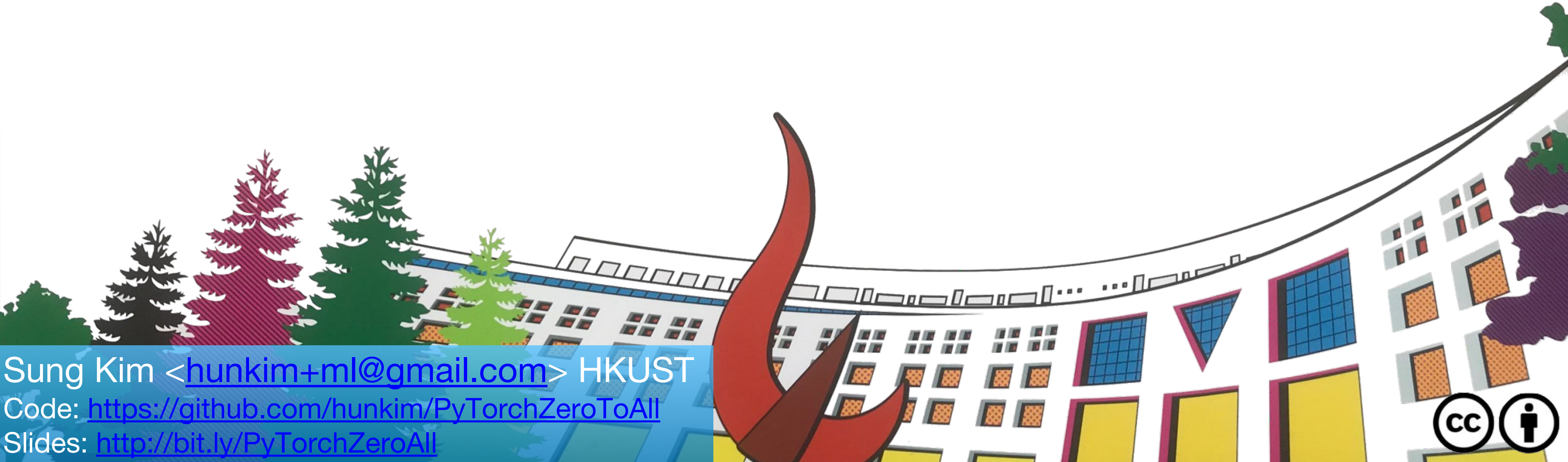
ML/DL for Everyone with PYTORCH

Lecture 10: CNN



Sung Kim <hunkim+ml@gmail.com> HKUST

Code: https://github.com/hunkim/PyTorchZeroToAll

Slides: http://bit.ly/PyTorchZeroAll

Call for Comments

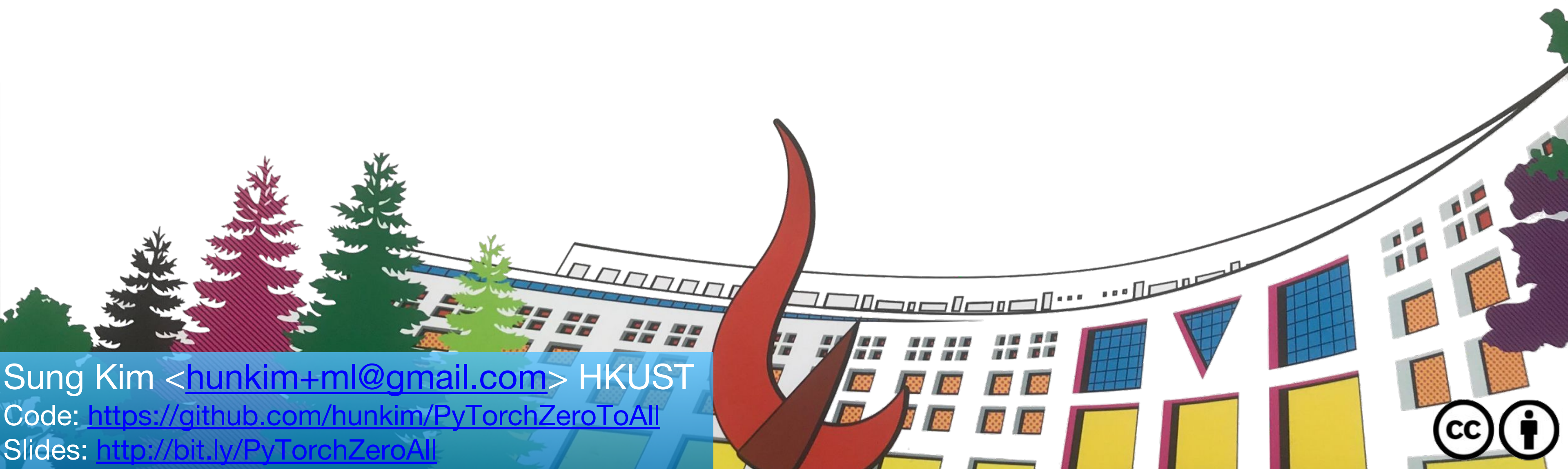
Please feel free to add comments directly on these slides.

Other slides: http://bit.ly/PyTorchZeroAll

Picture from http://www.tssablog.org/archives/3280

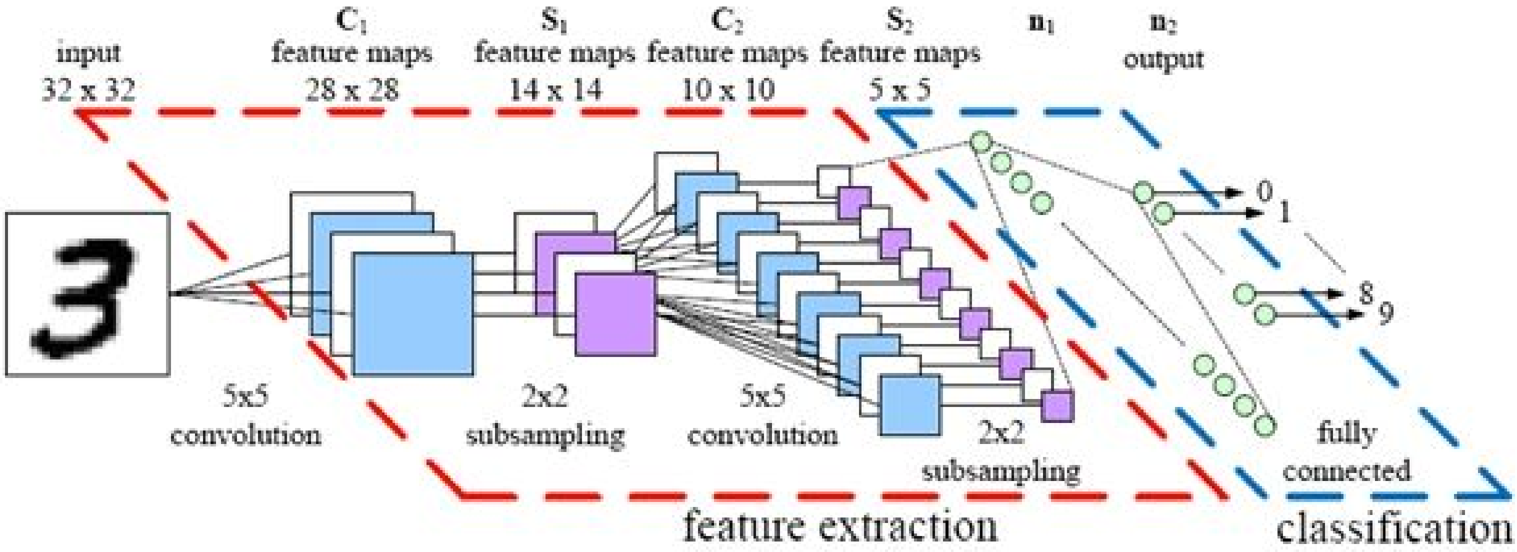
ML/DL for Everyone with PYTORCH

Lecture 10: CNN



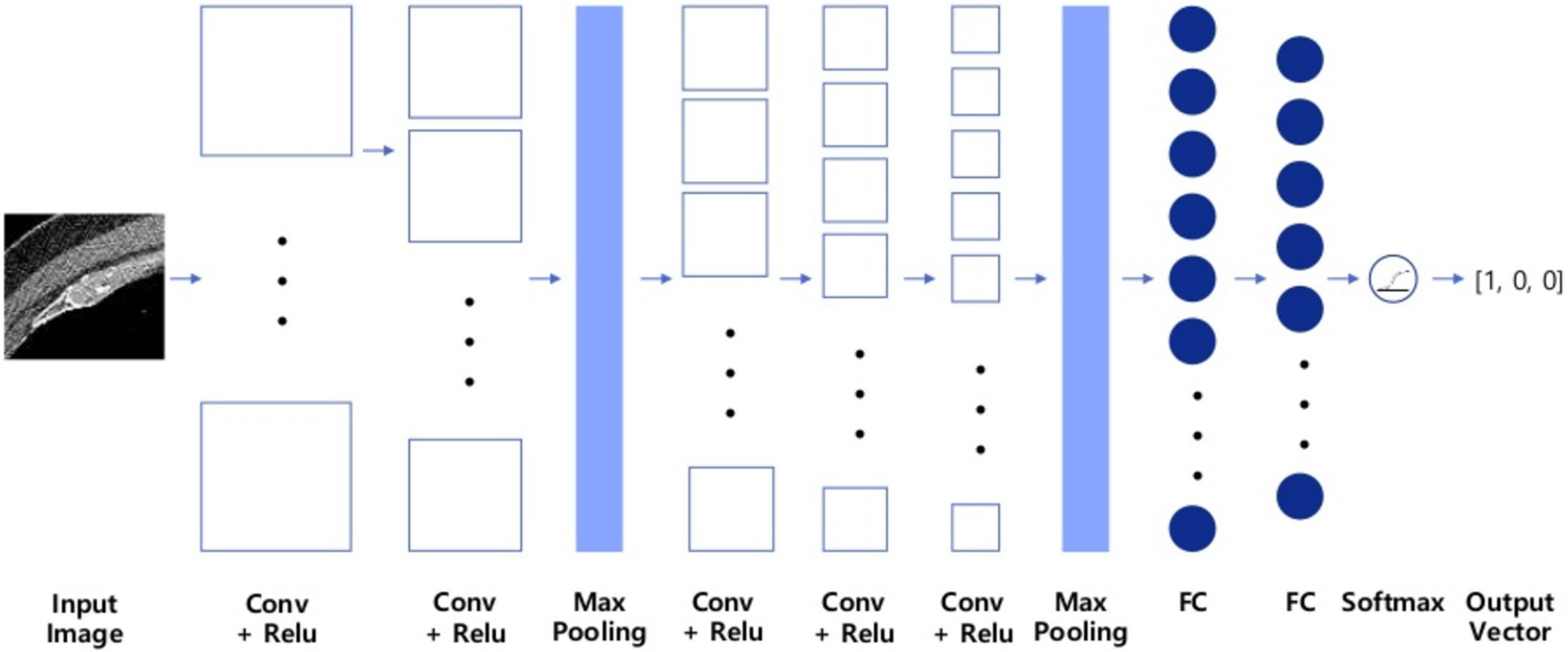
Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

CNN



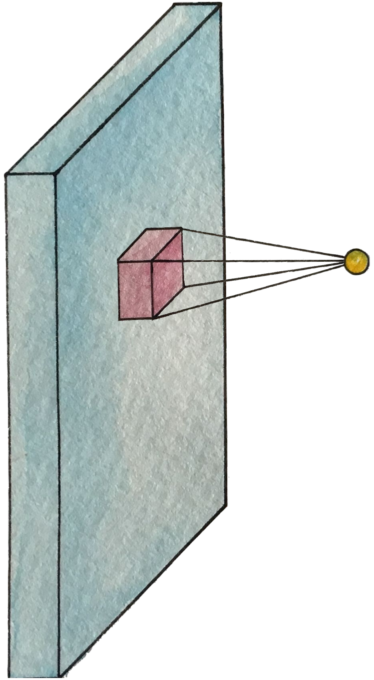
http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

CNN for CT images



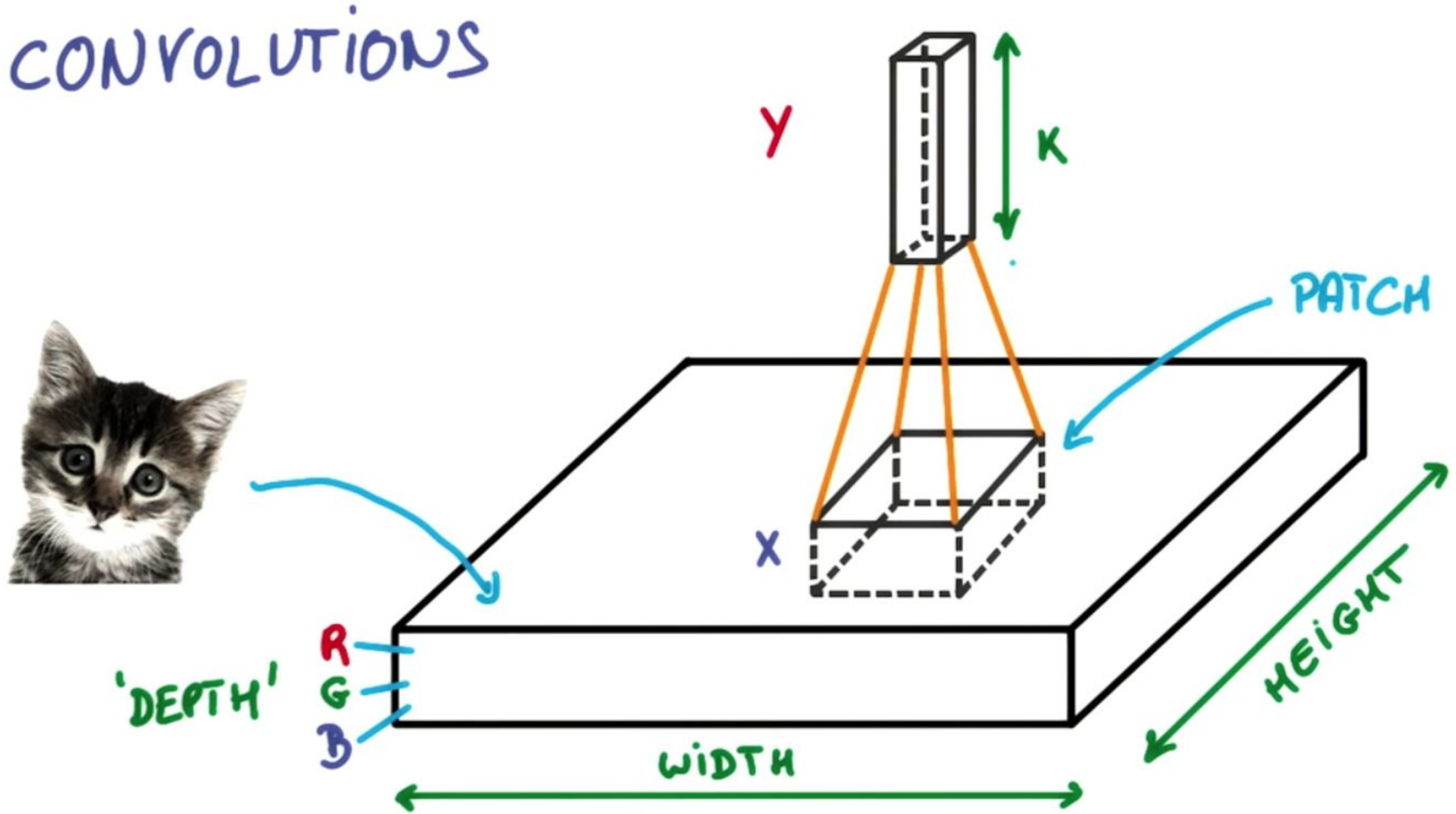
Asan Medical Center & Microsoft Medical Bigdata Contest Winner by GeunYoung Lee and Alex Kim
https://www.slideshare.net/GYLee3/ss-72966495

Convolution layer and max pooling



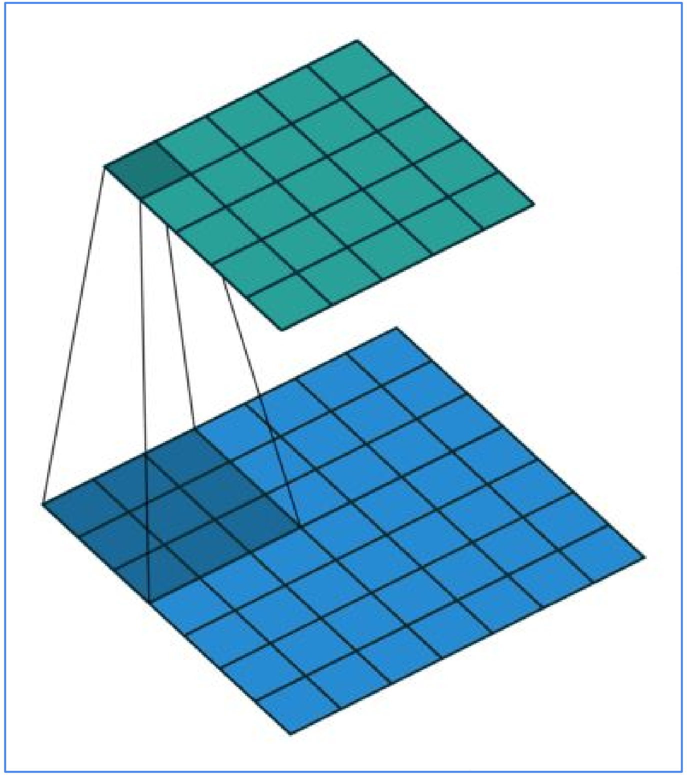
Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4



https://ireneli.eu/2016/02/03/deep-learning-05-talk-about-convolutional-neural-network%EF%BC%88cnn%EF%BC%89/

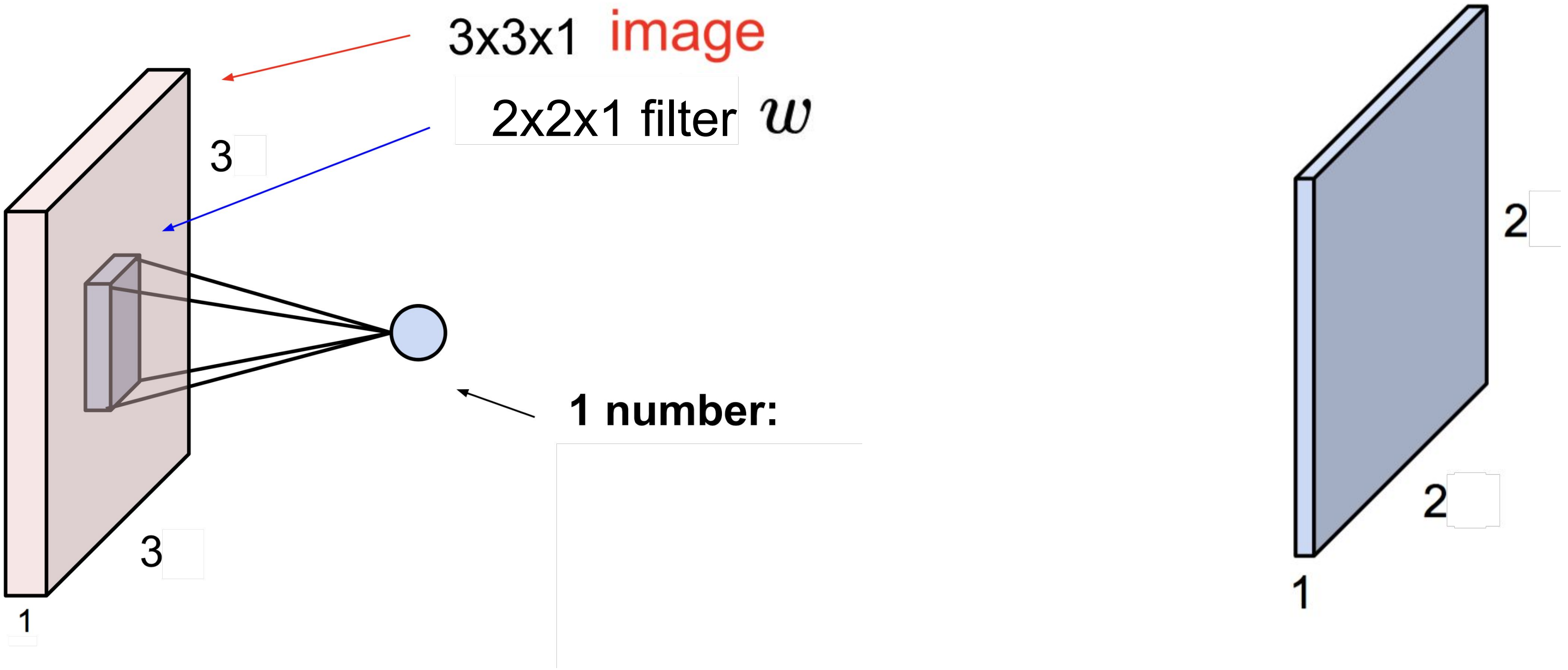
Convolution in Action



https://github.com/vdumoulin/conv_arithmetic

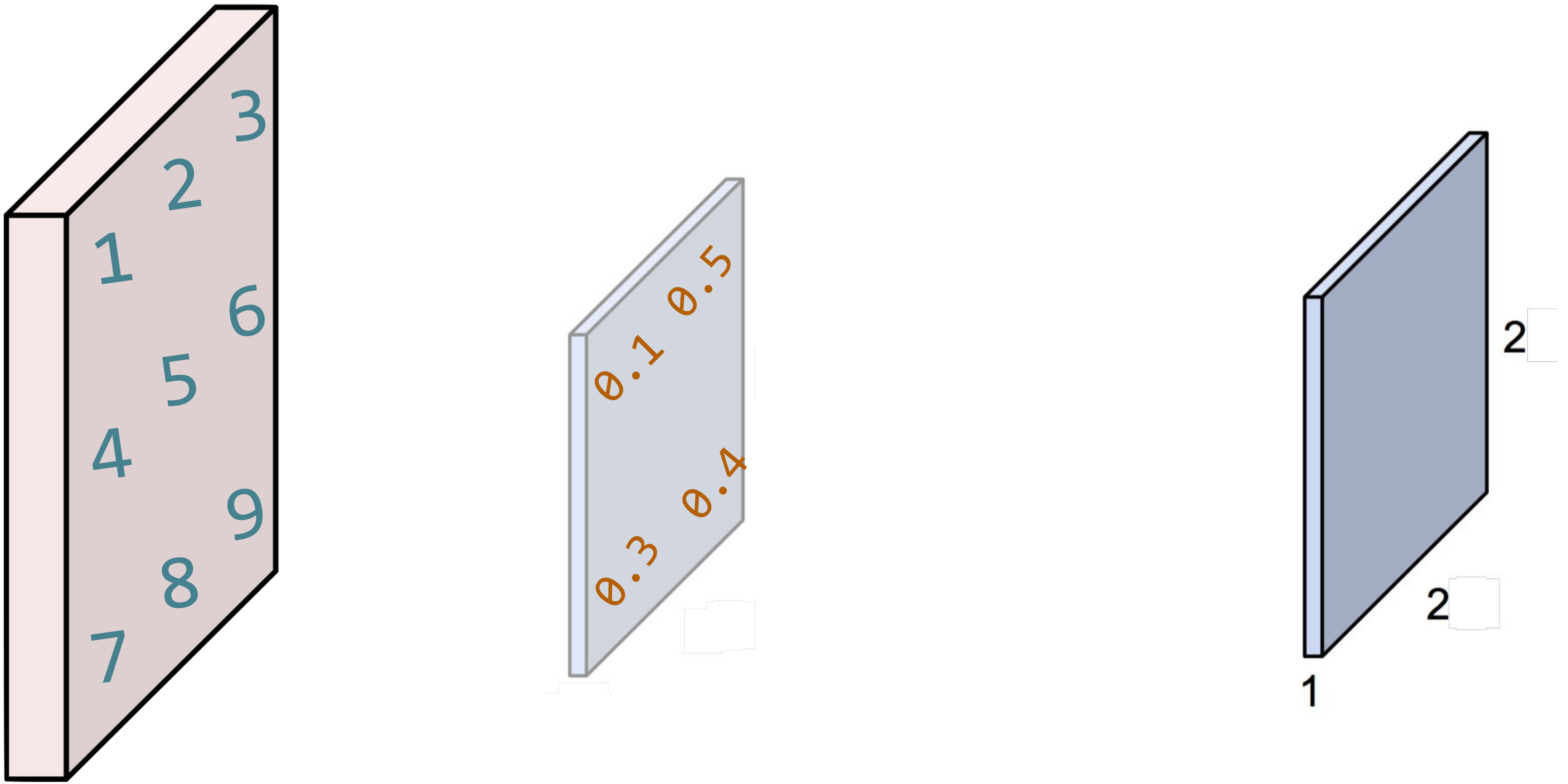
Simple convolution layer

Stride: 1x1



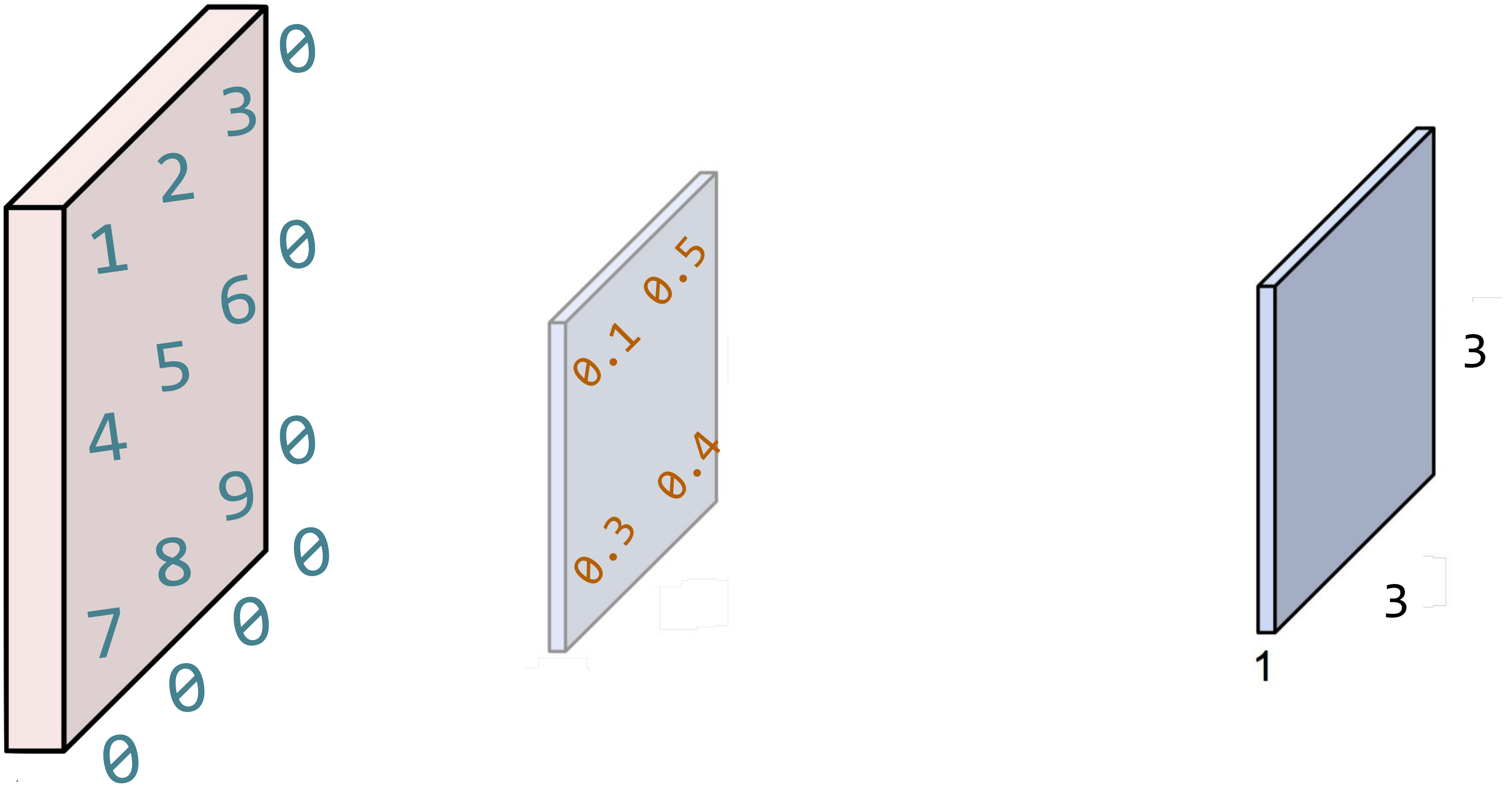
Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, No Padding

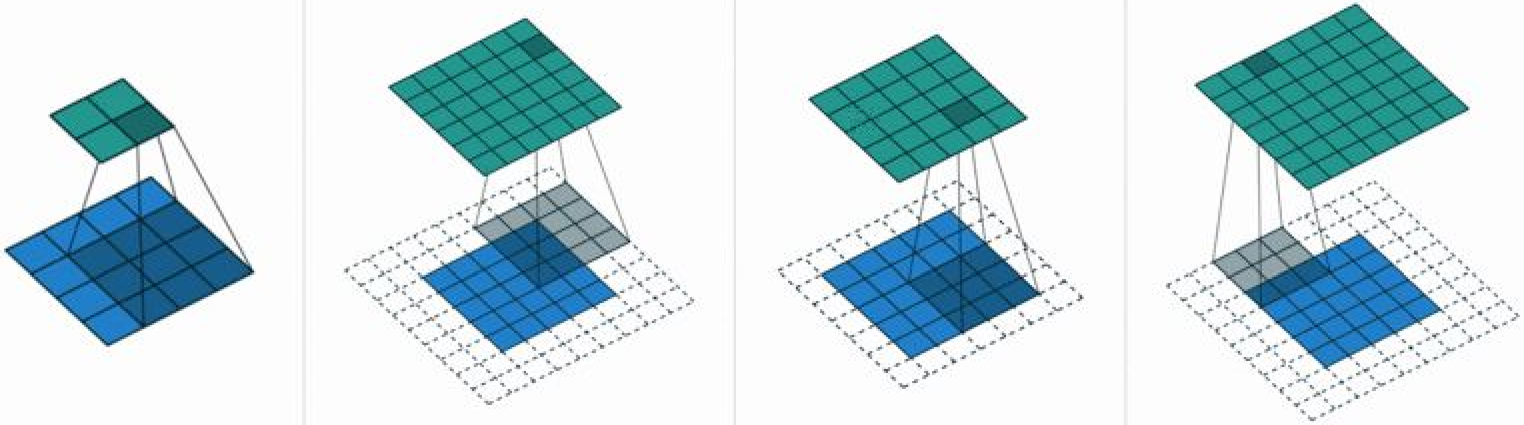


Simple convolution layer

`Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, With padding`



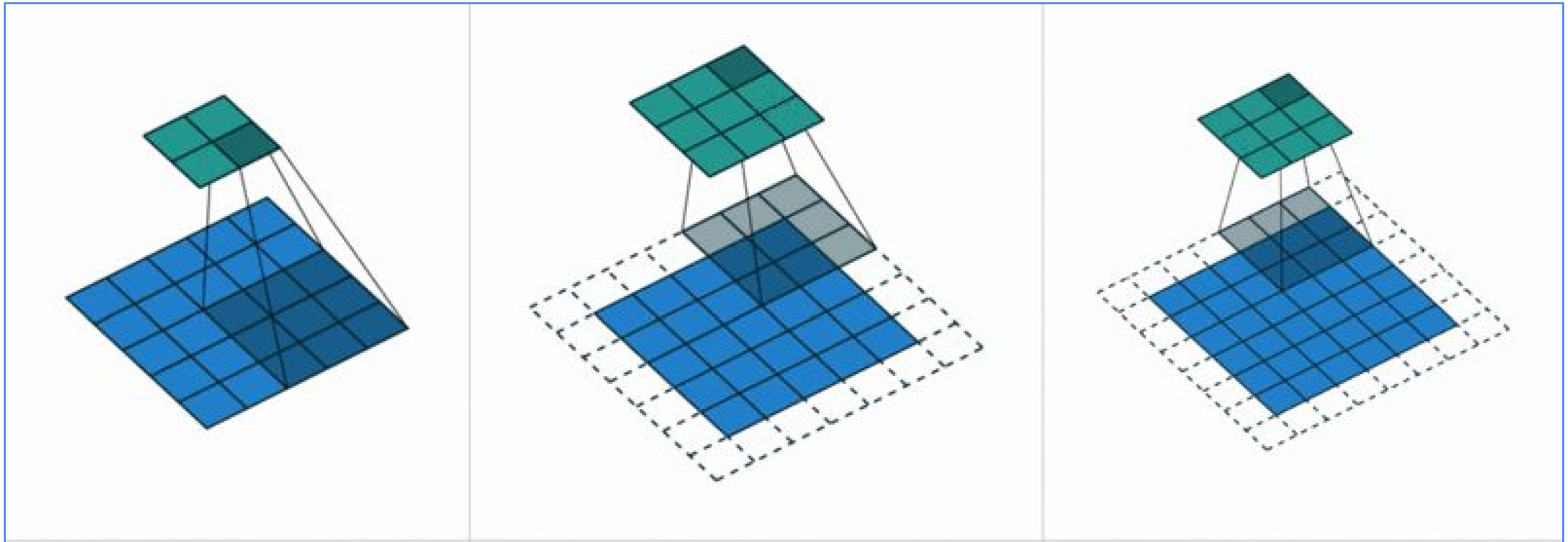
Convolution with padding in Action



No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides

https://github.com/vdumoulin/conv_arithmetic

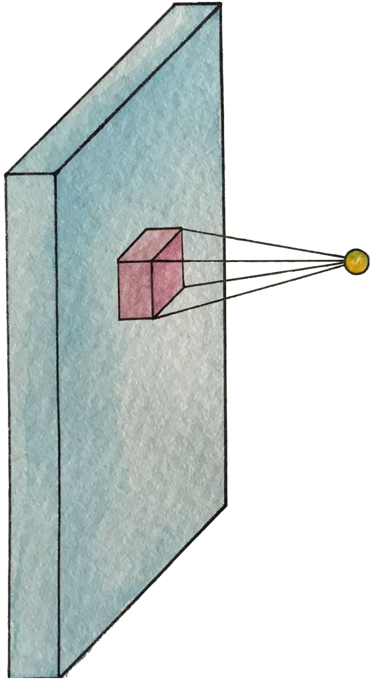
Convolution with stride in Action

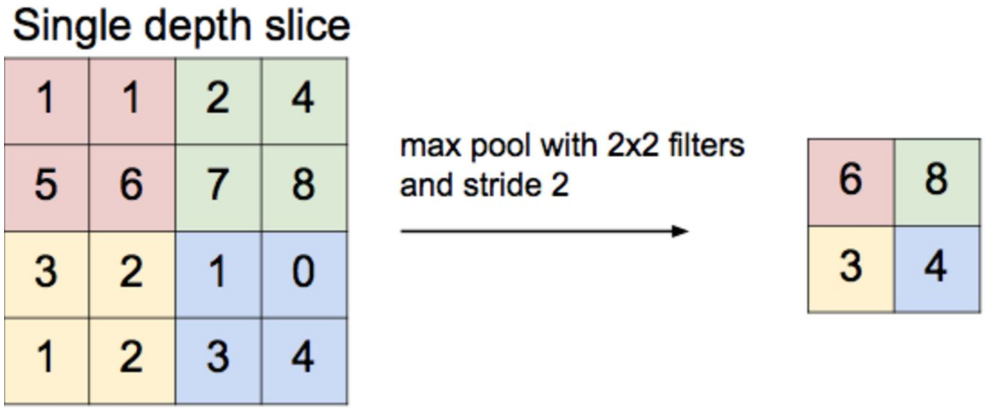


No padding, strides | Padding, strides | Padding, strides (odd)

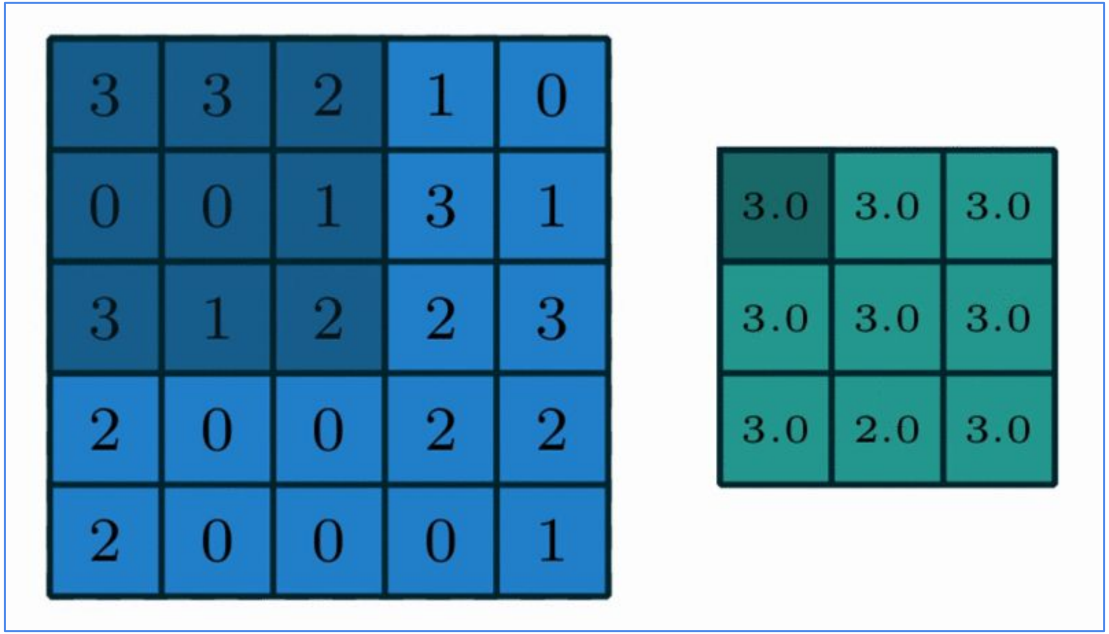
https://github.com/vdumoulin/conv_arithmetic

Max pooling



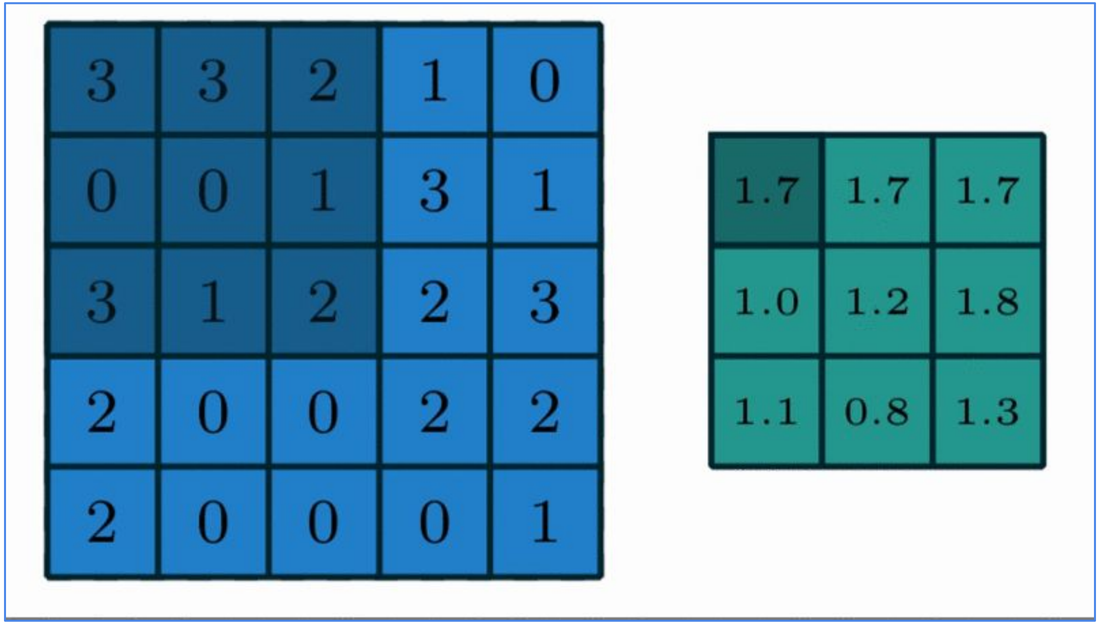

Single depth slice
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Max Pooling in Action



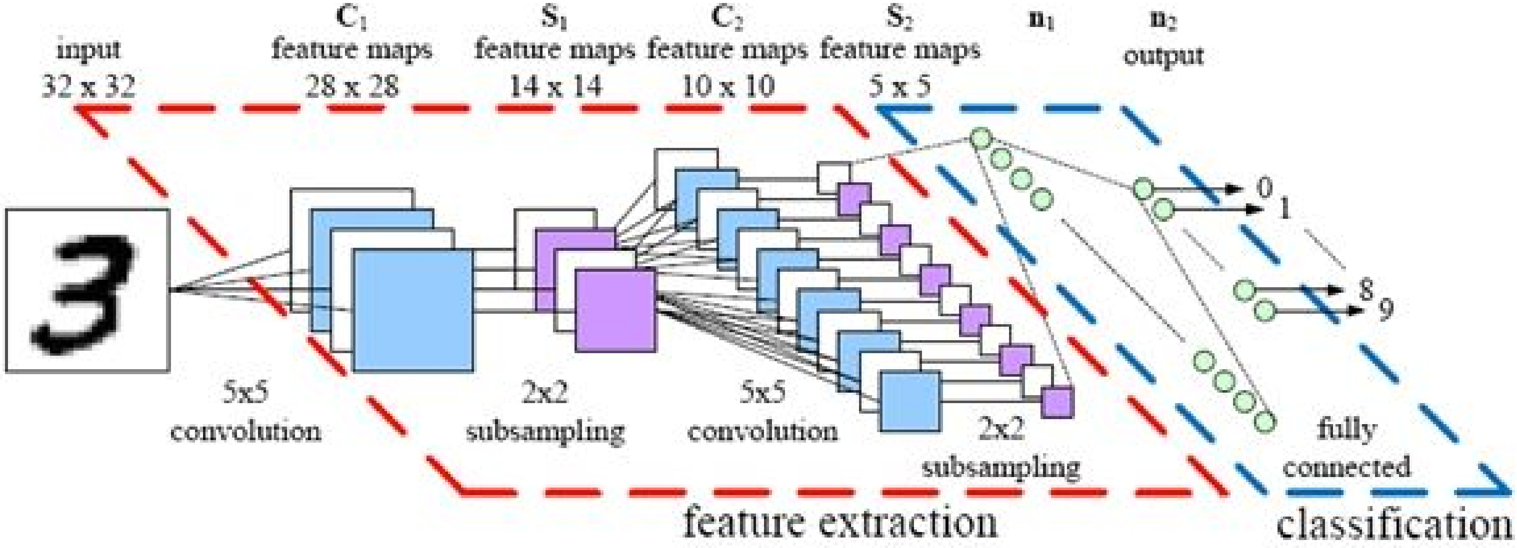
https://github.com/vdumoulin/conv_arithmetic

Avg Pooling in Action



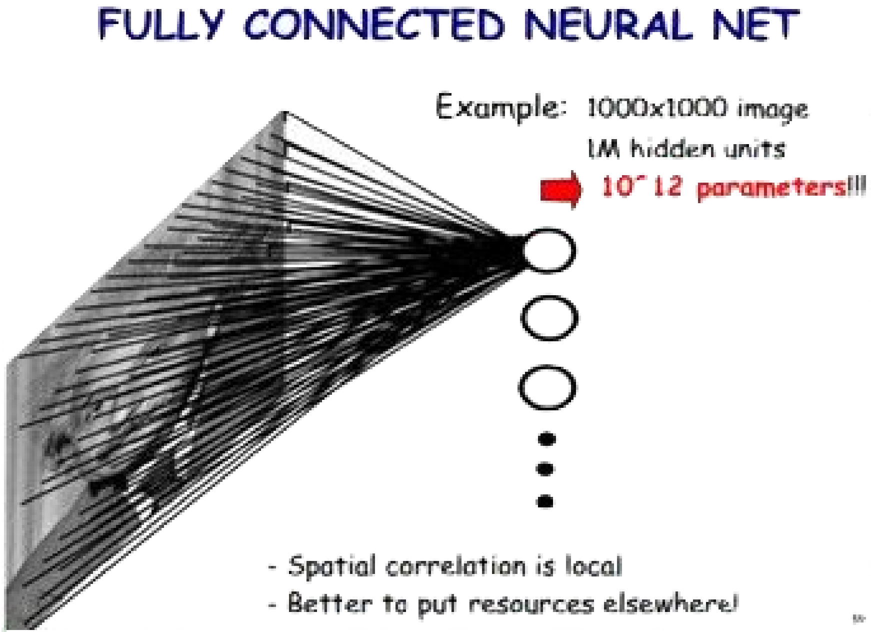
https://github.com/vdumoulin/conv_arithmetic

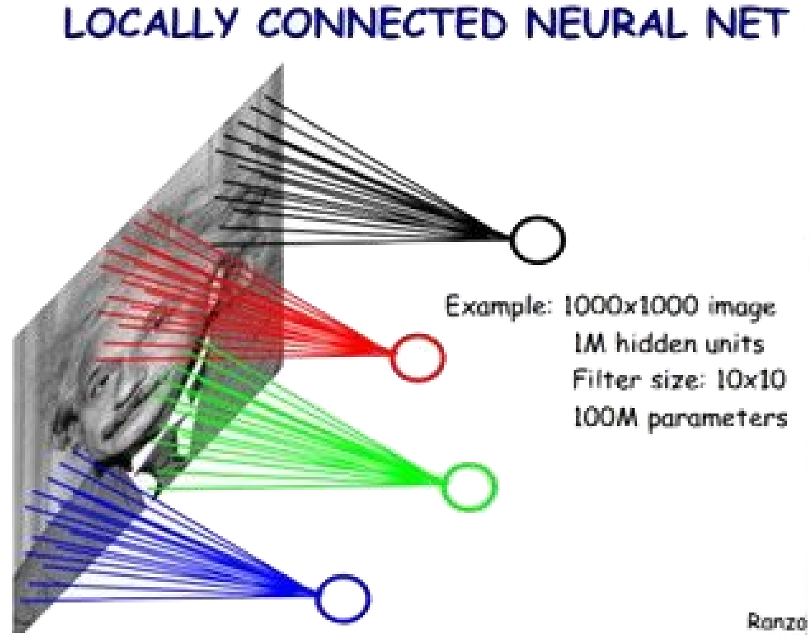
CNN



http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

Locally Connected Features

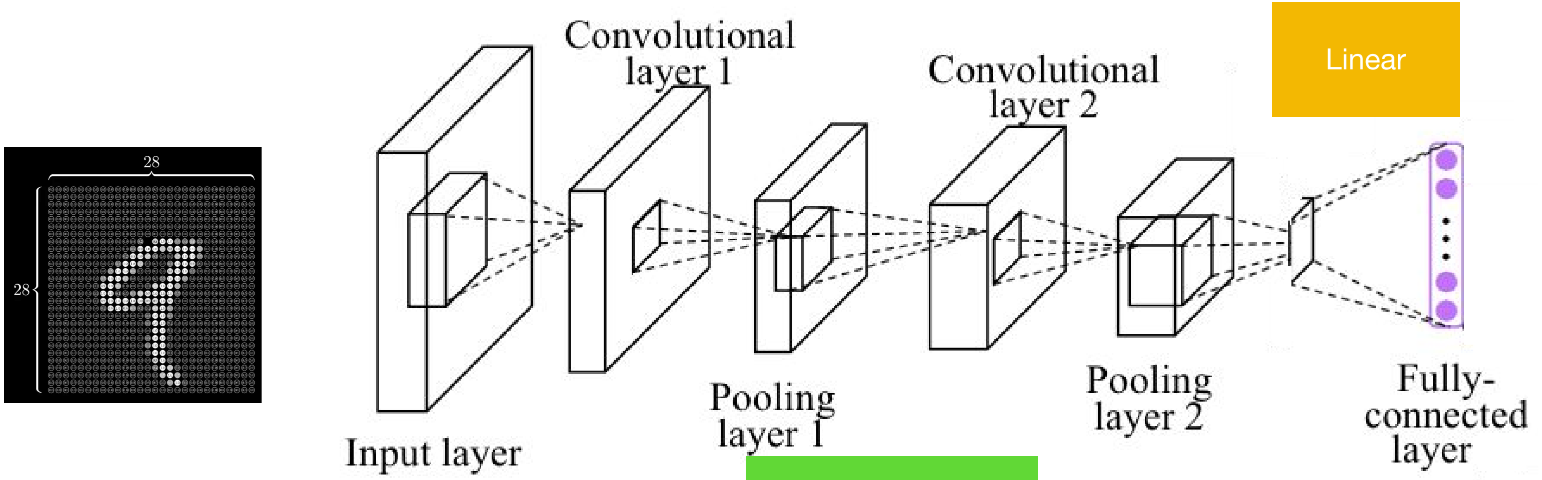




https://ireneli.eu/2016/02/03/deep-learning-05-talk-about-convolutional-neural-network%EF%BC%88cnn%EF%BC%89/

Simple CNN

```python
self.fc = nn.Linear(320, 10)
```

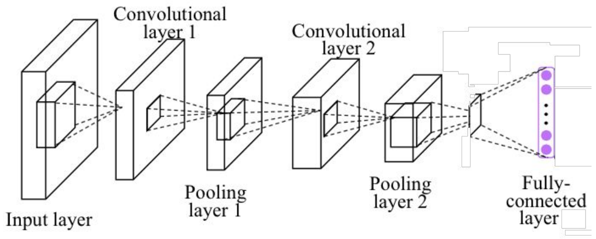


torch.nn.Conv2d (in_channels, out_channels, kernel_size)

nn.MaxPool2d (kernel_size)

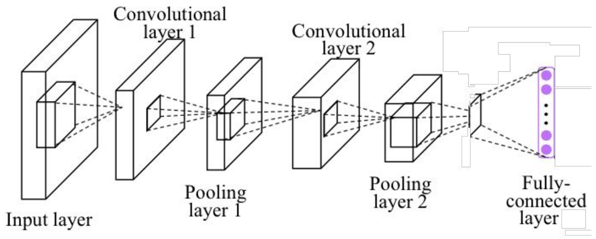
```python
self.mp = nn.MaxPool2d(2)
```

```python
self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
```



Simple CNN

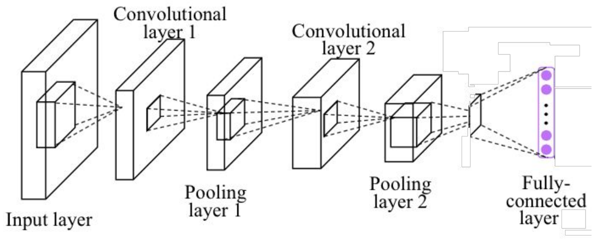
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(100???, 10) # ??? -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```



Simple CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(100???, 10) # ??? -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

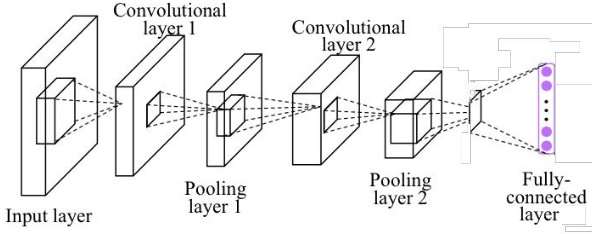
RuntimeError: size mismatch, m1: [64 x **320**], m2: [100 x 10]



Simple CNN

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

RuntimeError: size mismatch, m1: [64 x **320**], m2: [100 x 10]

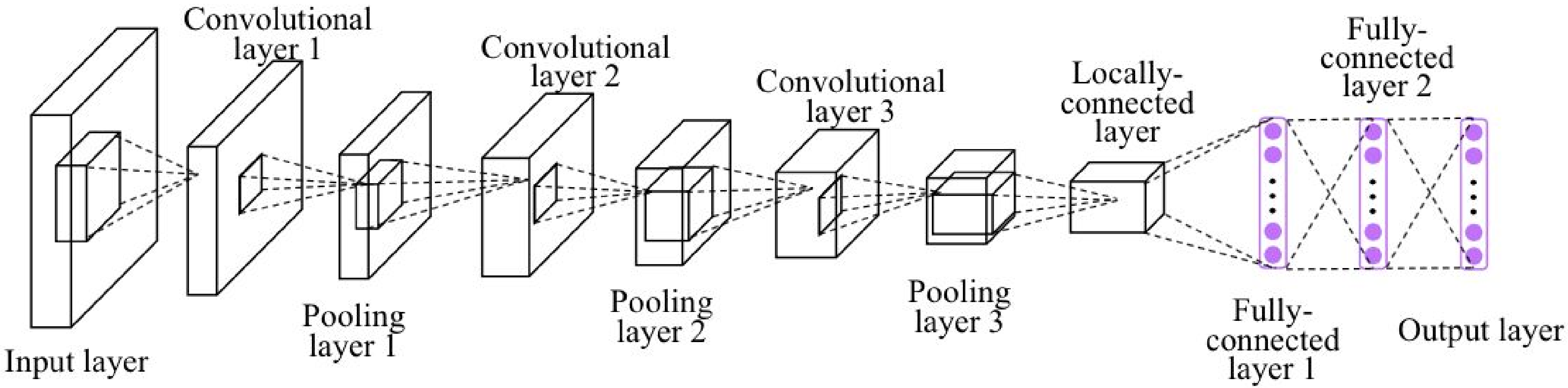


Simple CNN

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

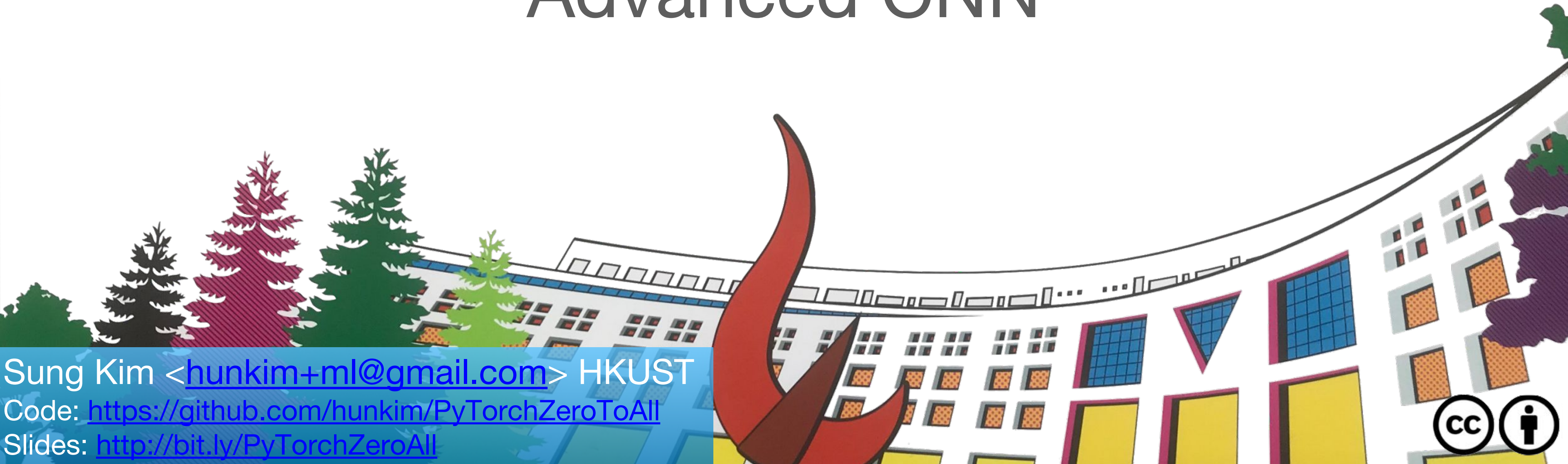
```
Train Epoch: 9 [46080/60000 (77%)]    Loss: 0.108415
Train Epoch: 9 [46720/60000 (78%)]    Loss: 0.140700
Train Epoch: 9 [47360/60000 (79%)]    Loss: 0.090830
Train Epoch: 9 [48000/60000 (80%)]    Loss: 0.031640
Train Epoch: 9 [48640/60000 (81%)]    Loss: 0.014934
Train Epoch: 9 [49280/60000 (82%)]    Loss: 0.090210
Train Epoch: 9 [49920/60000 (83%)]    Loss: 0.074975
Train Epoch: 9 [50560/60000 (84%)]    Loss: 0.058671
Train Epoch: 9 [51200/60000 (85%)]    Loss: 0.023464
Train Epoch: 9 [51840/60000 (86%)]    Loss: 0.018025
Train Epoch: 9 [52480/60000 (87%)]    Loss: 0.098865
Train Epoch: 9 [53120/60000 (88%)]    Loss: 0.013985
Train Epoch: 9 [53760/60000 (90%)]    Loss: 0.070476
Train Epoch: 9 [54400/60000 (91%)]    Loss: 0.065411
Train Epoch: 9 [55040/60000 (92%)]    Loss: 0.028783
Train Epoch: 9 [55680/60000 (93%)]    Loss: 0.008333
Train Epoch: 9 [56320/60000 (94%)]    Loss: 0.020412
Train Epoch: 9 [56960/60000 (95%)]    Loss: 0.036749
Train Epoch: 9 [57600/60000 (96%)]    Loss: 0.163087
Train Epoch: 9 [58240/60000 (97%)]    Loss: 0.117539
Train Epoch: 9 [58880/60000 (98%)]    Loss: 0.032256
Train Epoch: 9 [59520/60000 (99%)]    Loss: 0.026360

Test set: Average loss: 0.0483, Accuracy: 9846/10000 (98%)
```

Exercise 10-1: Implement CNN more layers



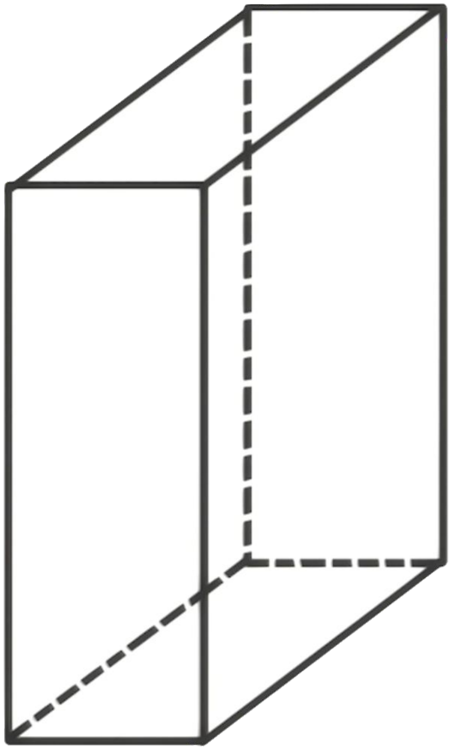
ML/DL for Everyone with PYTORCH

Lecture 10-1:
Advanced CNN



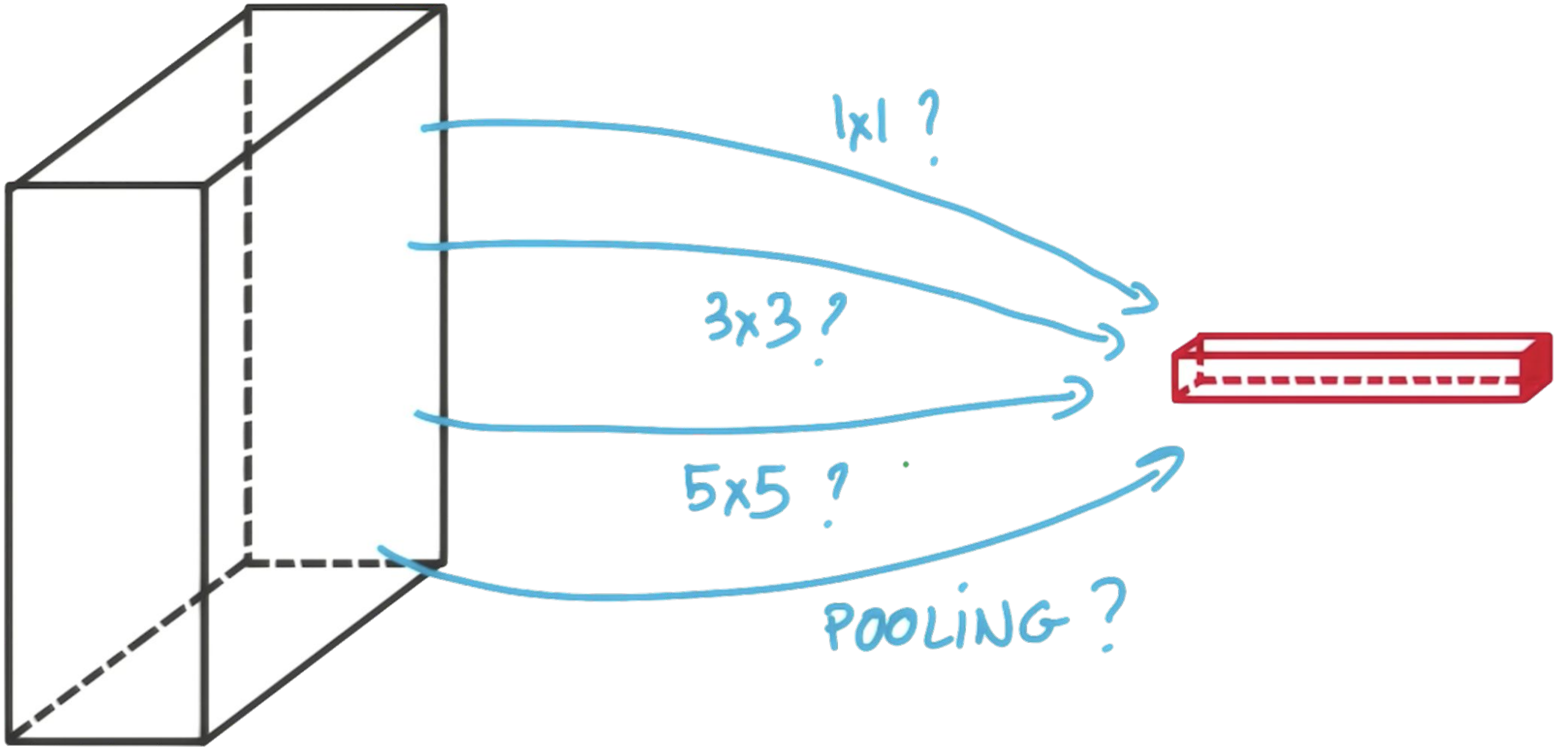
Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

INCEPTION MODULES

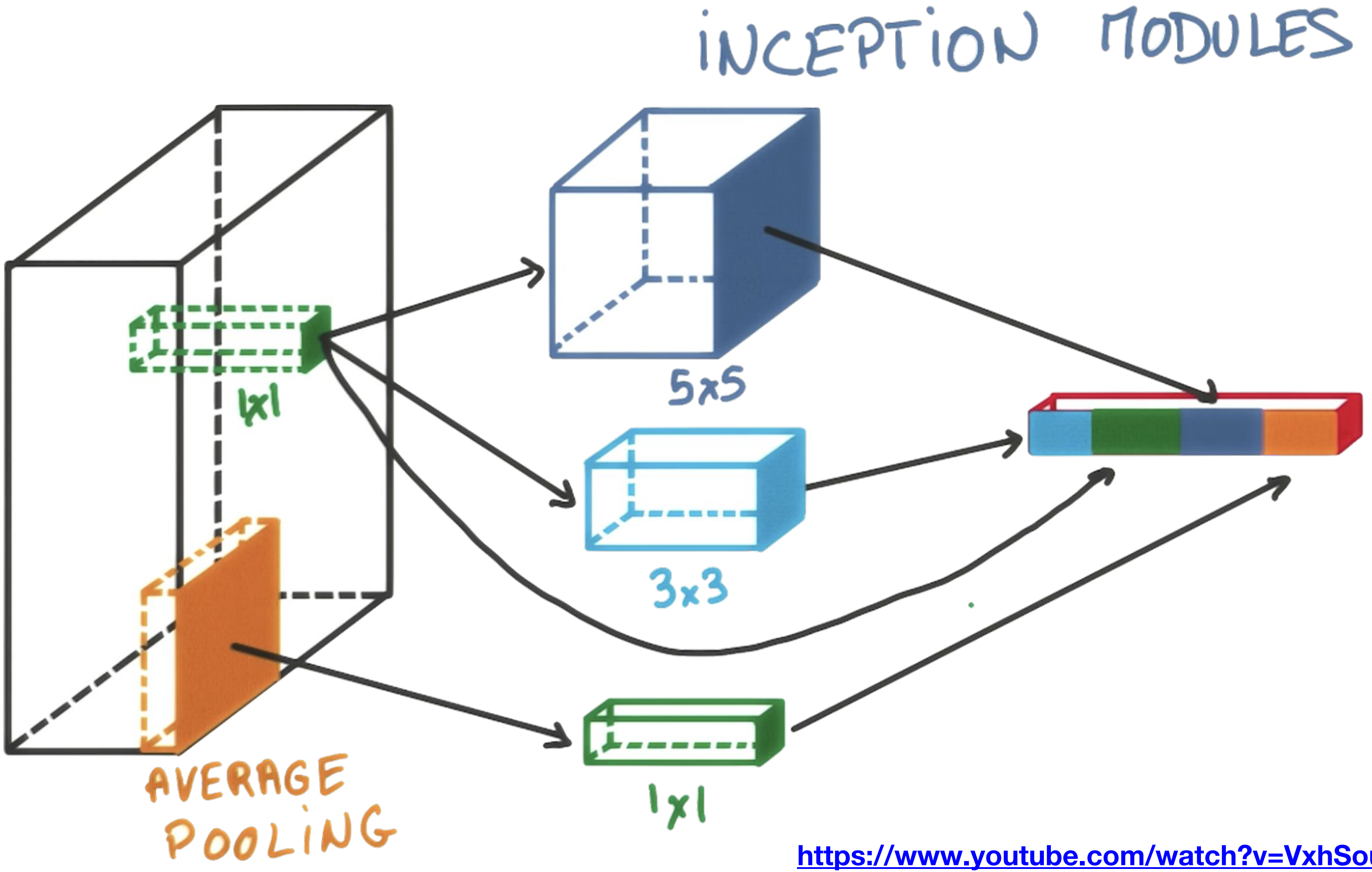


https://www.youtube.com/watch?v=VxhSouuSZDY

INCEPTION MODULES



https://www.youtube.com/watch?v=VxhSouuSZDY



https://www.youtube.com/watch?v=VxhSouuSZDY

Why 1x1 convolution

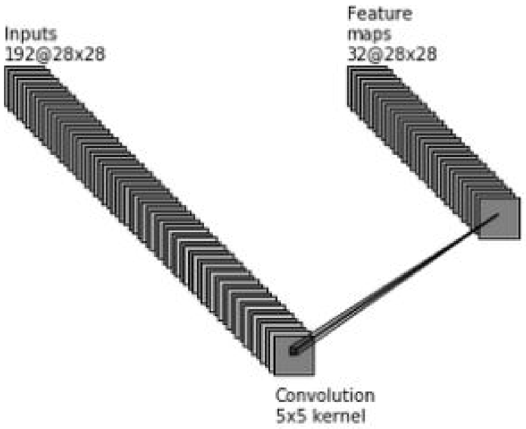


Figure 6. 5×5 convolutions inside the Inception module using the naive model

Without 1x1 convolution:

$$5^2 * 28^2 * 192 * 32 = 120,422,400 \text{ operations}$$

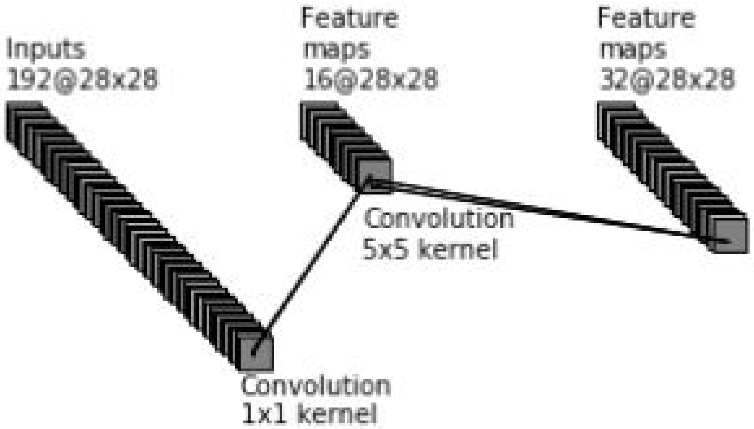


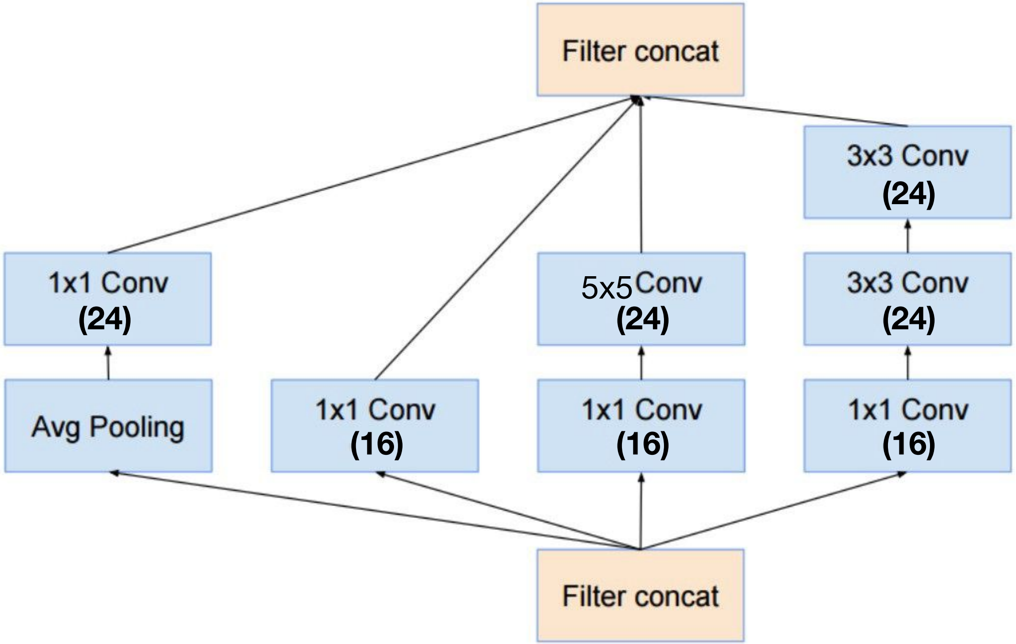
Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

With 1x1 convolution:

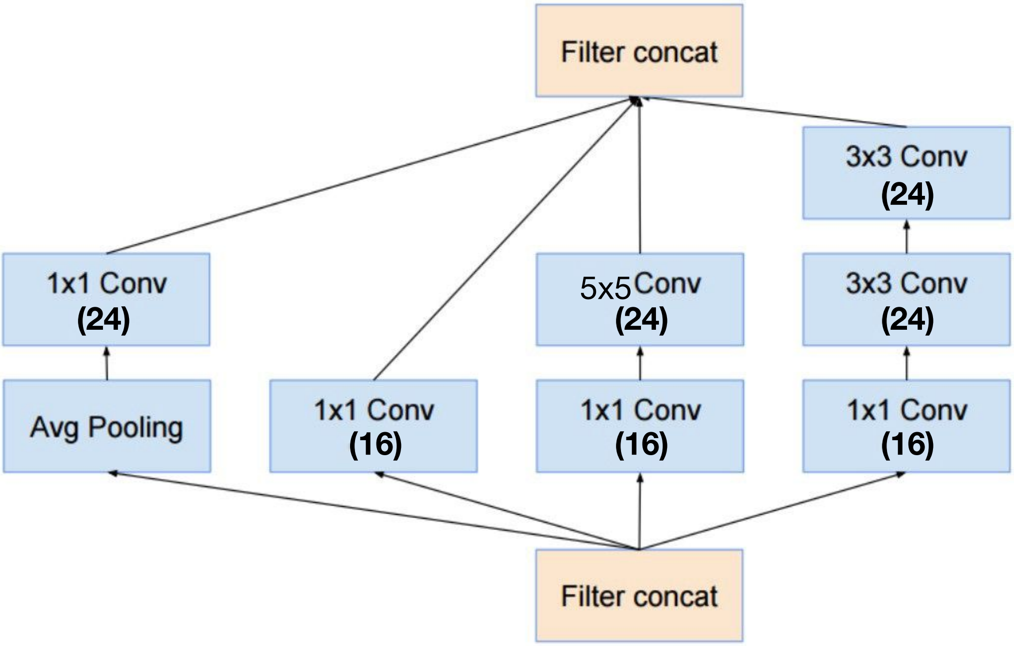
$$1^2 * 28^2 * 192 * 16$$
$$+ 5^2 * 28^2 * 16 * 32$$
$$= 12,443,648 \text{ operations}$$

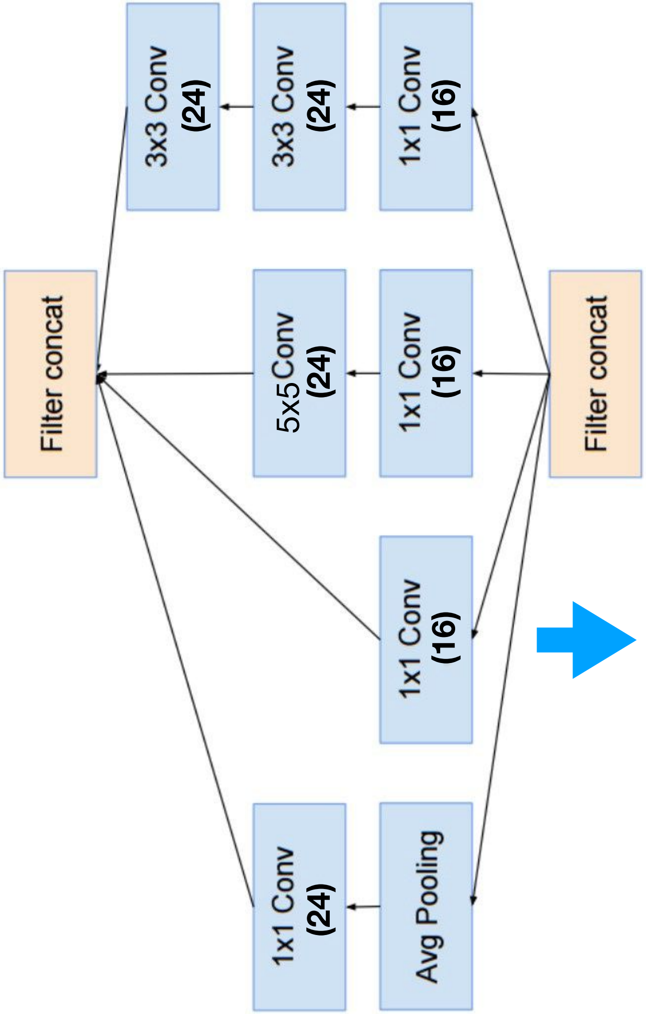
https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

Inception Module

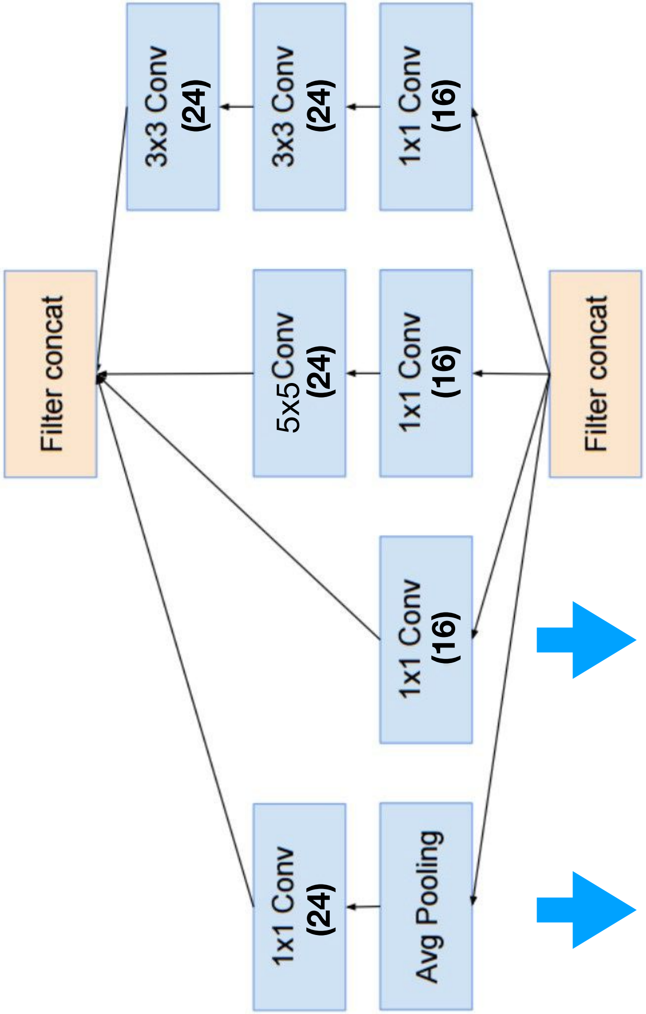


https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/


Filter concat
3x3 Conv
(24)
1x1 Conv
(24)
5x5Conv
(24)
3x3 Conv
(24)
Avg Pooling
1x1 Conv
(16)
1x1 Conv
(16)
1x1 Conv
(16)
Filter concat

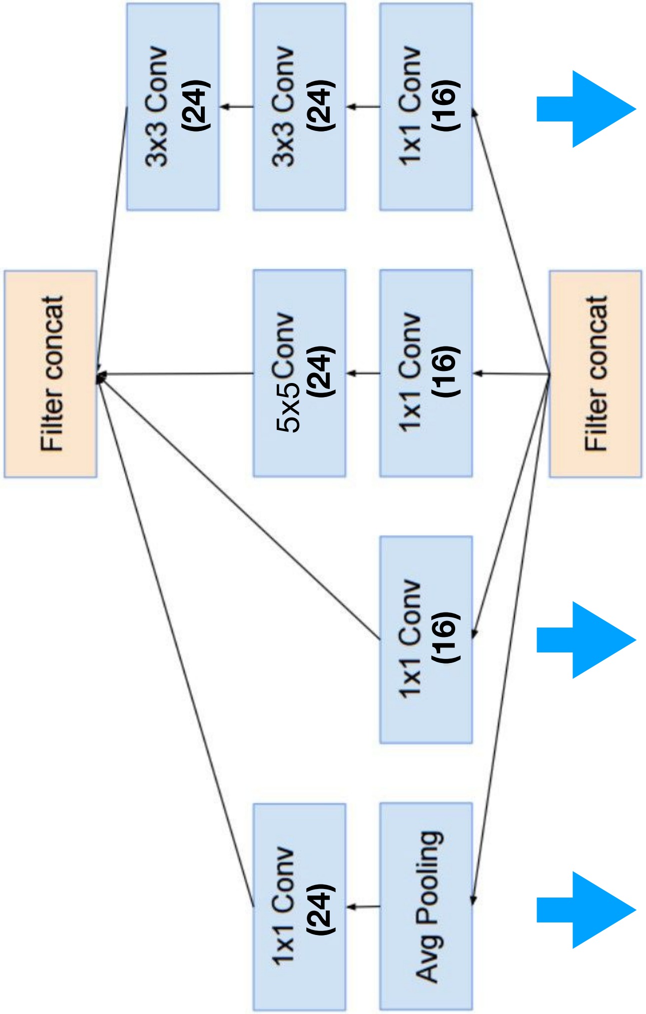



```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```




```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```

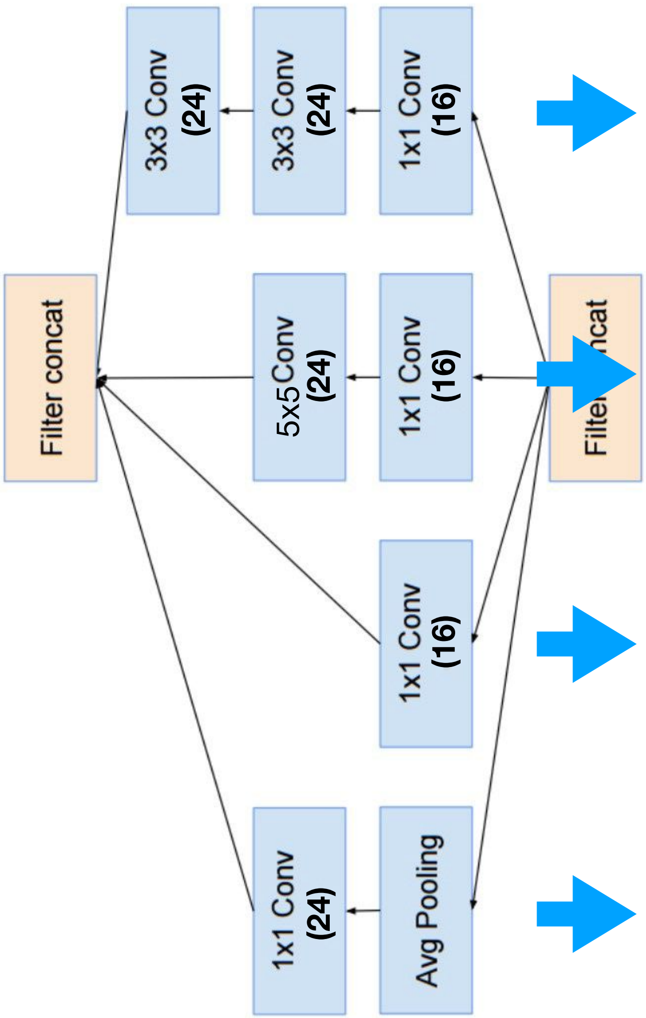
```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
branch_pool = self.branch_pool(branch_pool)
```




```
self.branch3x3dbl_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch3x3dbl_2 = nn.Conv2d(16, 24, kernel_size=3, padding=1)
self.branch3x3dbl_3 = nn.Conv2d(24, 24, kernel_size=3, padding=1)

branch3x3dbl = self.branch3x3dbl_1(x)
branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
branch3x3dbl = self.branch3x3dbl_3(branch3x3dbl)
```

```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```

```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
branch_pool = self.branch_pool(branch_pool)
```

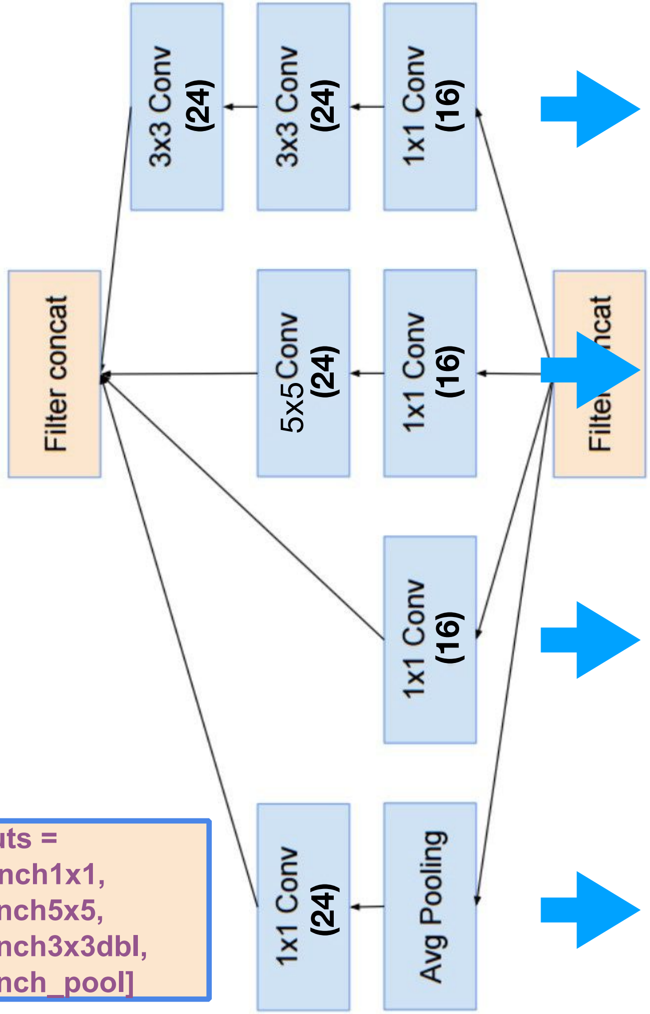



```
self.branch3x3dbl_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch3x3dbl_2 = nn.Conv2d(16, 24, kernel_size=3, padding=1)
self.branch3x3dbl_3 = nn.Conv2d(24, 24, kernel_size=3, padding=1)

branch3x3dbl = self.branch3x3dbl_1(x)
branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
branch3x3dbl = self.branch3x3dbl_3(branch3x3dbl)
```

```
self.branch5x5_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch5x5_2 = nn.Conv2d(16, 24, kernel_size=5, padding=2)

branch5x5 = self.branch5x5_1(x)
branch5x5 = self.branch5x5_2(branch5x5)
```

```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```

```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
branch_pool = self.branch_pool(branch_pool)
```




```python
self.branch3x3dbl_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch3x3dbl_2 = nn.Conv2d(16, 24, kernel_size=3, padding=1)
self.branch3x3dbl_3 = nn.Conv2d(24, 24, kernel_size=3, padding=1)

branch3x3dbl = self.branch3x3dbl_1(x)
branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
branch3x3dbl = self.branch3x3dbl_3(branch3x3dbl)
```

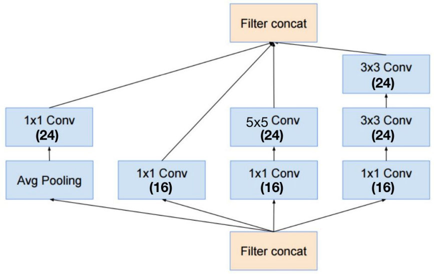
```python
self.branch5x5_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch5x5_2 = nn.Conv2d(16, 24, kernel_size=5, padding=2)

branch5x5 = self.branch5x5_1(x)
branch5x5 = self.branch5x5_2(branch5x5)
```

```python
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```

```python
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
branch_pool = self.branch_pool(branch_pool)
```

```python
outputs =
 [branch1x1,
  branch5x5,
  branch3x3dbl,
  branch_pool]
```

Inception Module

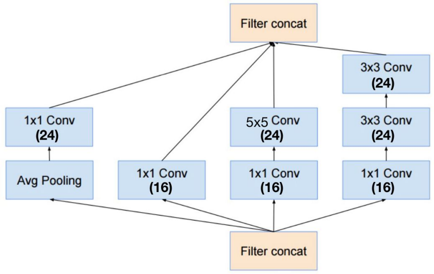



```python
class InceptionA(nn.Module):
    def __init__(self, in_channels):
        super(InceptionA, self).__init__()
        self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

        self.branch5x5_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
        self.branch5x5_2 = nn.Conv2d(16, 24, kernel_size=5, padding=2)

        self.branch3x3dbl_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
        self.branch3x3dbl_2 = nn.Conv2d(16, 24, kernel_size=3, padding=1)
        self.branch3x3dbl_3 = nn.Conv2d(24, 24, kernel_size=3, padding=1)

        self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

    def forward(self, x):
        branch1x1 = self.branch1x1(x)

        branch5x5 = self.branch5x5_1(x)
        branch5x5 = self.branch5x5_2(branch5x5)

        branch3x3dbl = self.branch3x3dbl_1(x)
        branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
        branch3x3dbl = self.branch3x3dbl_3(branch3x3dbl)

        branch_pool = F.avg_pool2d(x, kernel_size=3, stride=1, padding=1)
        branch_pool = self.branch_pool(branch_pool)

        outputs = [branch1x1, branch5x5, branch3x3dbl, branch_pool]
        return torch.cat(outputs, 1)
```

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(88, 20, kernel_size=5)

        self.incept1 = InceptionA(in_channels=10)
        self.incept2 = InceptionA(in_channels=20)

        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(1408, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = self.incept1(x)
        x = F.relu(self.mp(self.conv2(x)))
        x = self.incept2(x)
        x = x.view(in_size, -1)  # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

https://github.com/.../torchvision/models/inception.py

Inception Module

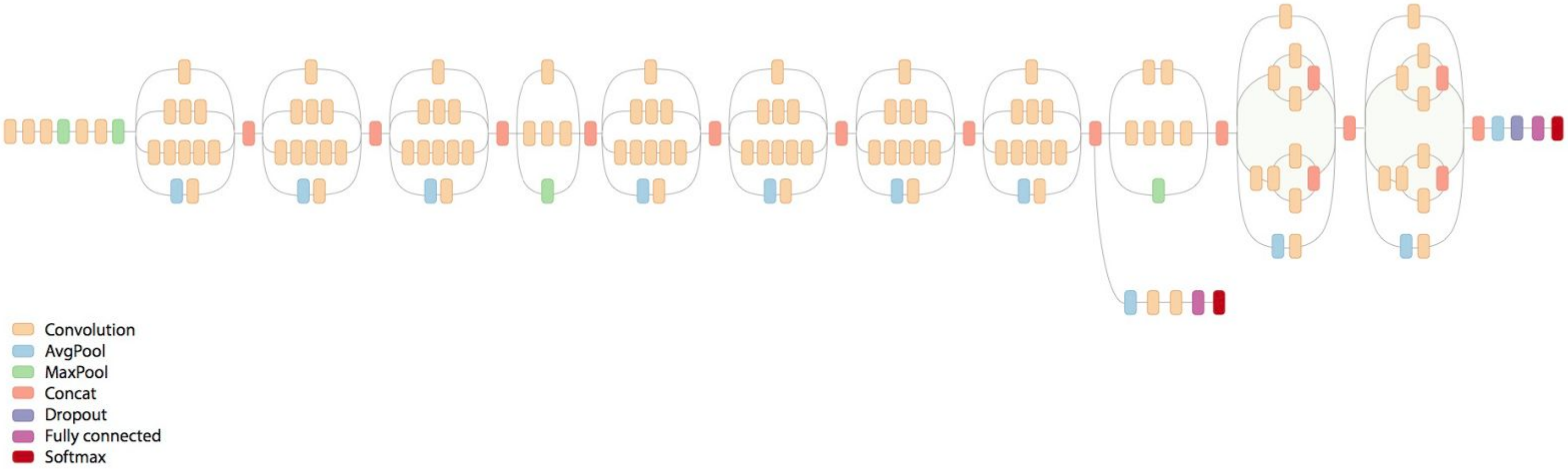


Train Epoch: 9 [44800/60000 (75%)]	Loss: 0.064180
Train Epoch: 9 [45440/60000 (76%)]	Loss: 0.020339
Train Epoch: 9 [46080/60000 (77%)]	Loss: 0.061476
Train Epoch: 9 [46720/60000 (78%)]	Loss: 0.039662
Train Epoch: 9 [47360/60000 (79%)]	Loss: 0.026798
Train Epoch: 9 [48000/60000 (80%)]	Loss: 0.071569
Train Epoch: 9 [48640/60000 (81%)]	Loss: 0.003835
Train Epoch: 9 [49280/60000 (82%)]	Loss: 0.005564
Train Epoch: 9 [49920/60000 (83%)]	Loss: 0.020116
Train Epoch: 9 [50560/60000 (84%)]	Loss: 0.128114
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.016599
Train Epoch: 9 [51840/60000 (86%)]	Loss: 0.006995
Train Epoch: 9 [52480/60000 (87%)]	Loss: 0.111267
Train Epoch: 9 [53120/60000 (88%)]	Loss: 0.052126
Train Epoch: 9 [53760/60000 (90%)]	Loss: 0.034962
Train Epoch: 9 [54400/60000 (91%)]	Loss: 0.029465
Train Epoch: 9 [55040/60000 (92%)]	Loss: 0.031482
Train Epoch: 9 [55680/60000 (93%)]	Loss: 0.015132
Train Epoch: 9 [56320/60000 (94%)]	Loss: 0.010435
Train Epoch: 9 [56960/60000 (95%)]	Loss: 0.014344
Train Epoch: 9 [57600/60000 (96%)]	Loss: 0.014952
Train Epoch: 9 [58240/60000 (97%)]	Loss: 0.153132
Train Epoch: 9 [58880/60000 (98%)]	Loss: 0.112024
Train Epoch: 9 [59520/60000 (99%)]	Loss: 0.009406

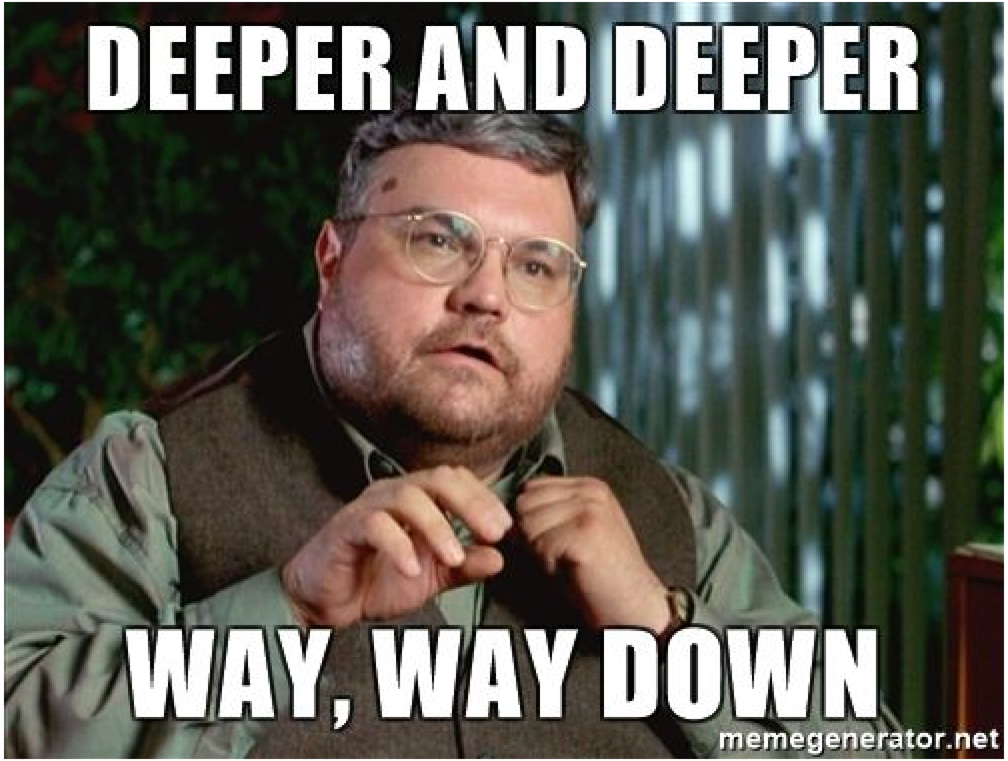
Test set: Average loss: 0.0470, Accuracy: 9866/10000 (99%)

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(88, 20, kernel_size=5)

        self.incept1 = InceptionA(in_channels=10)
        self.incept2 = InceptionA(in_channels=20)

        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(1408, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = self.incept1(x)
        x = F.relu(self.mp(self.conv2(x)))
        x = self.incept2(x)
        x = x.view(in_size, -1)  # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

Exercise 10-2: Implement full inception v3/v4



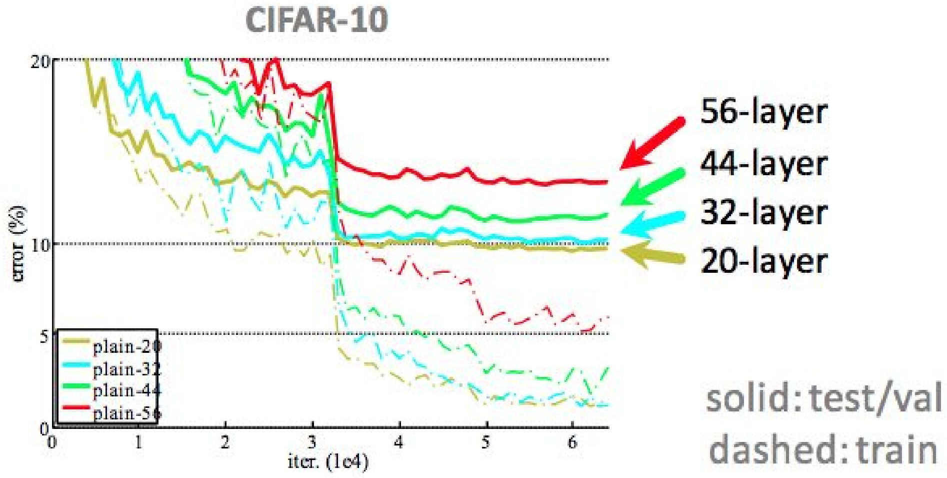
https://research.googleblog.com/2016/08/improving-inception-and-image.html

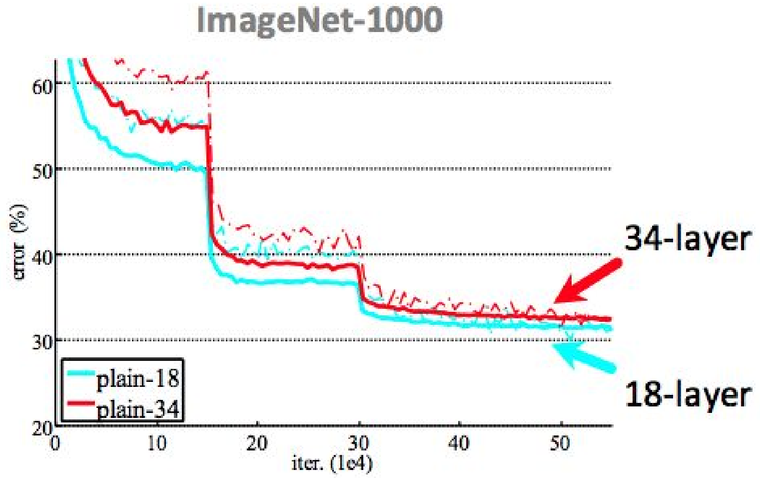


https://www.youtube.com/watch?v=C6tLw-rPQ2o

Can we just keep stacking layers?

Not
exactly…





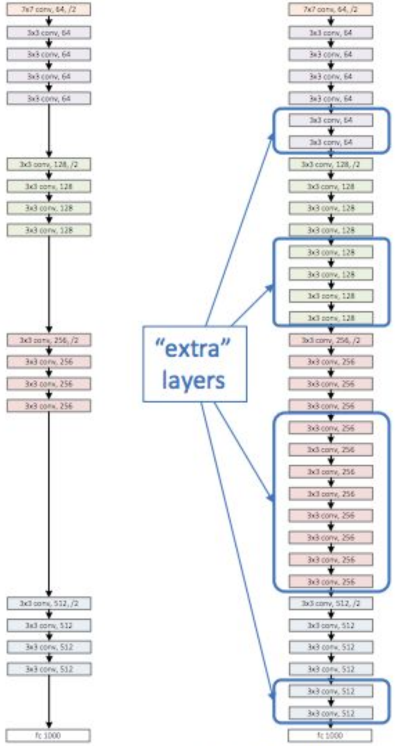
http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Problems with stacking layers

- Vanishing Gradients problem
- Back propagation kind of gives up…
- Degradation problem - with increased network depth accuracy gets saturated and then rapidly degrades

Making shallow model deep…

a shallower model (18 layers)



a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

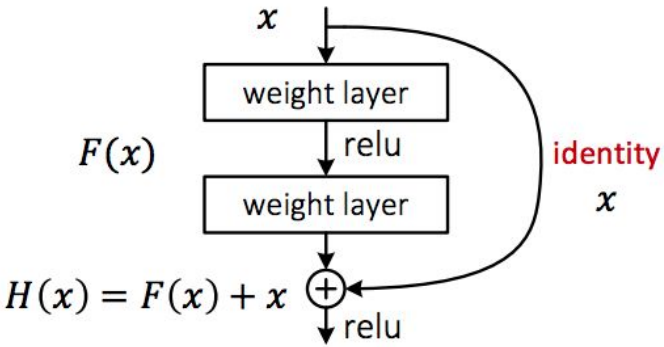
What is Residual Learning?

Deeper the network the harder it is to find the gradients

Residual learning makes it easier to find gradients

Deep Residual Learning

- Residual net



$H(x)$ is any desired mapping,

~~hope the 2 weight layers fit $H(x)$~~

hope the 2 weight layers fit $F(x)$

let $H(x) = F(x) + x$

- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

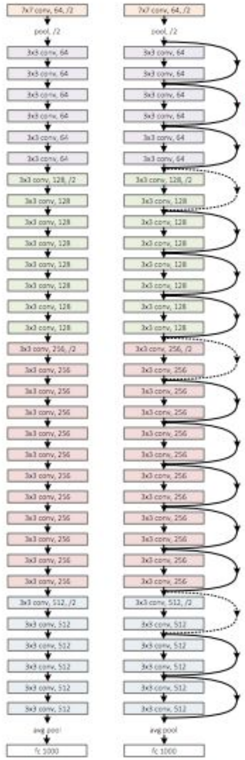
http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Turning a plain net into ResNet

Network “Design”

- Keep it simple
- Our basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2 (~same complexity per layer)
  - Simple design; just deep!
- Other remarks:
  - no hidden fc
  - no dropout

plain net

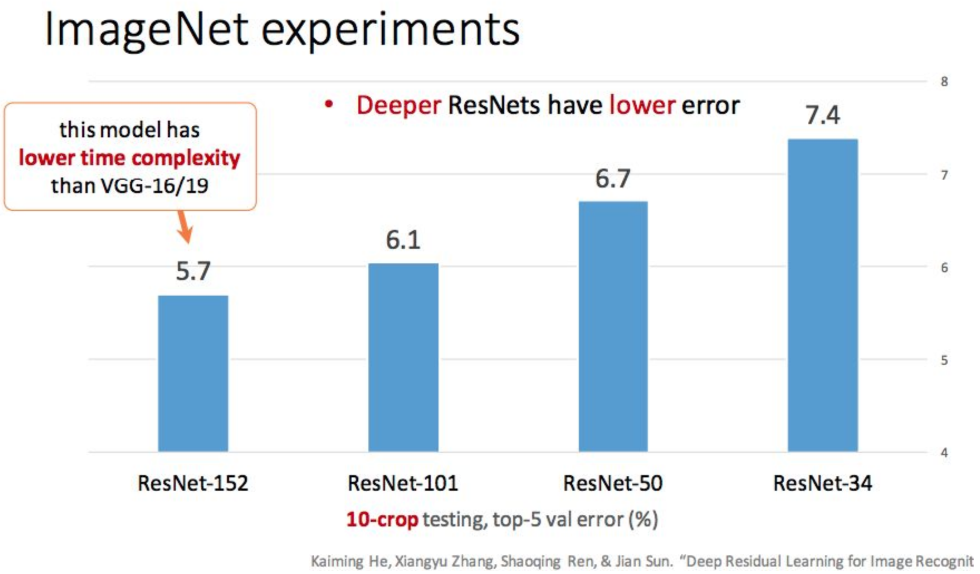


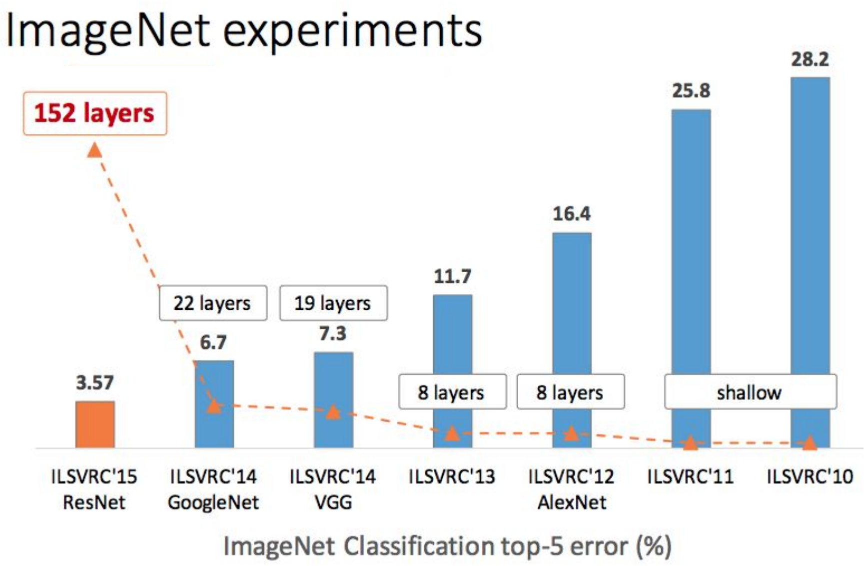
ResNet

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Performance with deeper ResNet

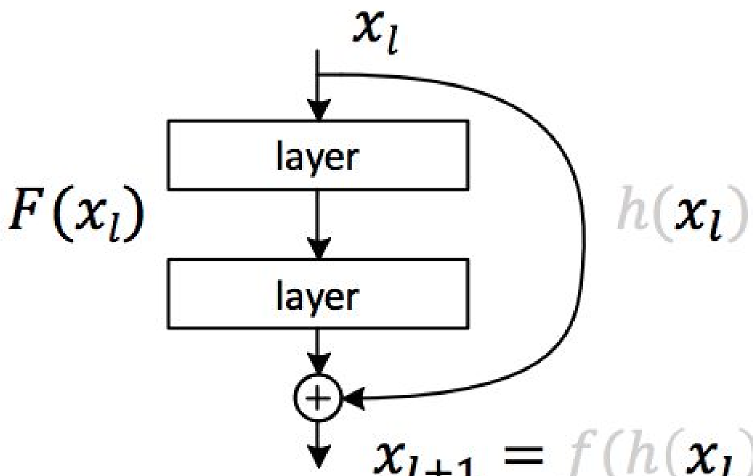




http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

What’s the reasoning?

On identity mappings for **optimization**



$$x_{l+1} = f(h(x_l) + F(x_l))$$

- shortcut mapping: h = identity
- ~~after-add mapping: f = ReLU~~
- What if f = identity?

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Identity Mappings in Deep Residual Networks”. arXiv 2016.

Improving Forward/back propagation

Very smooth forward propagation

$$x_{l+1} = x_l + F(x_l)$$

$$x_{l+2} = x_{l+1} + F(x_{l+1})$$

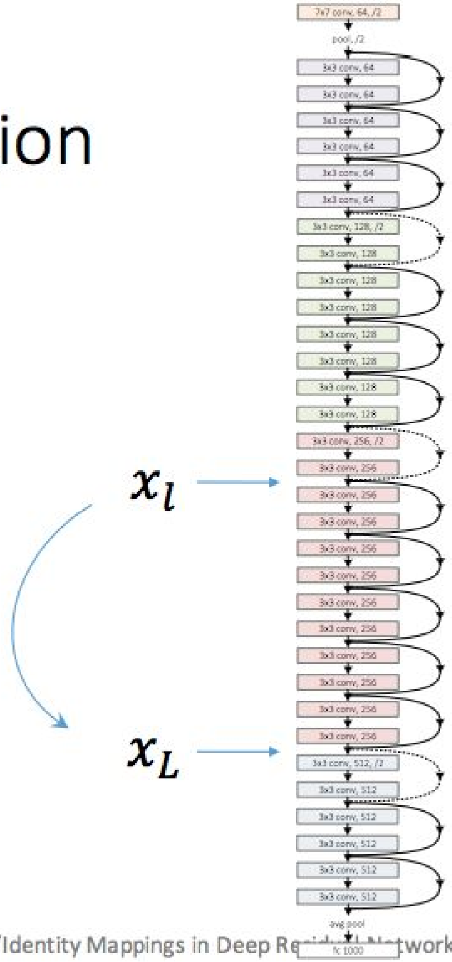
$$x_{l+2} = x_l + F(x_l) + F(x_{l+1})$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

Very smooth forward propagation

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

- Any x_l is directly forward-prop to any x_L, plus residual.
- Any x_L is an additive outcome.
  - in contrast to multiplicative: $x_L = \prod_{i=l}^{L-1} W_i x_l$



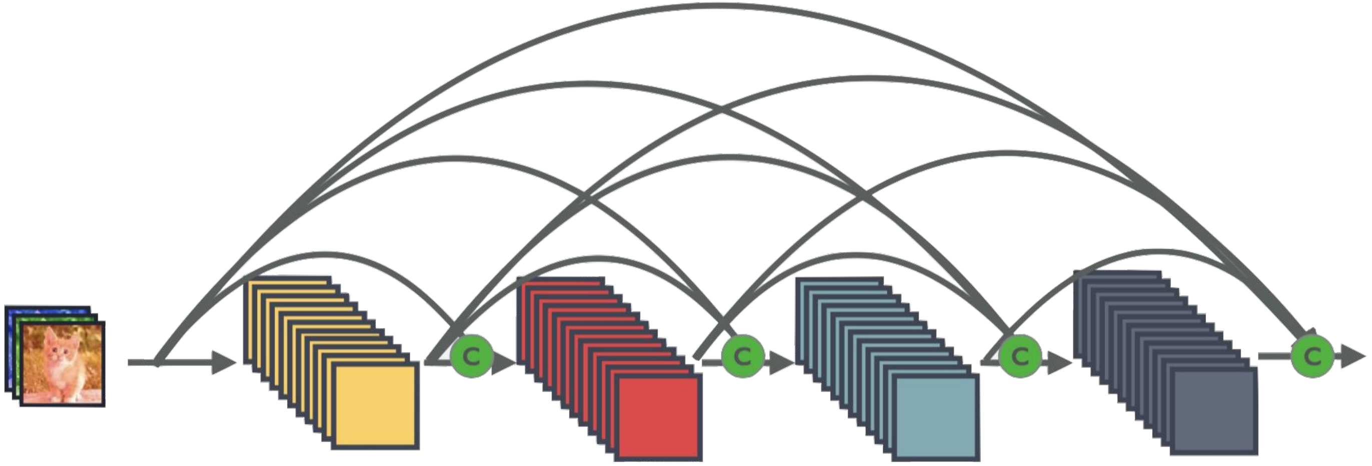
Becomes multiplicative in the case that h(x) is not identity

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Identity Mappings in Deep Residual Networks". arXiv 2016.

Exercise 10-3: Implement ResNet

https://arxiv.org/abs/1608.06993

Exercise 10-4: Implement DenseNet



https://arxiv.org/abs/1608.06993

WHAT NEXT

Lecture 11: RNN