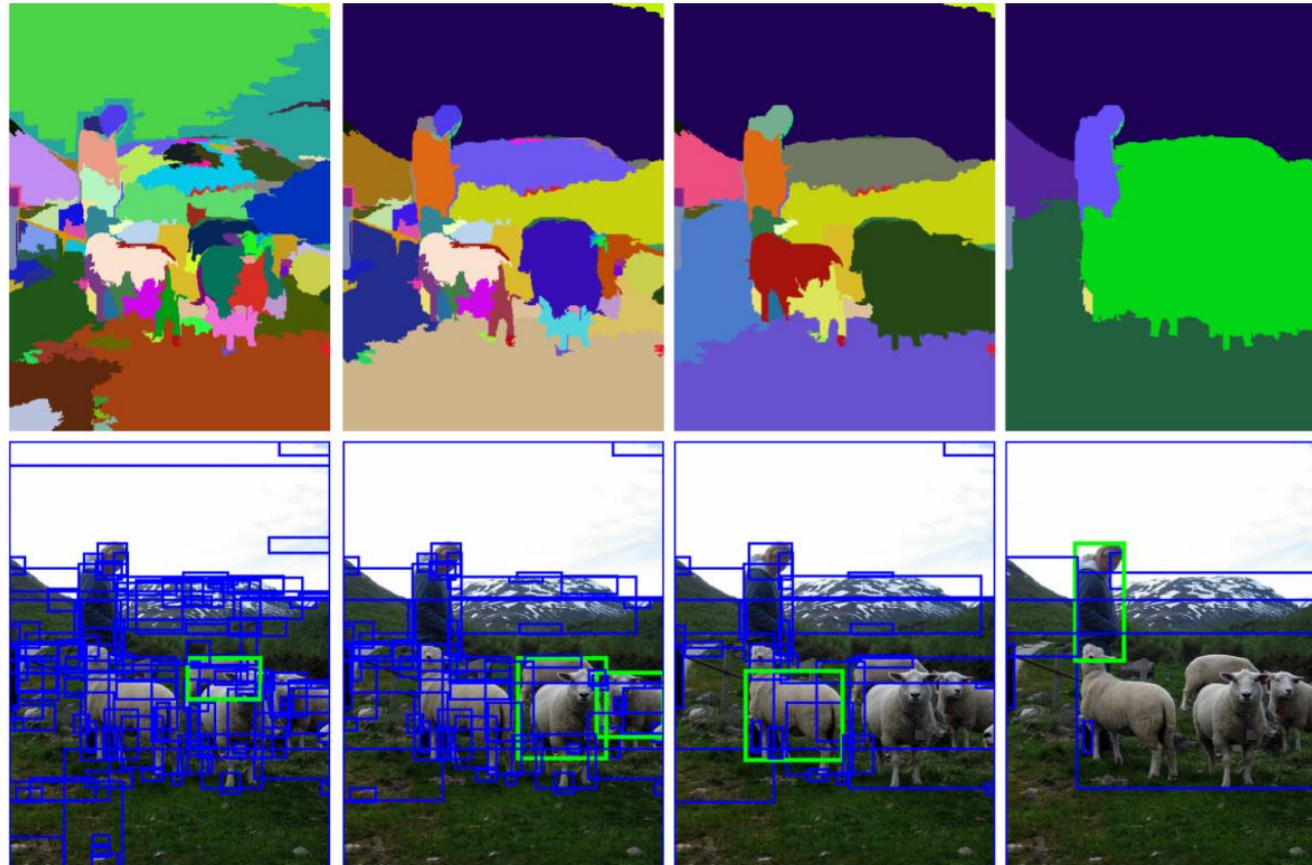
Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**

- (1) takes an input image
- **(2) extracts around 2000 bottom-up regions using selective search**
 - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013



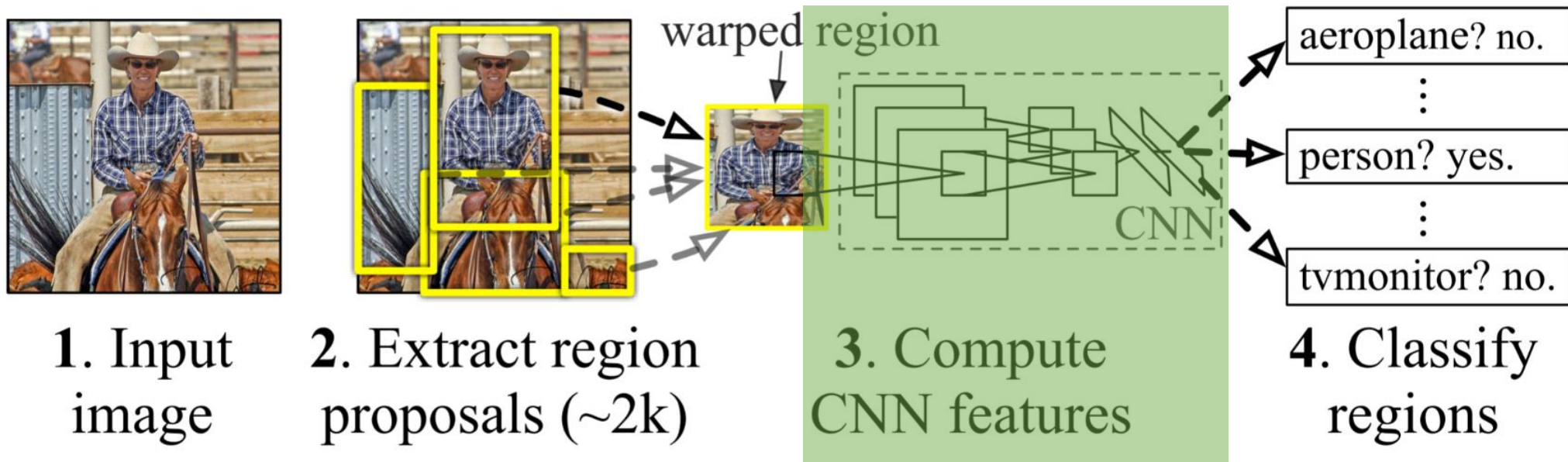
2. Extract region proposals (~2k)





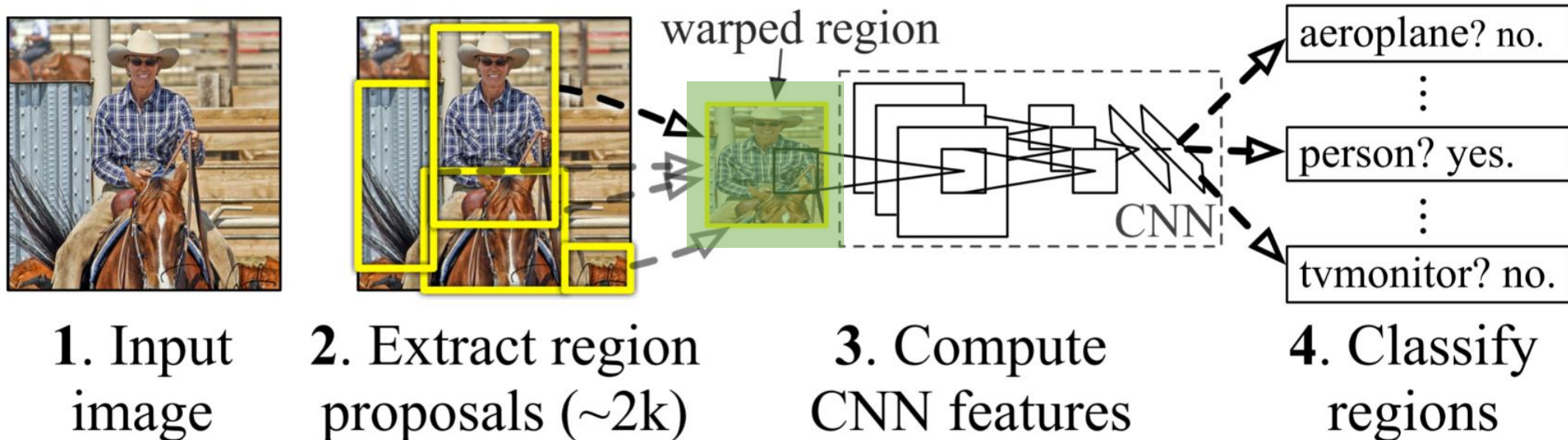
Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**
- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
 - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- **(3) computes features for each region using a large convolutional neural network (CNN)**
 - AlexNet is used to compute the features



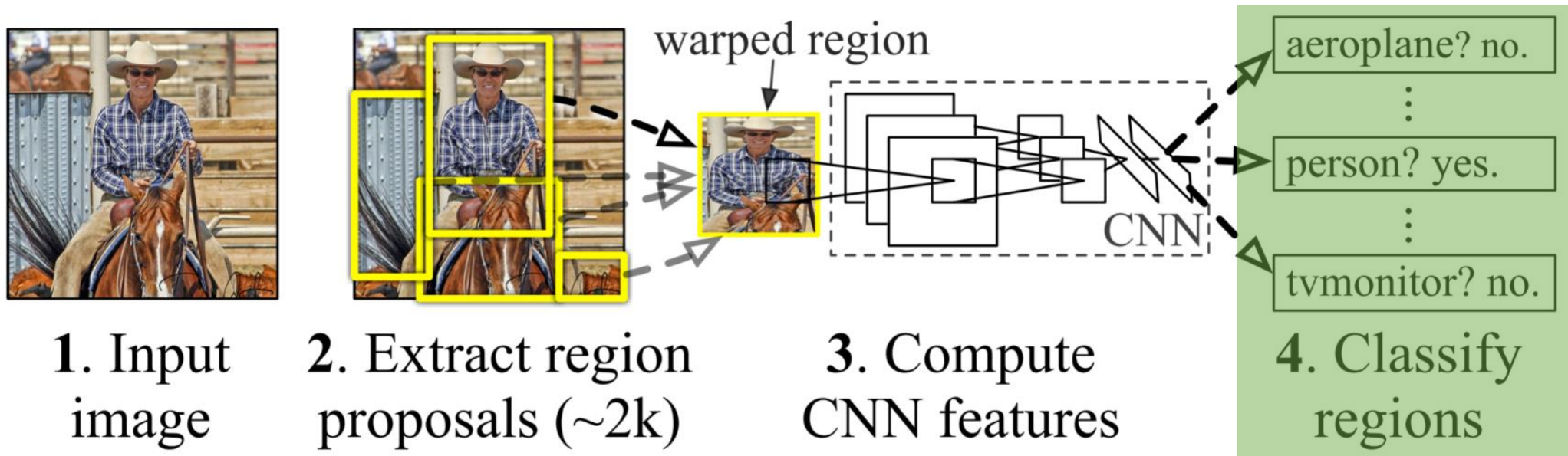
Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**
- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
 - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- **(3) computes features for each region using a large convolutional neural network (CNN)**
 - AlexNet is used to compute the features (**227×227 pixels**)



Region-Based CNNs (R-CNNs)

- **R-CNN - 2014**
- (1) takes an input image
- (2) extracts around 2000 bottom-up regions using selective search
 - J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013
- (3) computes features for each region using a large convolutional neural network (CNN)
 - AlexNet is used to compute the features (227×227 pixels)
- **(4) classifies each region using class-specific linear SVMs**

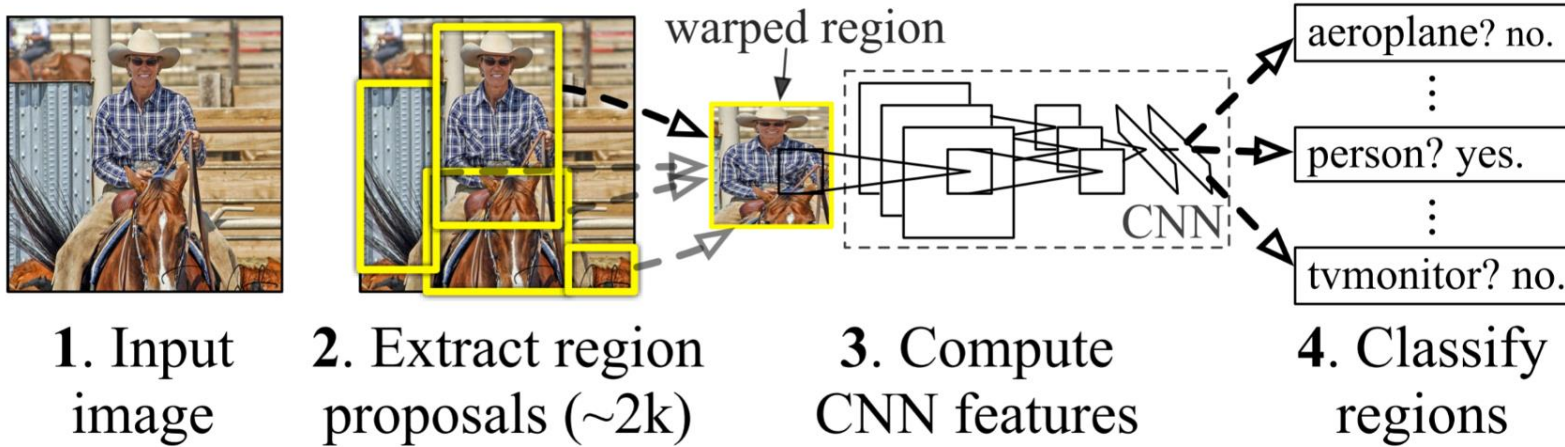




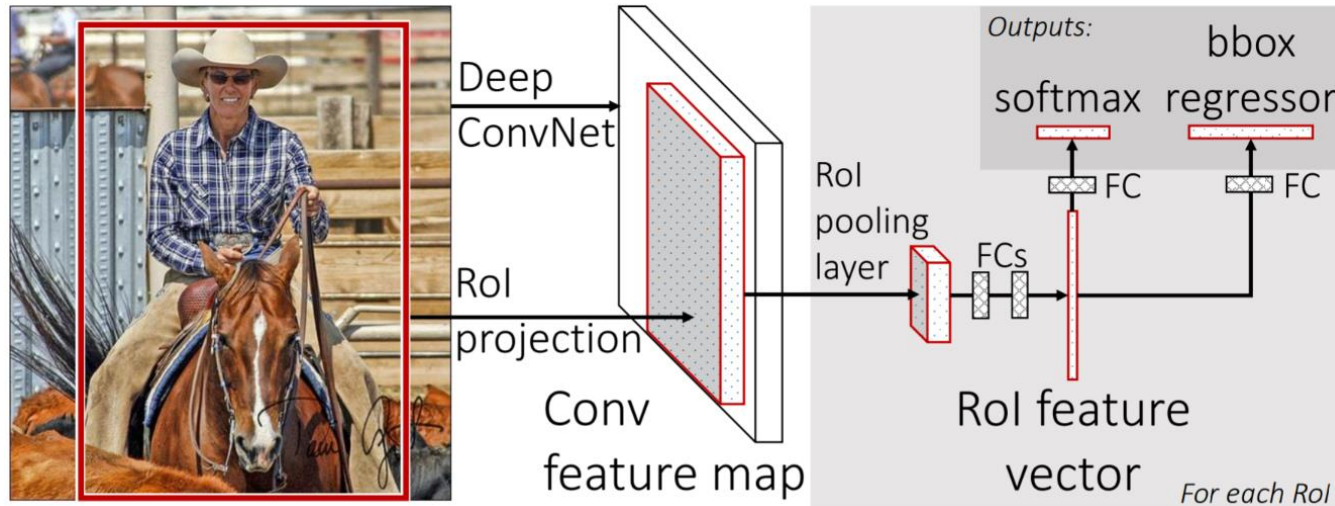
Region-Based CNNs (Fast R-CNNs)

- **R-CNN vs. Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape

R-CNN



Fast R-CNN

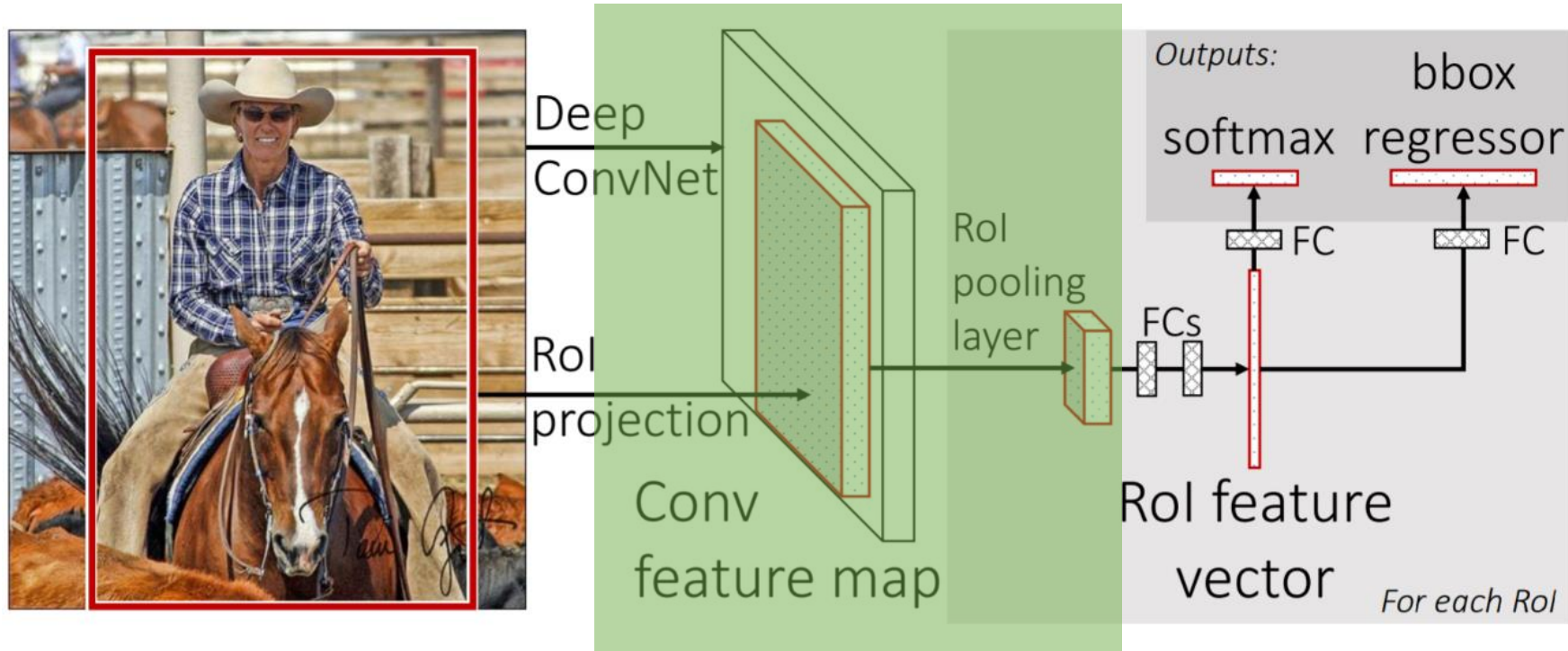


[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

Region-Based CNNs (Fast R-CNNs)

- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape
- **ROI Pooling (Region of interest pooling)** solves the problem
 - for every ROI (**proposal**) from the input, feature map which corresponds to that ROI is selected
 - transform this feature-map into a fixed dimension map



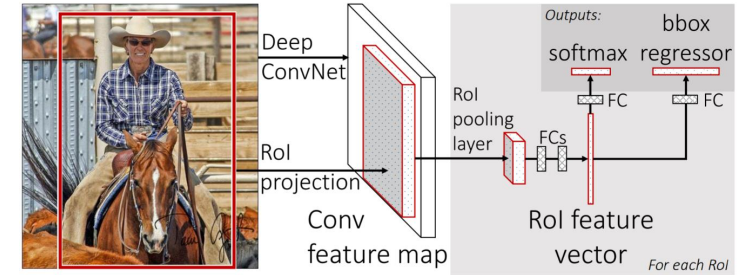
[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

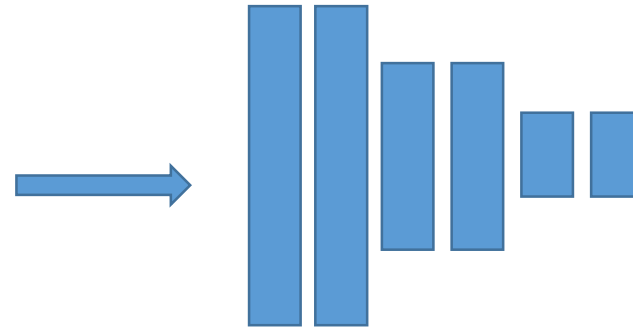


Region-Based CNNs (Fast R-CNNs)

- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape
- Example ([1, 2]):
 - input image size is **1056x640**
 - after several conv and pool operations the output feature map size is **reduced to 66x40**
 - **this feature map is used by ROI pooling layer**



1056x640



VGG-16



66x40 feature map

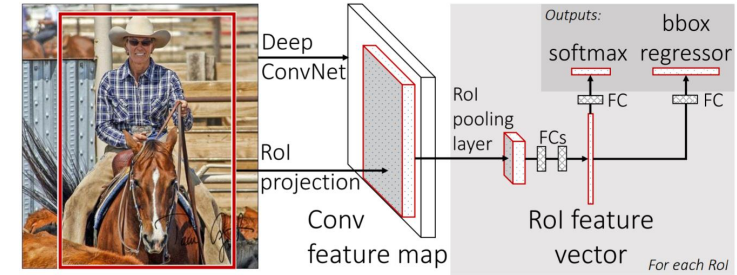
[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

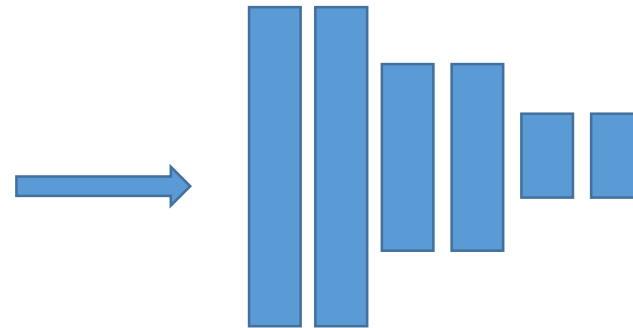


Region-Based CNNs (Fast R-CNNs)

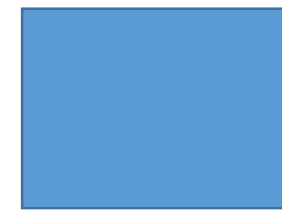
- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape
- Example ([1, 2]):
 - input image size is **1056x640**
 - after several conv and pool operations the output feature map size is **reduced to 66x40**
 - **this feature map is used by ROI pooling layer**
 - **Get Rols from the feature map?**



1056x640



VGG-16



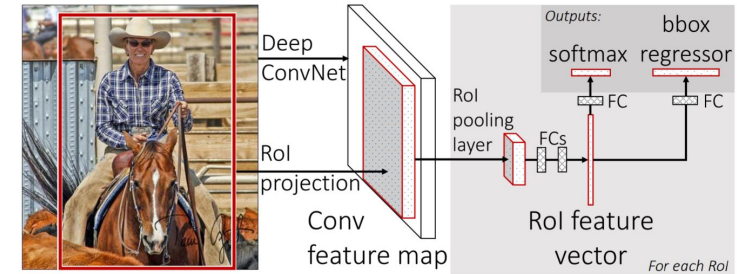
66x40 feature map

[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

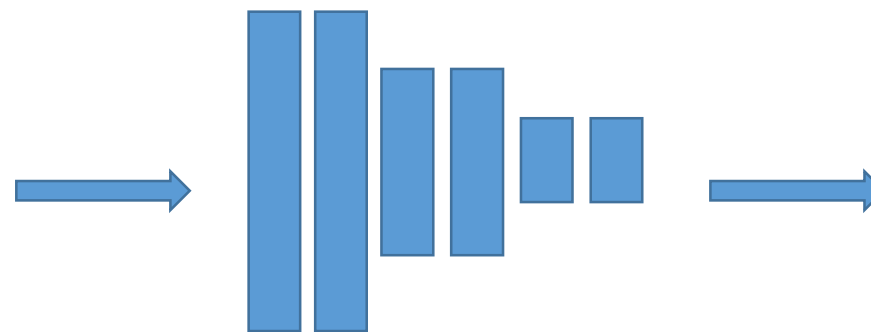
[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

Region-Based CNNs (Fast R-CNNs)

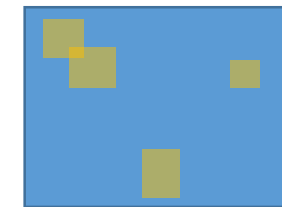
- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape
- Example ([1, 2]): input image size is **1056x640**
 - after several conv and pool operations the output feature map size is **reduced to 66x40**
 - **this feature map is used by ROI pooling layer**
 - **Get Rols from the feature map?**
 - The extracted regions of interest (**proposal**) are generated based on input image size, so we need to rescale these regions to feature map size. In this particular case by 16 ($1056/66=16$ or $640/40=16$).



1056x640



VGG-16



66x40 feature map

[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

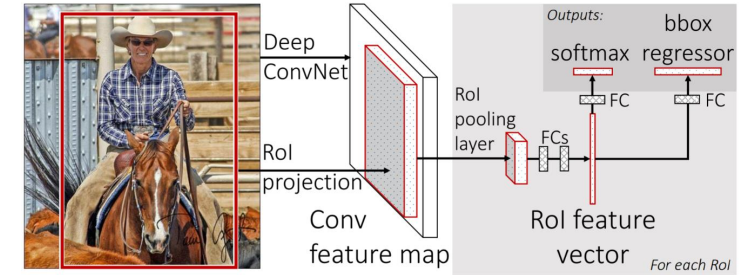
Region-Based CNNs (Fast R-CNNs)

- **Fast R-CNN – 2015**

- different shapes of regions > fully connected layers require fixed shape

- Example ([1, 2]):

- input image size is **1056x640**
- after several conv and pool operations the output feature map size is **reduced to 66x40**
- **this feature map is used by ROI pooling layer**
- **Get Rols from the feature map?**
- The extracted regions of interest (**proposal**) are generated based on input image size, so we need to rescale these regions to feature map size. In this particular case by 16 (1056/66=16 or 640/40=16).
- **For every proposal in the input proposals, we take the corresponding feature map section and divide that section into W*H. After that take the maximum element of each block and copy to the output. So as the output we obtain fixed dimension feature map irrespective of the various sizes of the input proposals.**



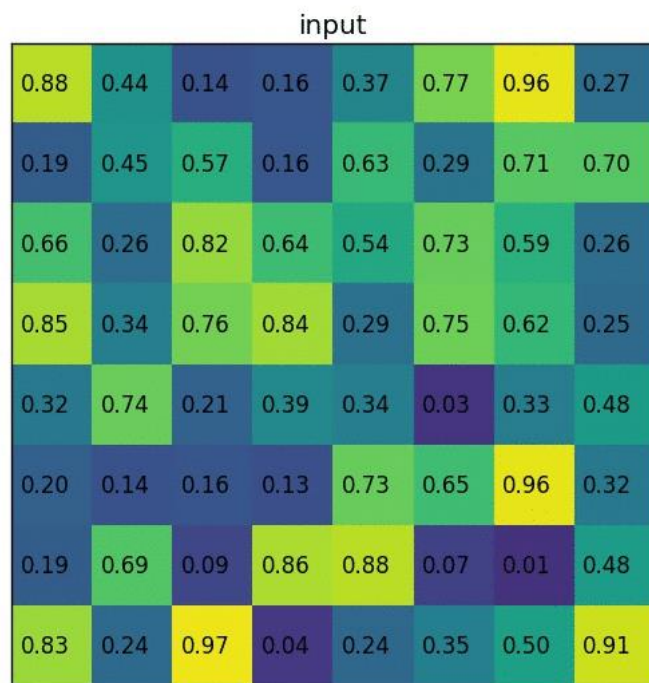
```
Scaled_Proposals = Proposals * spatial_scale
for every ROI in Scaled_Proposals:
    fmap_subset = feature_map[ROI] (Feature_map for that ROI)
    Divide fmap_subset into P_wxP_h blocks (ex: 6*6 blocks)
    Take the maximum element of each block and copy to output block
```

[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

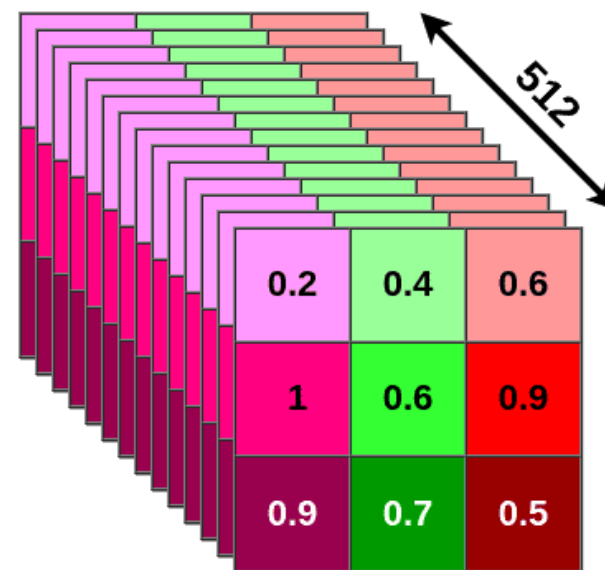
[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

Region-Based CNNs (Fast R-CNNs)

- **Fast R-CNN – 2015**
- different shapes of regions > fully connected layers require fixed shape
- **ROI Pooling (Region of interest pooling)** solves the problem



3x3 RoI Pooling (full size)



After the pooling process, (for example) the 3x3x512 matrixes can be used as input for FC layers for further processing. For each region we obtain fixed size of vector.

source: https://miro.medium.com/max/840/1*5V5myclRNU-mK-rPywL57w.gif

[1] <https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af>

[2] <https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

Region-Based CNNs (Faster R-CNNs)

- **Faster R-CNN – 2015**
- Selective search in R-CNN and Fast R-CNN is replaced by **Region Proposal Network**
- two modules:
 - 1. module is a deep fully convolutional network that proposes regions
 - 2. module is the Fast R-CNN detector that uses the proposed regions

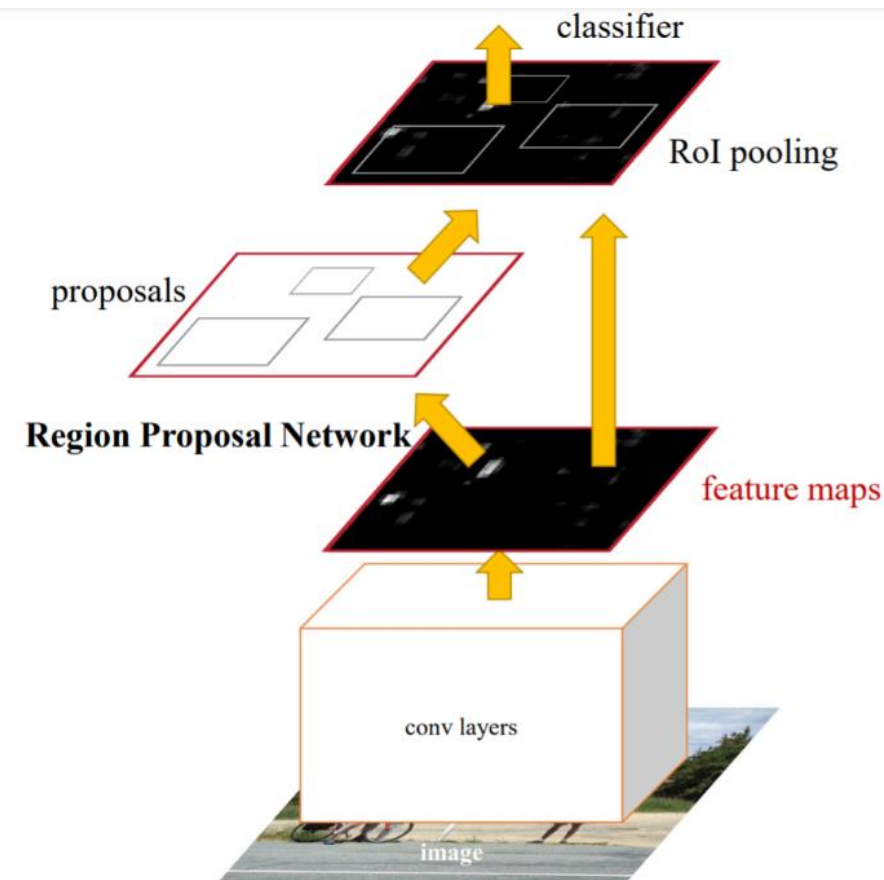
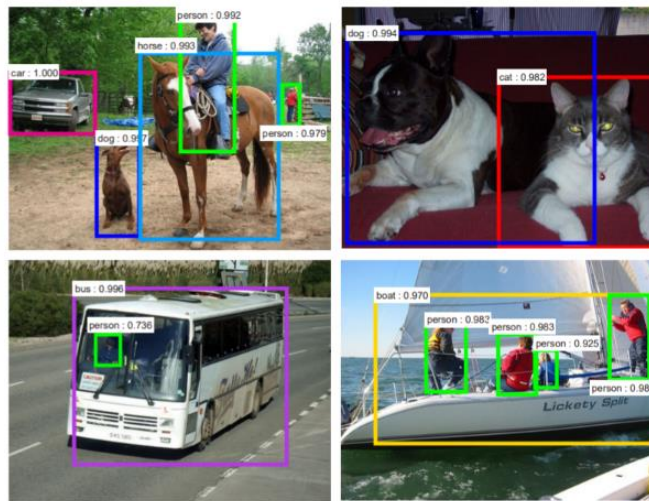
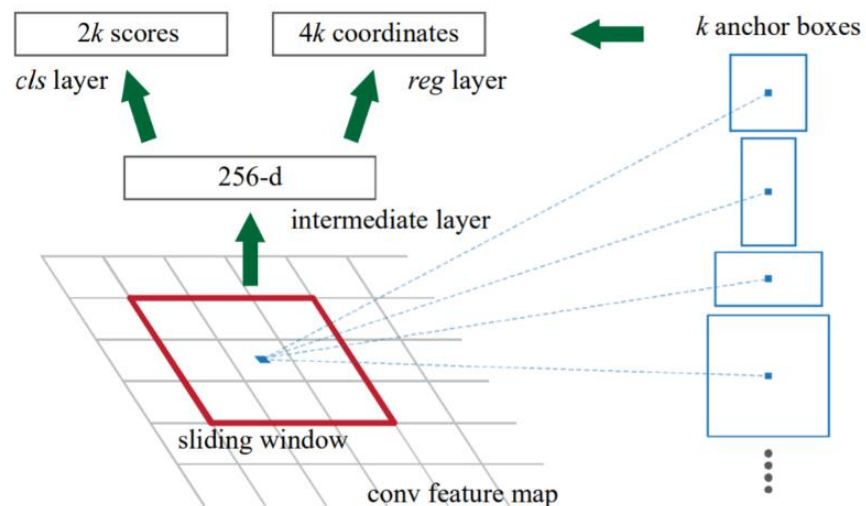
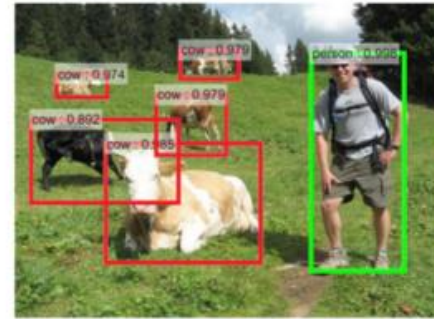
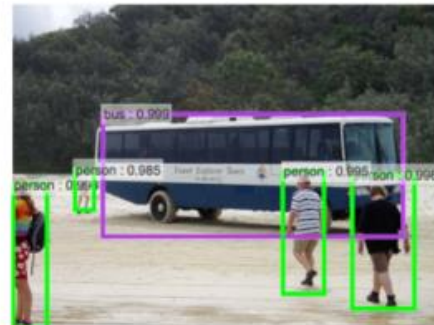
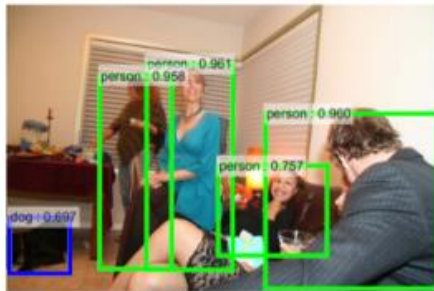
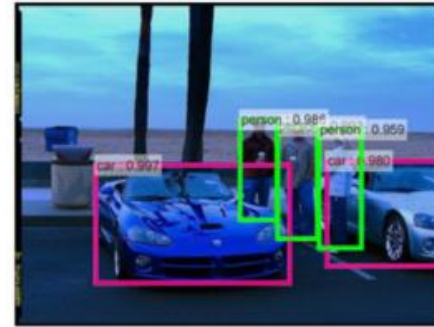
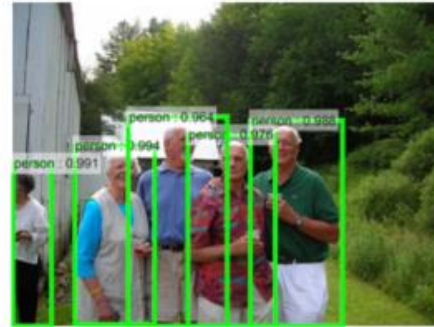
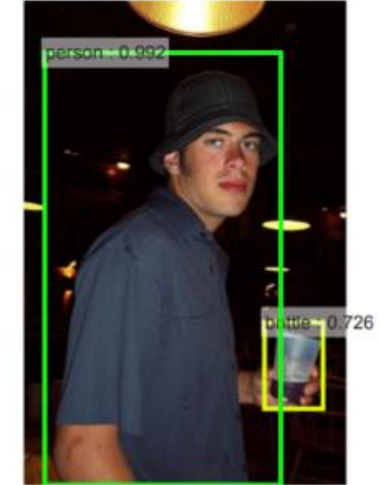
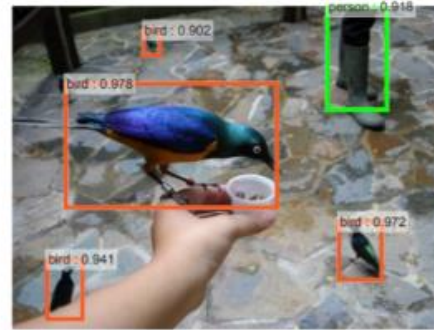
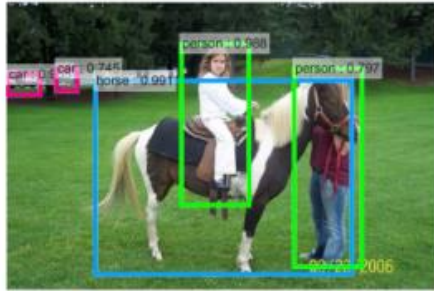


Figure 3: **Left:** Region Proposal Network (RPN). **Right:** Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.



Region-Based CNNs (Faster R-CNNs)

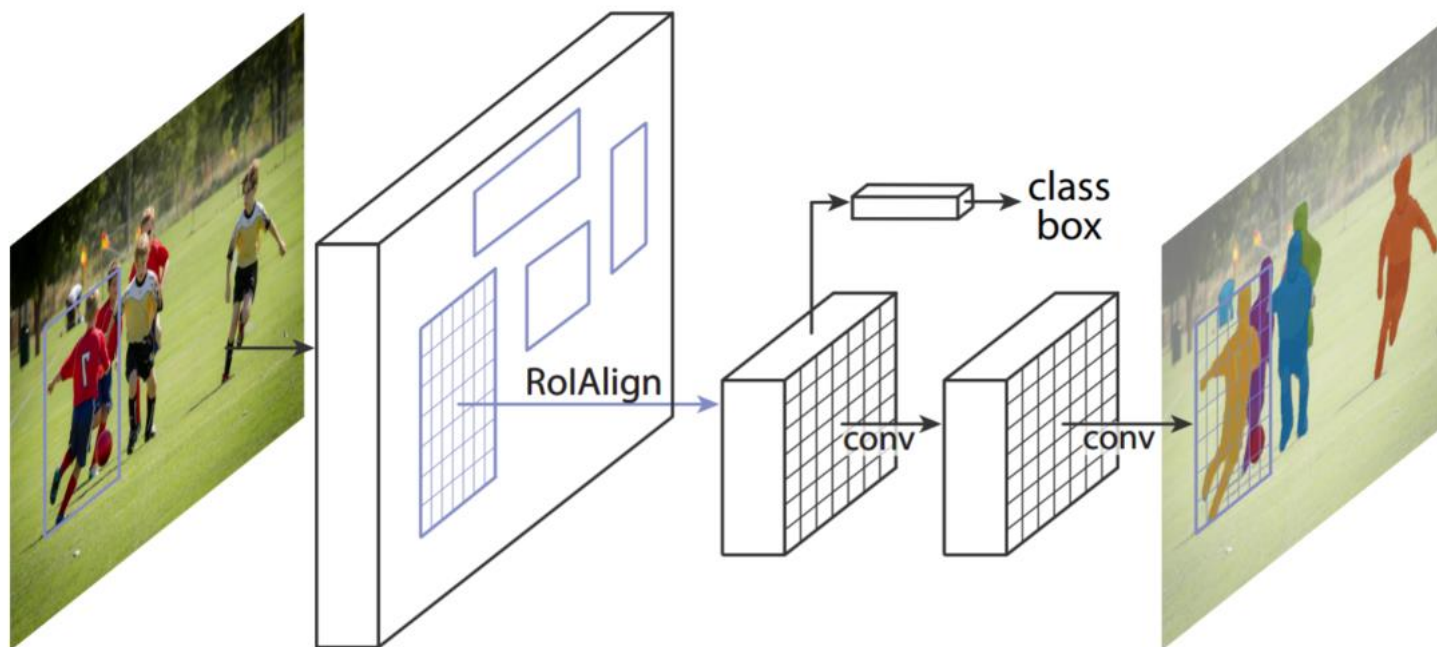
- Faster R-CNN – 2015





Region-Based CNNs (Mask R-CNNs)

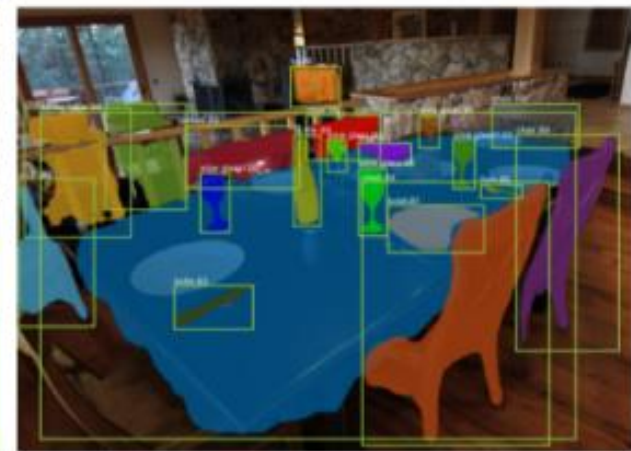
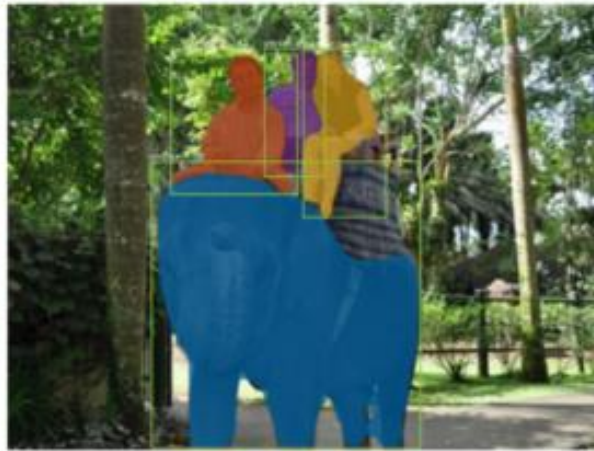
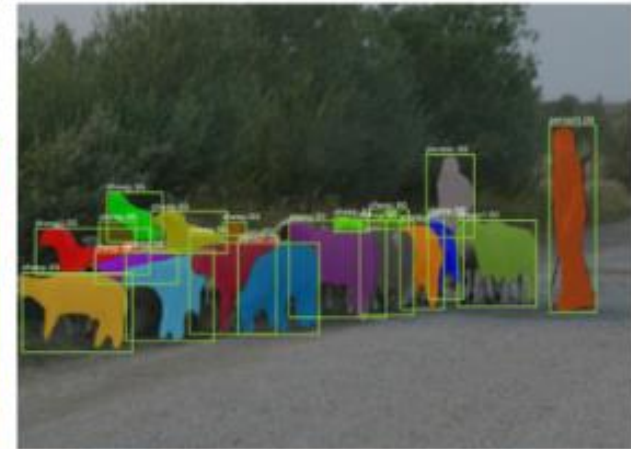
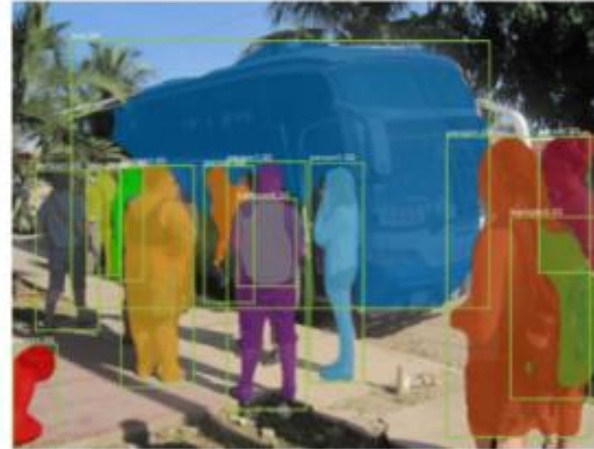
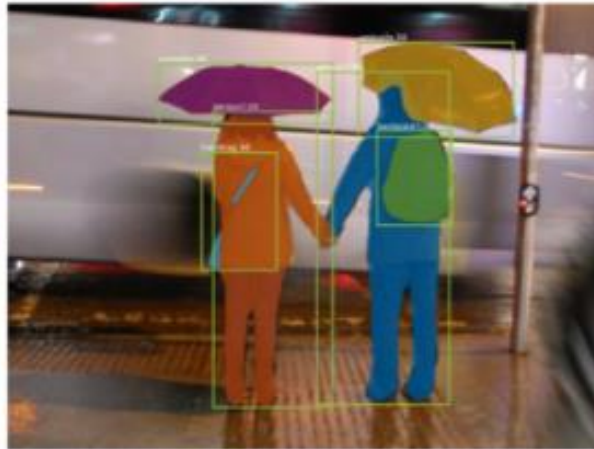
- **Mask R-CNN – 2017**
- **Extends Faster R-CNN**
 - With the use of branch for predicting segmentation masks on each Region of Interest (RoI)





Region-Based CNNs (Mask R-CNNs)

- Mask R-CNN – 2017





Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**

https://pytorch.org/vision/main/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html

```
def main():

    cv2.namedWindow("detection", 0)
    print("main")

    test_images = [img for img in glob.glob("test_images/*.jpg")]
    test_images.sort()

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.eval().to(device)

    transformRCNN = transforms.Compose([
        transforms.ToTensor(),
    ])
```

<https://arxiv.org/abs/1506.01497>

<https://arxiv.org/abs/1703.06870>



Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**

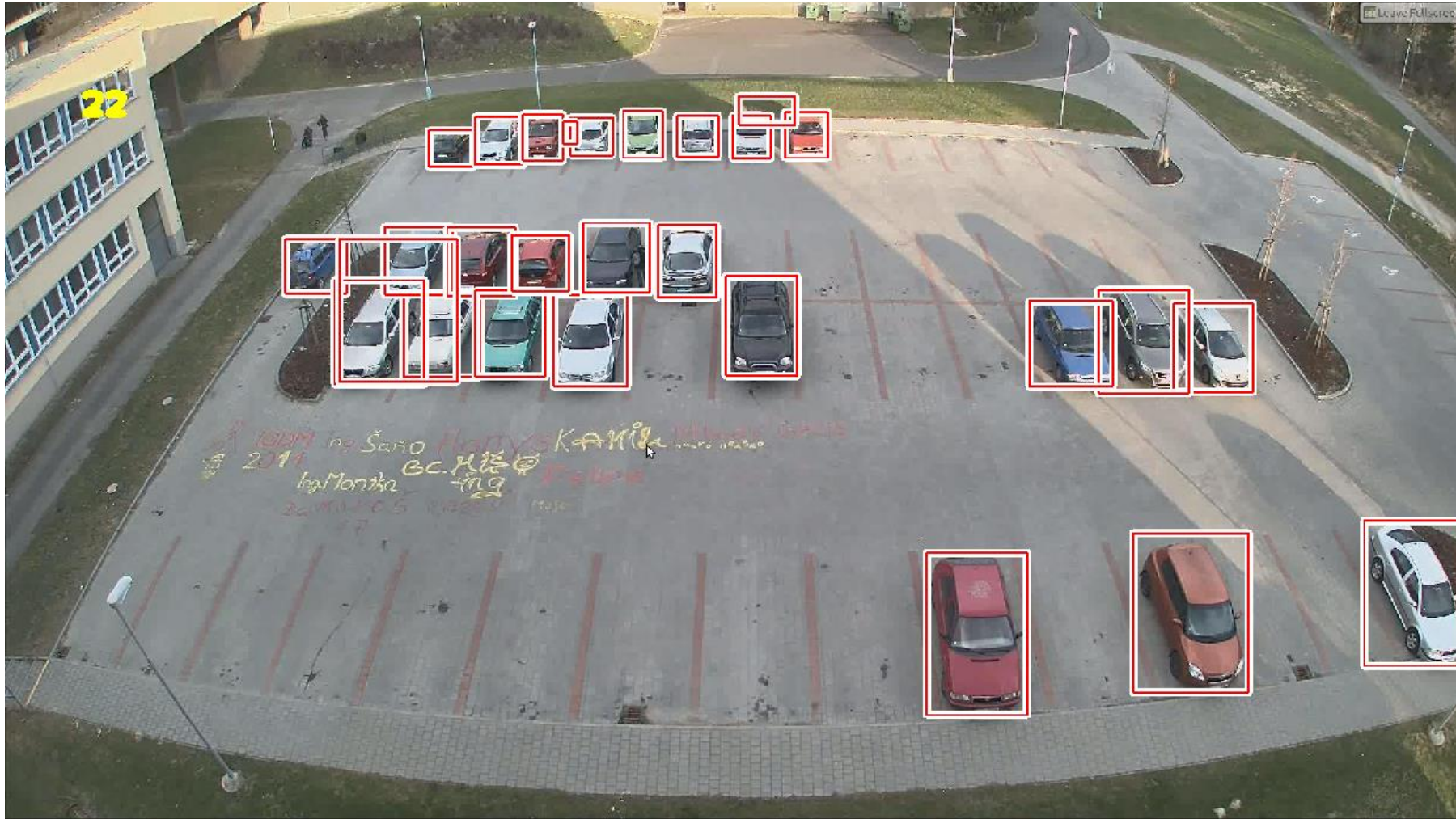
```
coco_names = ['__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck',  
'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',  
'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A', 'handbag', 'tie', 'suitcase',  
'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',  
'tennis racket', 'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich',  
'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining  
table', 'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven',  
'toaster', 'sink', 'refrigerator', 'N/A', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']
```

```
for img in test_images:  
    one_img = cv2.imread(img)  
    one_img_paint = one_img.copy()  
  
    one_img_rgb = cv2.cvtColor(one_img, cv2.COLOR_BGR2RGB)  
    img_pil = Image.fromarray(one_img_rgb)  
    imageRCNN = transformRCNN(img_pil).to(device)  
    imageRCNN = imageRCNN.unsqueeze(0)  
    outputsRCNN = model(imageRCNN)  
    pred_classes = [coco_names[i] for i in outputsRCNN[0]['labels'].cpu().numpy()]  
    pred_scores = outputsRCNN[0]['scores'].detach().cpu().numpy()  
    pred_bboxes = outputsRCNN[0]['boxes'].detach().cpu().numpy()  
  
    print(pred_scores)  
    print(pred_classes)
```



Region-Based CNNs (Faster R-CNNs)

- **EXAMPLE Faster R-CNN – PyTorch**



<https://arxiv.org/abs/1506.01497>

<https://arxiv.org/abs/1703.06870>

- YOLO - You Only Look Once**

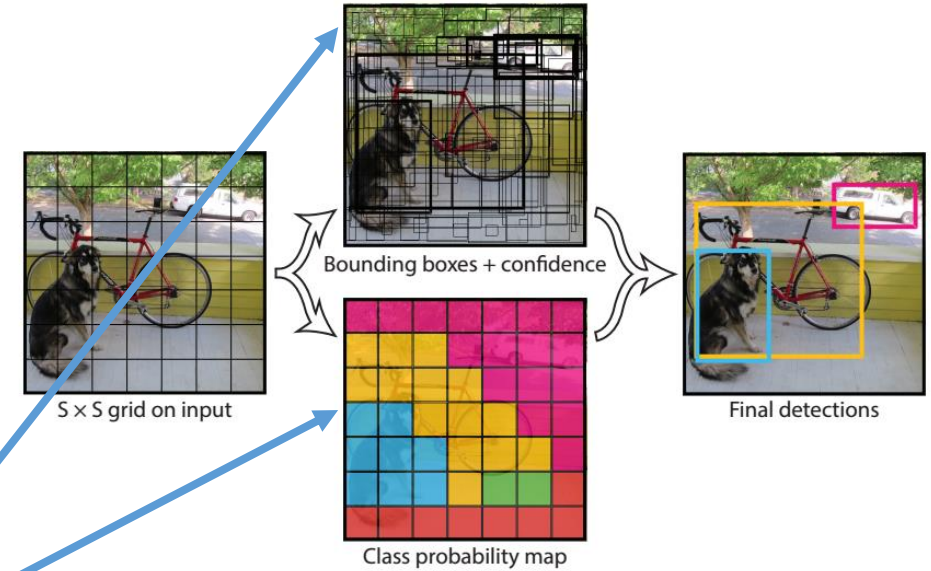
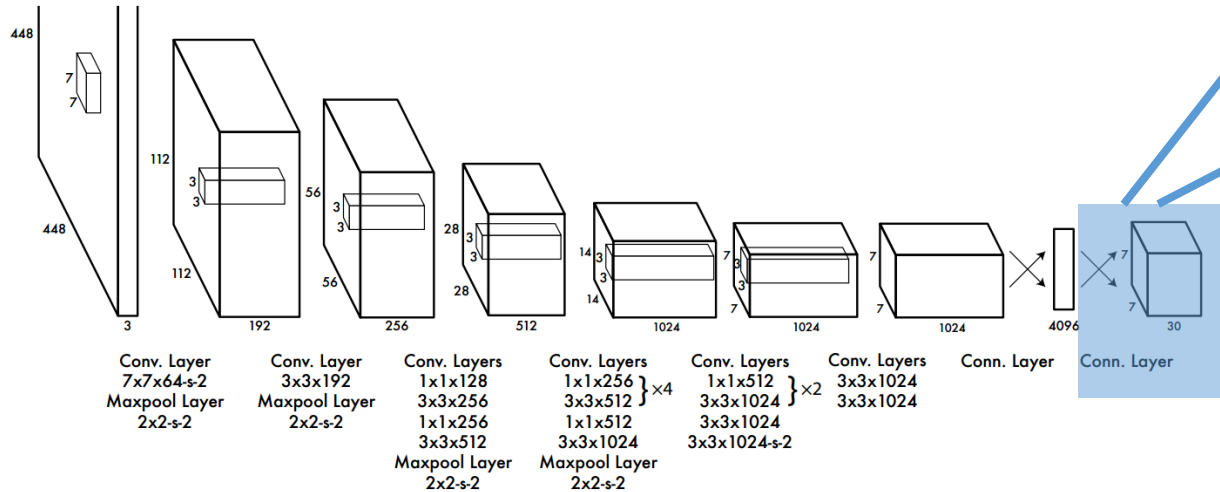


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

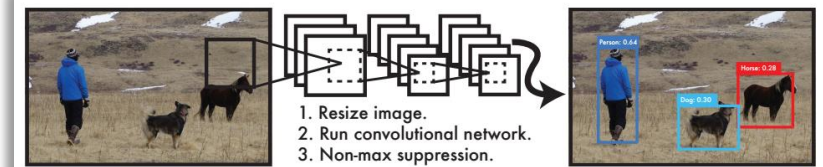


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

YOLO vs. Fast R-CNN

- YOLO - You Only Look Once

Figure 4 shows the breakdown of each error type averaged across all 20 classes.

YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of its top detections are false positives that don't contain any objects. Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

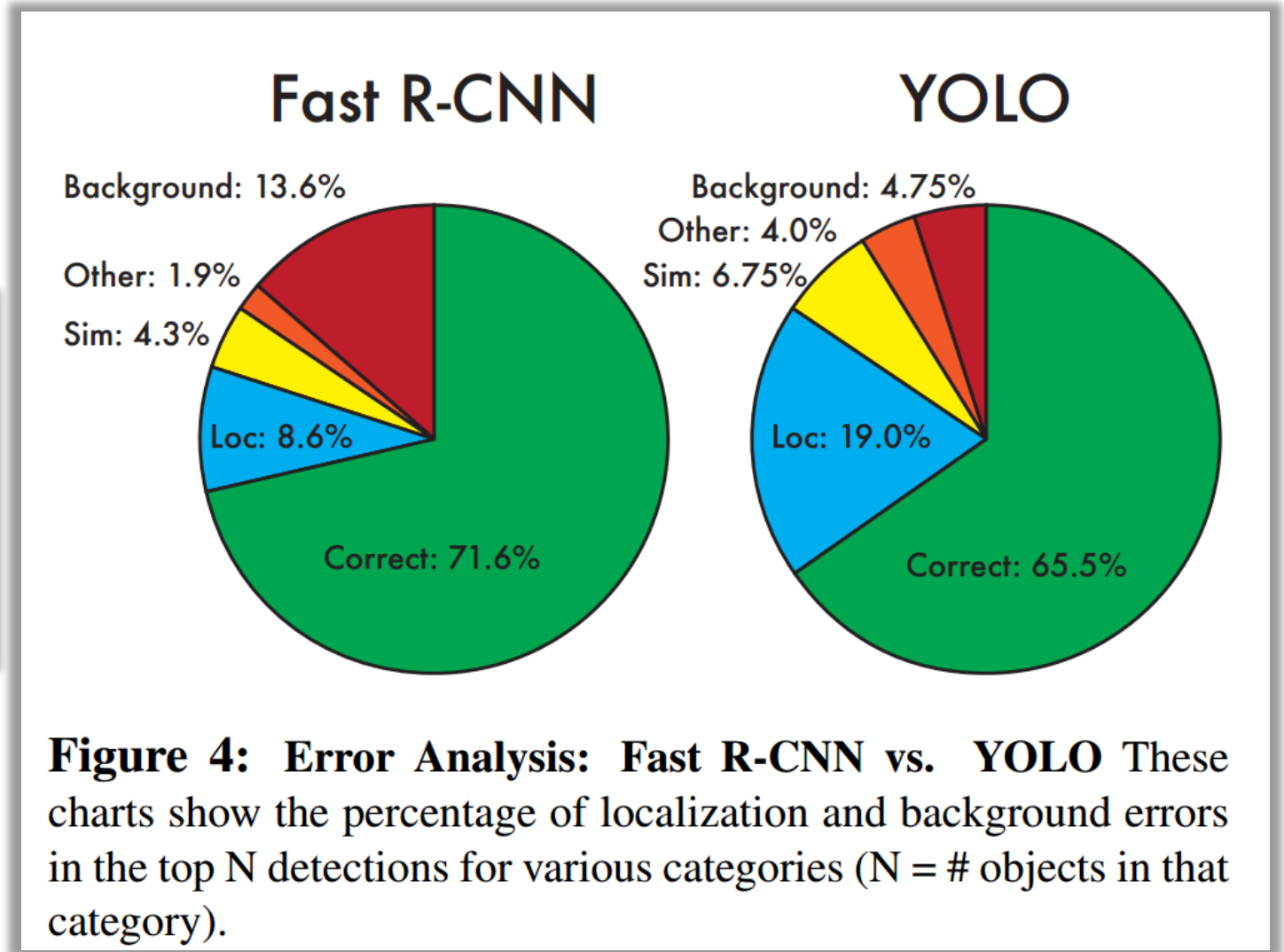


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

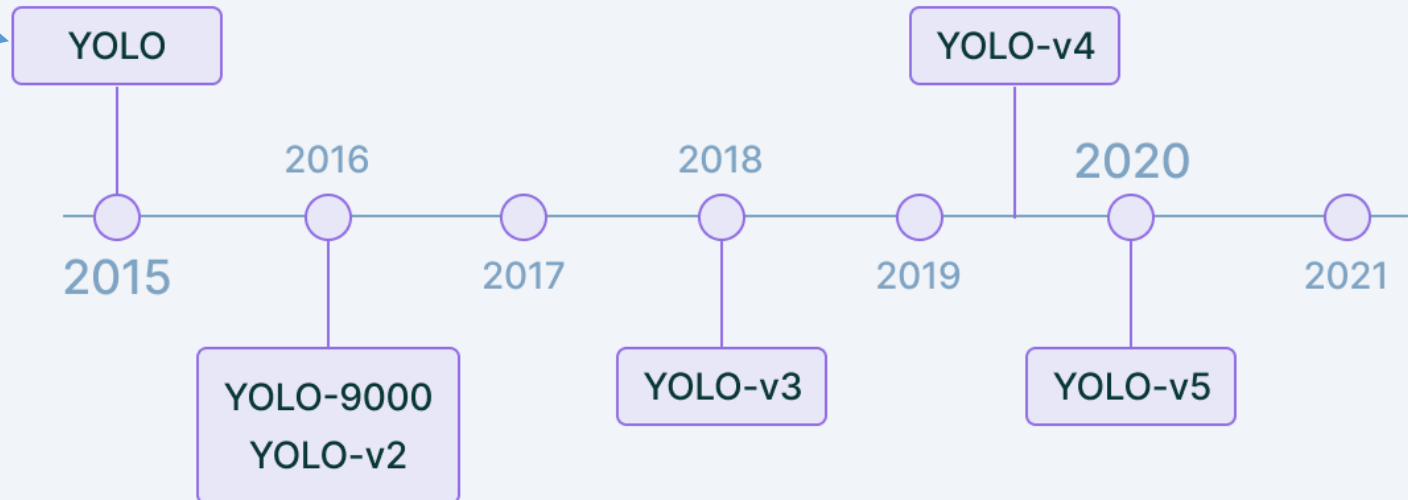


Versions of YOLO

2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

YOLO timeline



V7 Labs

<https://www.v7labs.com/blog/yolo-object-detection>



MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard
Weijun Wang

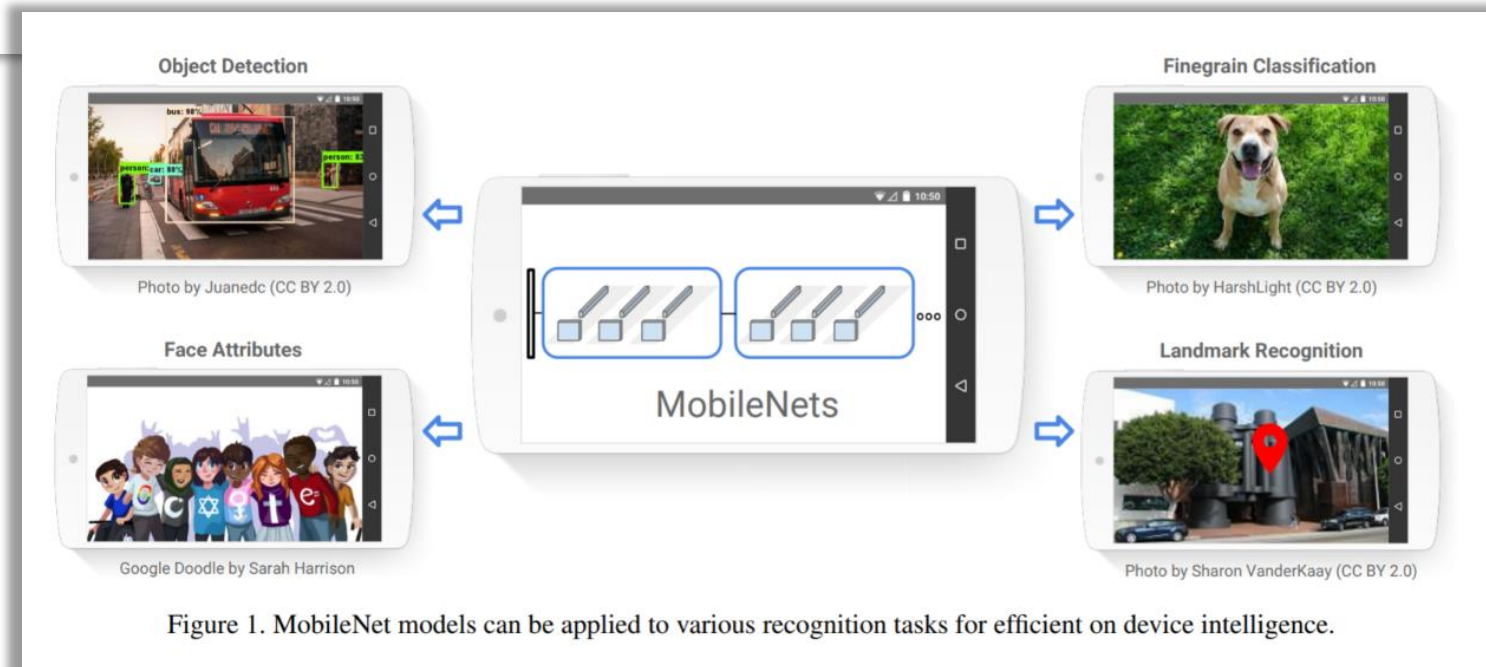
Menglong Zhu
Tobias Weyand

Bo Chen
Marco Andreetto

Dmitry Kalenichenko
Hartwig Adam

Google Inc.

{howarda, menglong, bochen, dkalenichenko, weijunw, weyand, anm, hadam}@google.com



Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \quad (2)$$

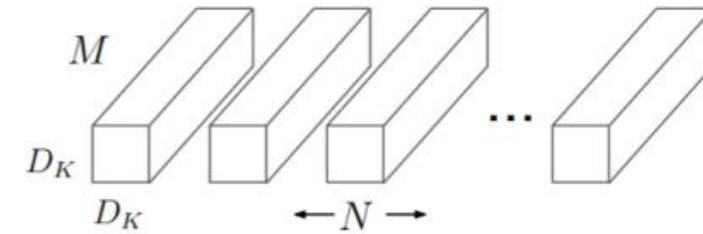
where the computational cost depends multiplicatively on the number of input channels M , the number of output channels N the kernel size $D_k \times D_k$ and the feature map size $D_F \times D_F$. MobileNet models address each of these terms and their interactions. First it uses depthwise separable convolutions to break the interaction between the number of output channels and the size of the kernel.

Depthwise separable convolutions cost:

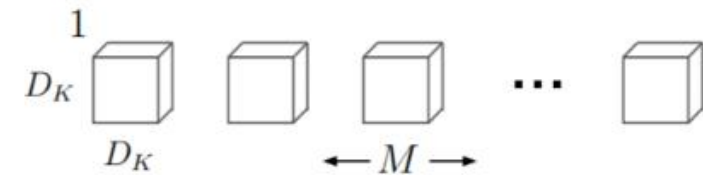
$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad (5)$$

which is the sum of the depthwise and 1×1 pointwise convolutions.

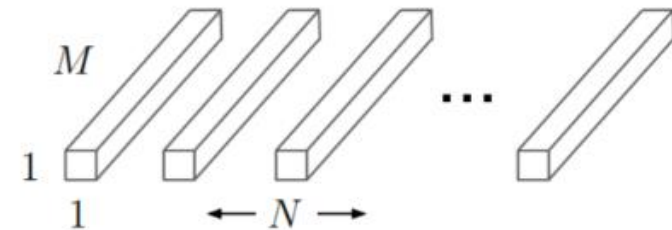
MobileNet uses 3×3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy as seen in Section 4.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.



Mobile Devices

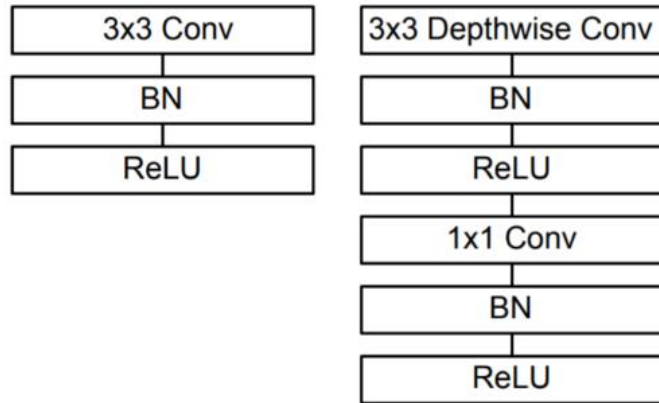


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$



Mobile Devices

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138



PYTORCH MOBILE

End-to-end workflow from Training to Deployment for iOS and Android mobile devices

Home

iOS

Android

Shortcuts

Quickstart with a HelloWorld Example

PyTorch Demo Application

More PyTorch Android Demo Apps

PyTorch Android Tutorial and Recipes

Building PyTorch Android from Source

Using the PyTorch Android Libraries Built

from Source or Nightly

Using the Nightly PyTorch Android

Libraries

Custom Build

Use PyTorch JIT interpreter

Android Tutorials

API Docs

Android

QUICKSTART WITH A HELLOWORLD EXAMPLE

HelloWorld is a simple image classification application that demonstrates how to use PyTorch Android API. This application runs TorchScript serialized TorchVision pretrained resnet18 model on static image which is packaged inside the app as android asset.

1. Model Preparation

Let's start with model preparation. If you are familiar with PyTorch, you probably should already know how to train and save your model. In case you don't, we are going to use a pre-trained image classification model (**MobileNetV2**). To install it, run the command below:

```
pip install torchvision
```

To serialize the model you can use python `script` in the root folder of HelloWorld app:

<https://pytorch.org/mobile/android/>



Quickstart

HelloWorld is a simple image classification application that demonstrates how to use PyTorch Android API. This application runs TorchScript serialized TorchVision pretrained **MobileNet v3 model** on static image which is packaged inside the app as android asset.

1. Model Preparation

Let's start with model preparation. If you are familiar with PyTorch, you probably should already know how to train and save your model. In case you don't, we are going to use a pre-trained image classification model(MobileNet v3), which is packaged in **TorchVision**. To install it, run the command below:

```
pip install torch torchvision
```

To serialize and optimize the model for Android, you can use the Python **script** in the root folder of HelloWorld app:

```
import torch
import torchvision
from torch.utils.mobile_optimizer import optimize_for_mobile

model = torchvision.models.mobilenet_v3_small(pretrained=True)
model.eval()
example = torch.rand(1, 3, 224, 224)
traced_script_module = torch.jit.trace(model, example)
optimized_traced_model = optimize_for_mobile(traced_script_module)
optimized_traced_model._save_for_lite_interpreter("app/src/main/assets/model.ptl")
```

If everything works well, we should have our scripted and optimized model - `model.pt` generated in the assets folder of android application. That will be packaged inside android application as `asset` and can be used on the device.

<https://github.com/pytorch/android-demo-app/tree/master/HelloWorldApp>



Quickstart

HelloWorld is a simple image classification application that demonstrates how to use PyTorch Android API. This application runs TorchScript serialized TorchVision pretrained **MobileNet v3 model** on static image which is packaged inside the app as android asset.

1. Model Preparation

Let's start with model preparation. If you are familiar with PyTorch, you probably should already know how to train and save your model. In case you don't, we are going to use a pre-trained image classification model(MobileNet v3), which is packaged in **TorchVision**. To install it, run the command below:

```
pip install torch torchvision
```

To serialize and optimize the model for Android, you can use the Python **script** in the root folder of HelloWorld app:

```
import torch
import torchvision
from torch.utils.mobile_optimizer import optimize_for_mobile

model = torchvision.models.mobilenet_v3_small(pretrained=True)
model.eval()
example = torch.rand(1, 3, 224, 224)
traced_script_module = torch.jit.trace(model, example)
optimized_traced_model = optimize_for_mobile(traced_script_module)
optimized_traced_model._save_for_lite_interpreter("app/src/main/assets/model.ptl")
```

If everything works well, we should have our scripted and optimized model - `model.pt` generated in the assets folder of android application. That will be packaged inside android application as `asset` and can be used on the device.

<https://github.com/pytorch/android-demo-app/tree/master/HelloWorldApp>

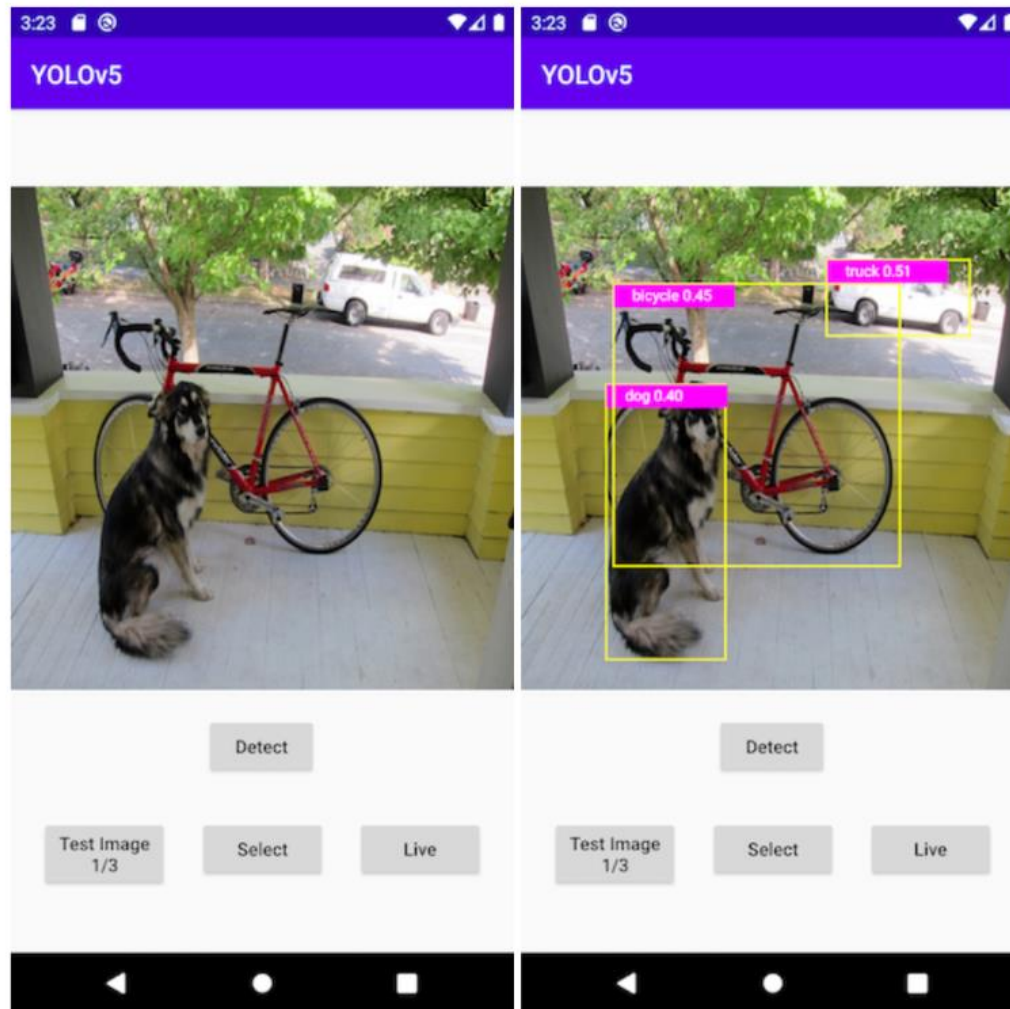


Mobile Devices

☰ README.md

detection - see this [video](#) for a screencast of the app running.

Some example images and the detection results are as follows:



<https://github.com/pytorch/android-demo-app/tree/master/ObjectDetection>