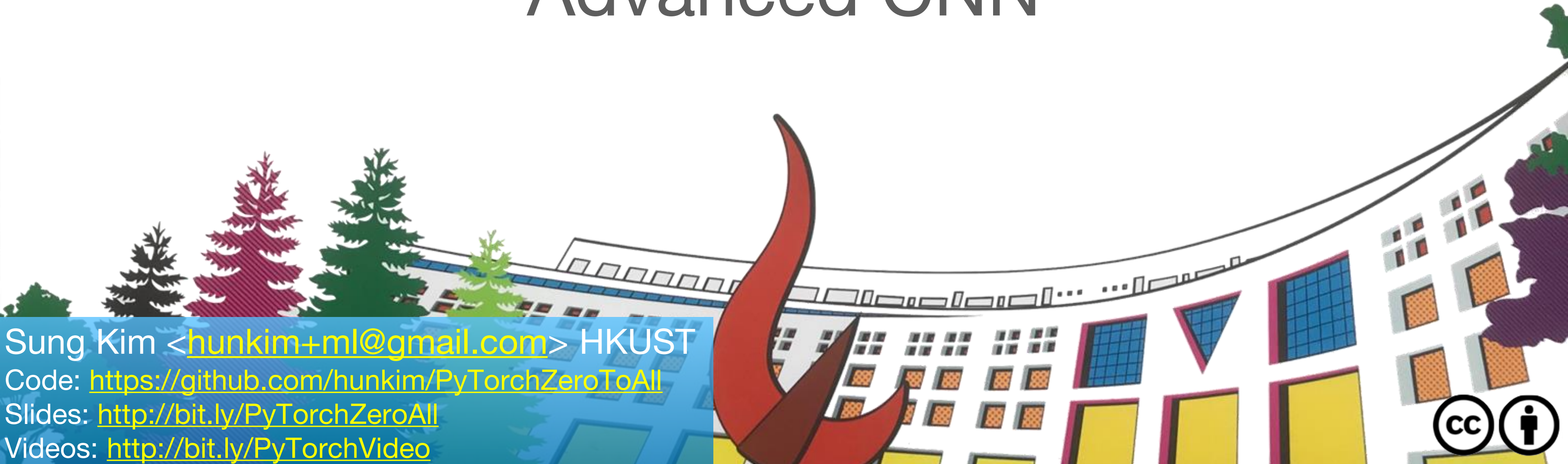
ML/DL for Everyone with PYTORCH

Lecture 11:

Advanced CNN



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

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

Slides: http://bit.ly/PyTorchZeroAll

Videos: http://bit.ly/PyTorchVideo

Call for Comments

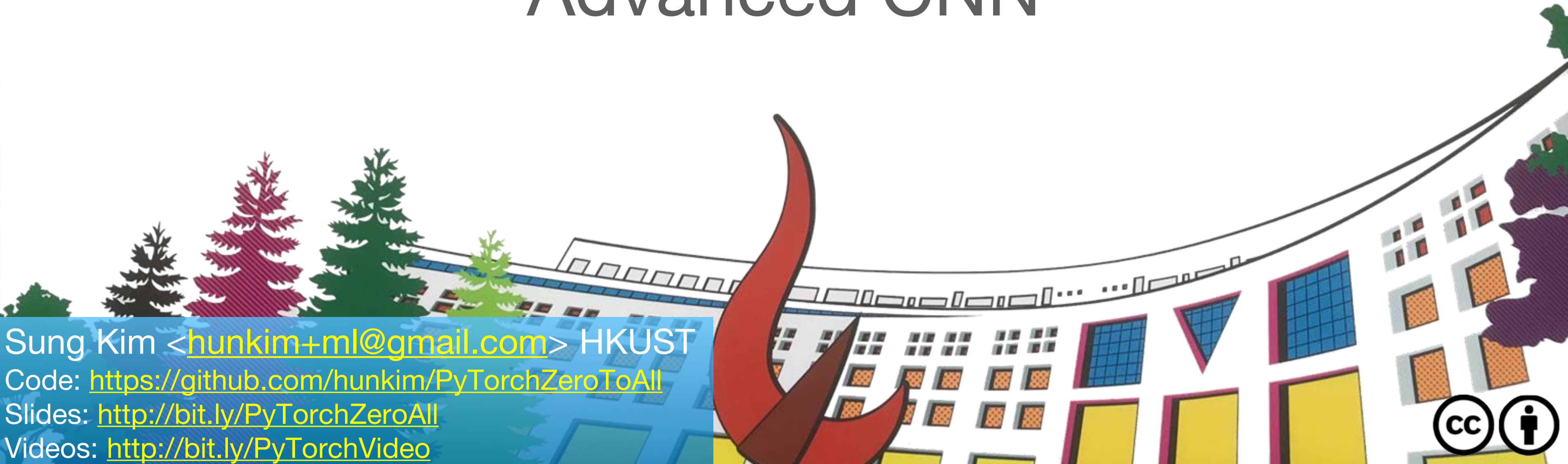
Please feel free to add comments directly on these slides.

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

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

ML/DL for Everyone with PYTORCH

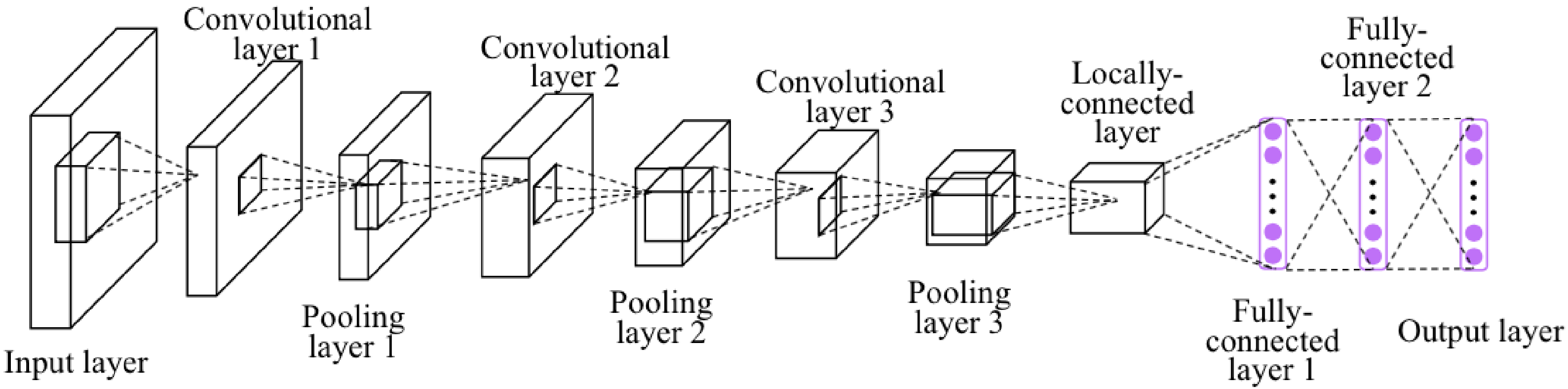
Lecture 11: Advanced CNN



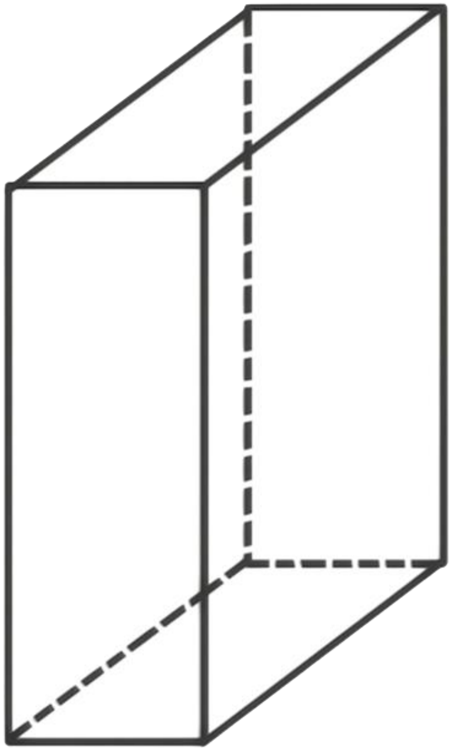
Sung Kim <hunkim+ml@gmail.com> HKUST

Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll
Videos: http://bit.ly/PyTorchVideo

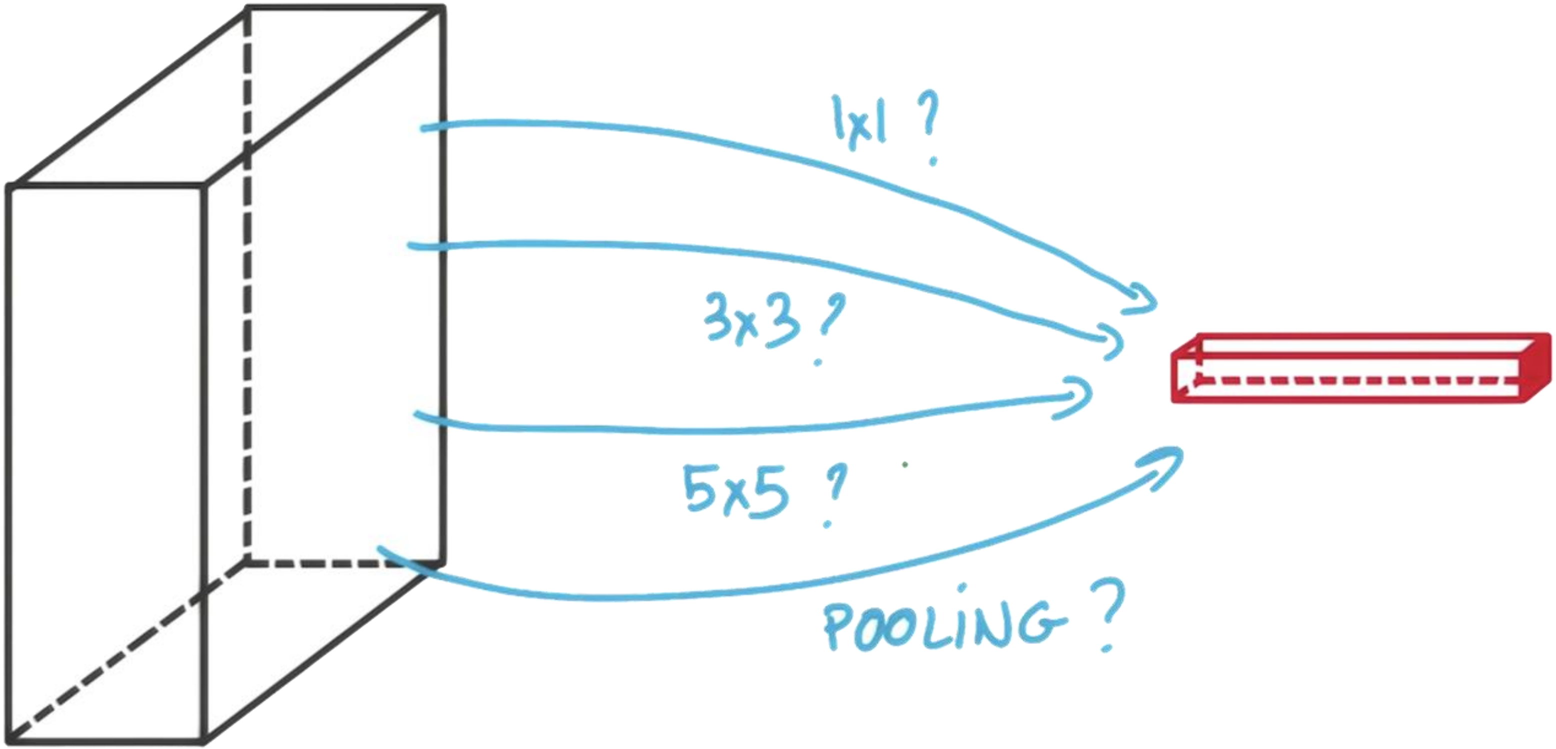
CNN



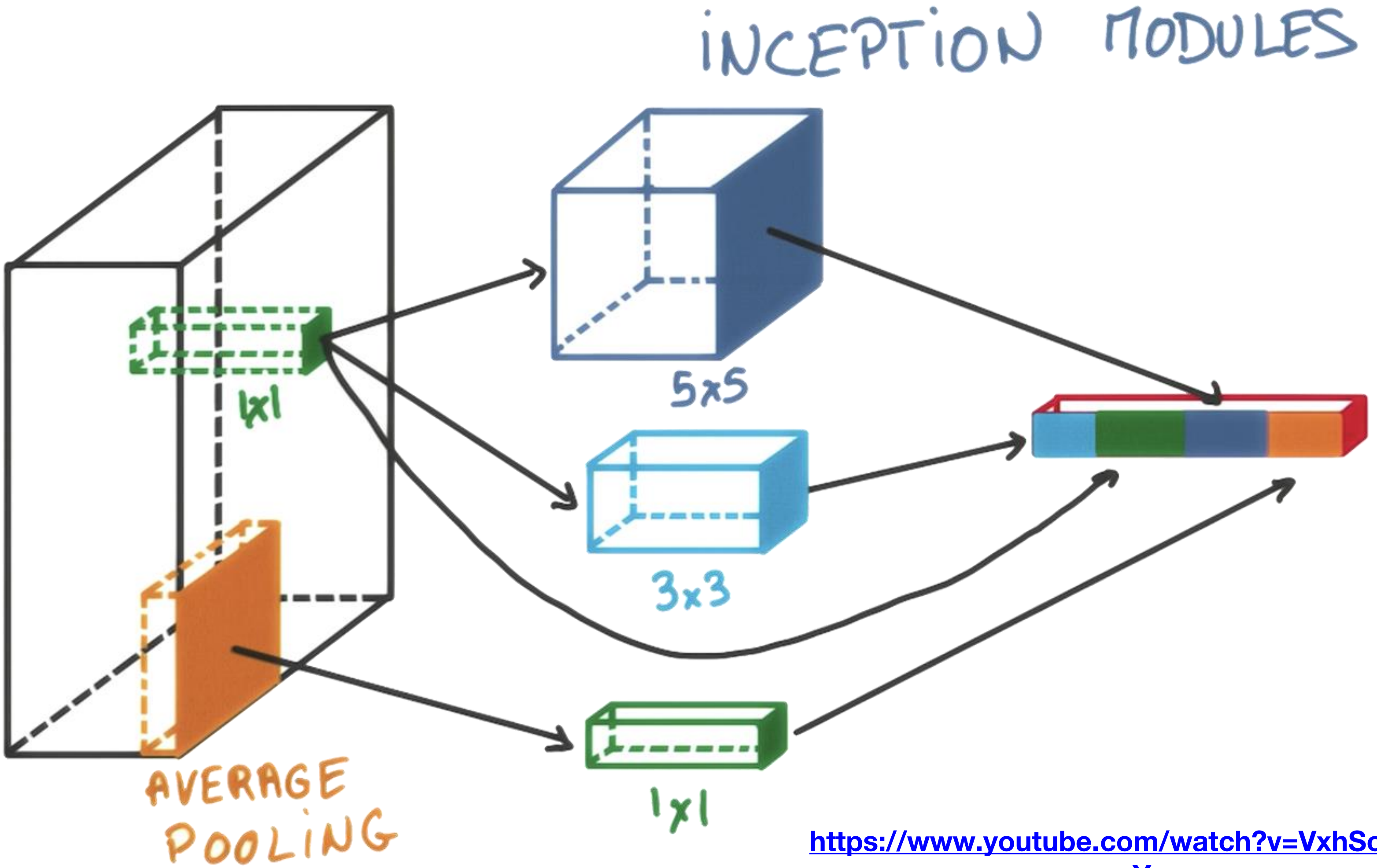
Convolutional layer 1
Convolutional layer 2
Convolutional layer 3
Locally-connected layer
Fully-connected layer 2
Input layer
Pooling layer 1
Pooling layer 2
Pooling layer 3
Fully-connected layer 1
Output layer



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

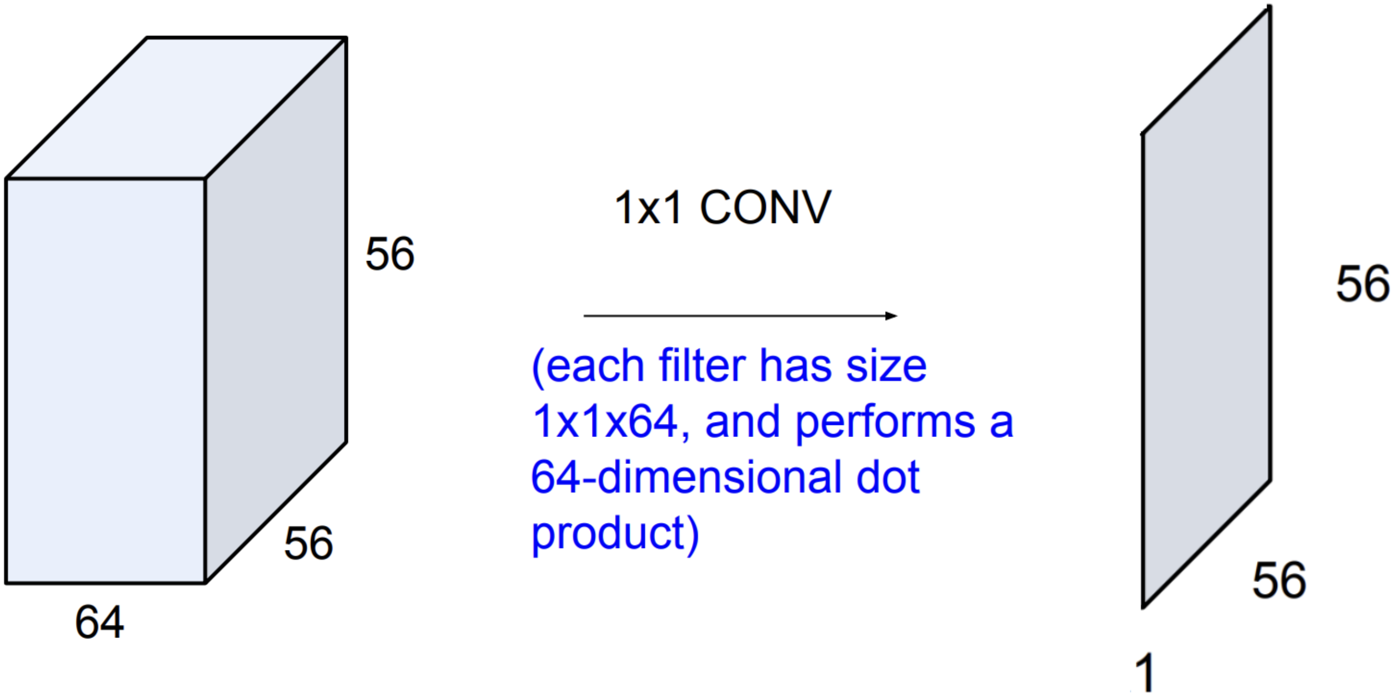


https://www.youtube.com/watch?v=VxhSouuSZDY



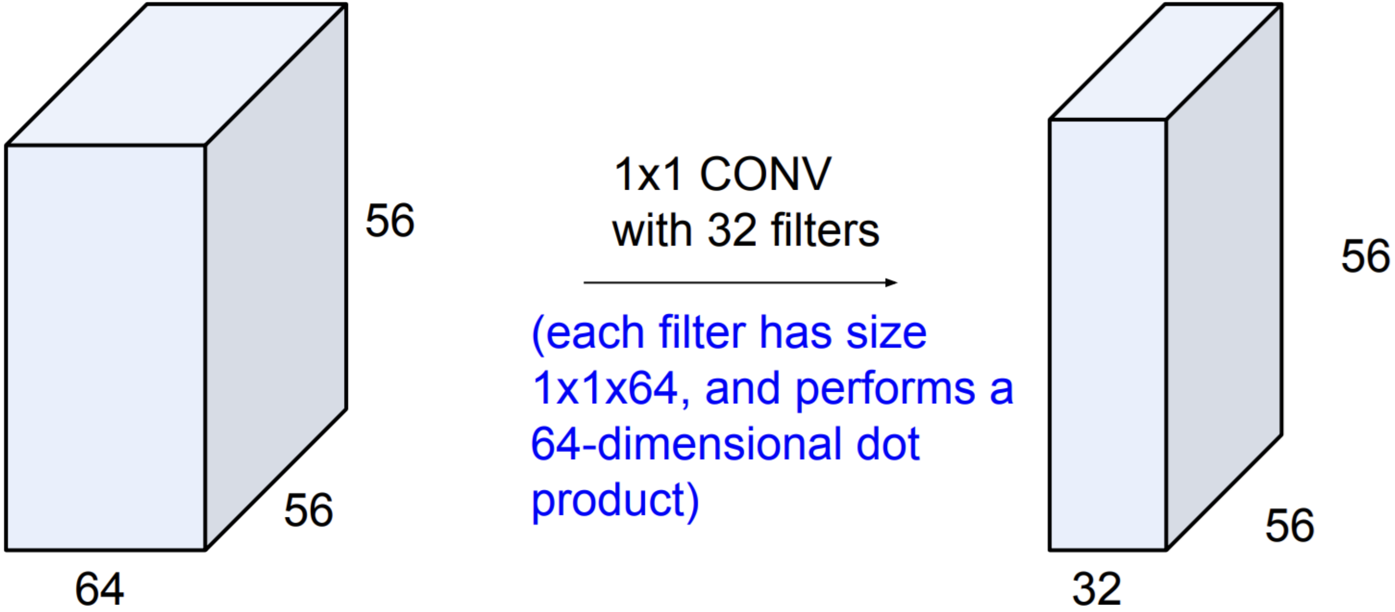
https://www.youtube.com/watch?v=VxhSouuSZDY

Why 1x1 convolution?



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Why 1x1 convolution?



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Why 1x1 convolution

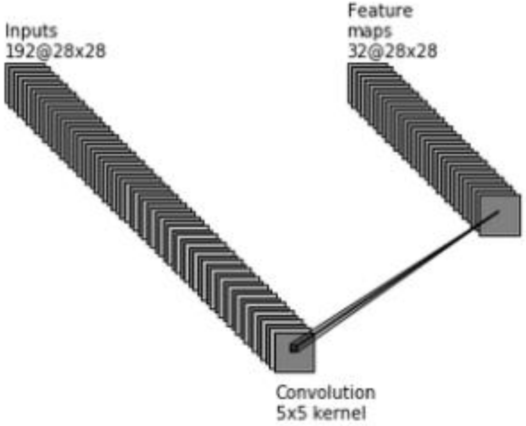


Figure 6. 5×5 convolutions inside the Inception module using the naive model

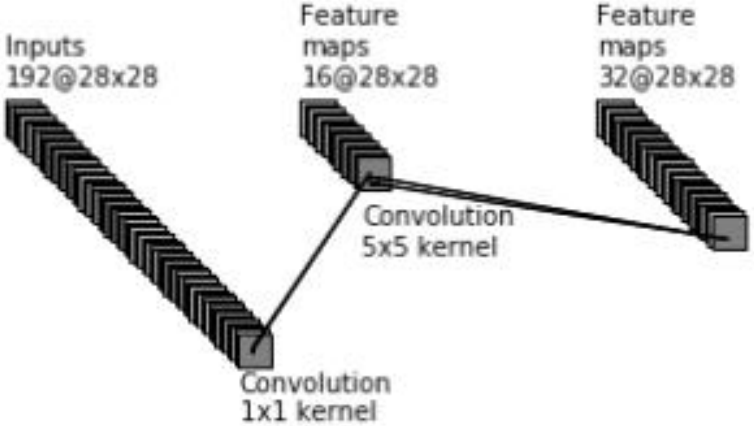


Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

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

Why 1x1 convolution

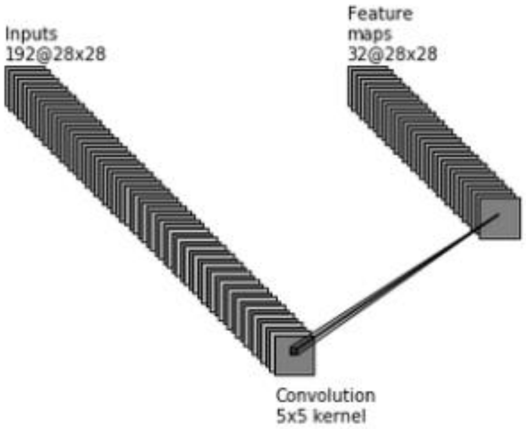


Figure 6. 5×5 convolutions inside the Inception module using the naive model

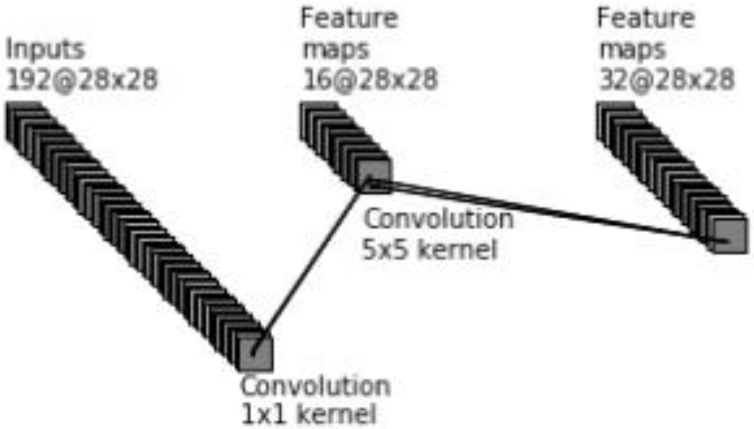


Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

Without 1x1 convolution:

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

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

Why 1x1 convolution

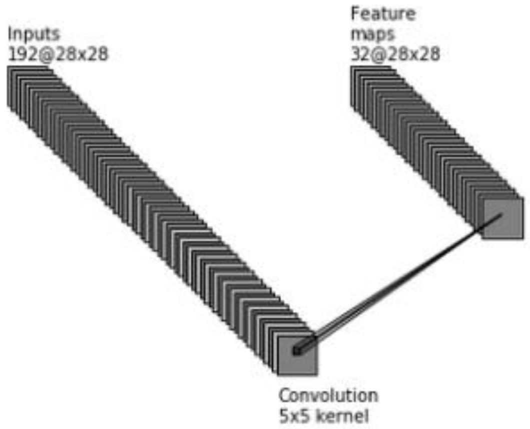


Figure 6. 5×5 convolutions inside the Inception module using the naive model

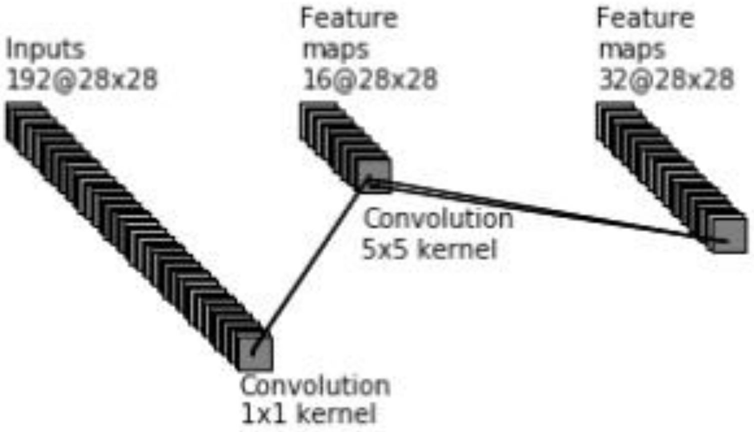


Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

Without 1x1 convolution:

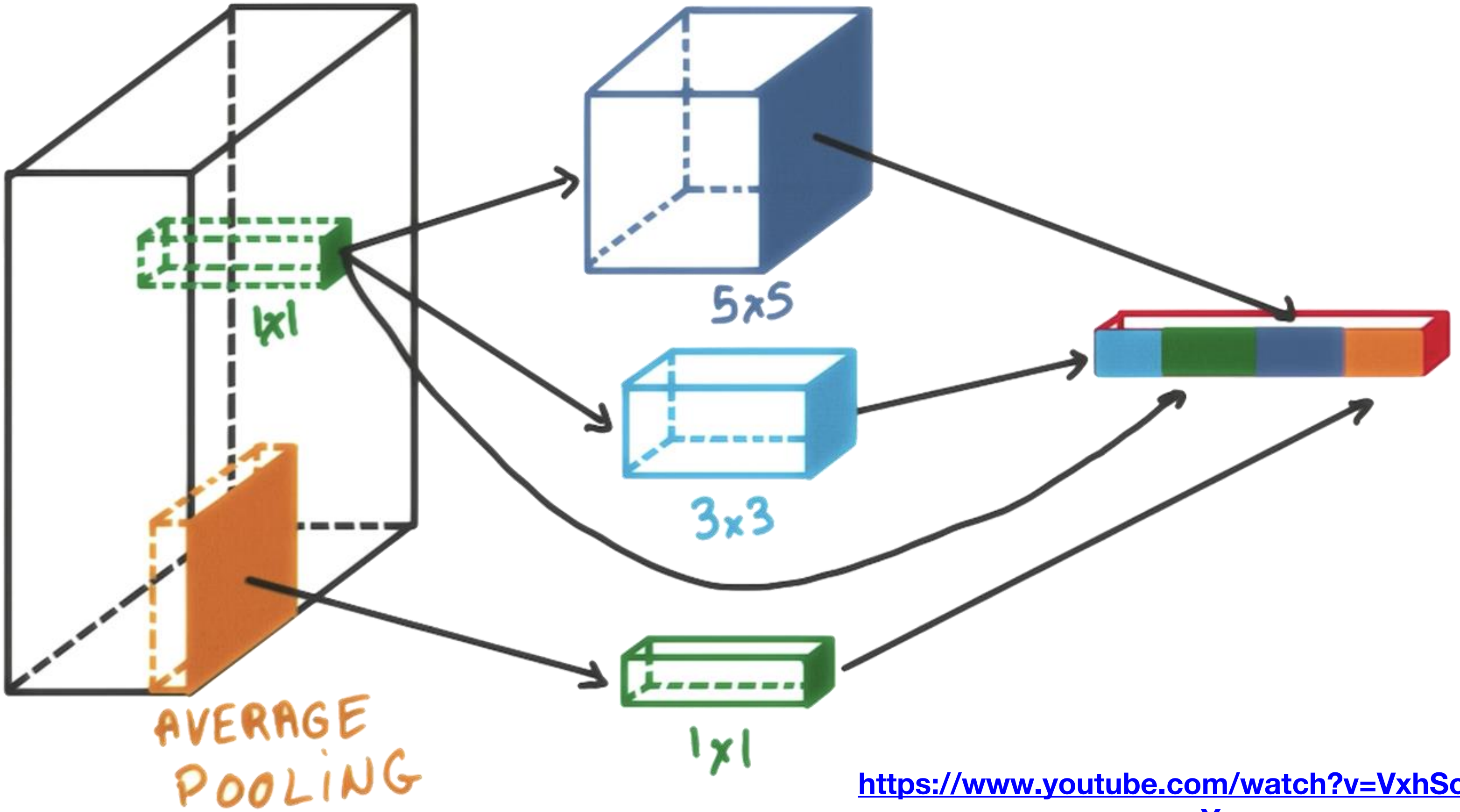
$5^2 * 28^2 * 192 * 32 = 120,422,400$ operations

With 1x1 convolution:

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

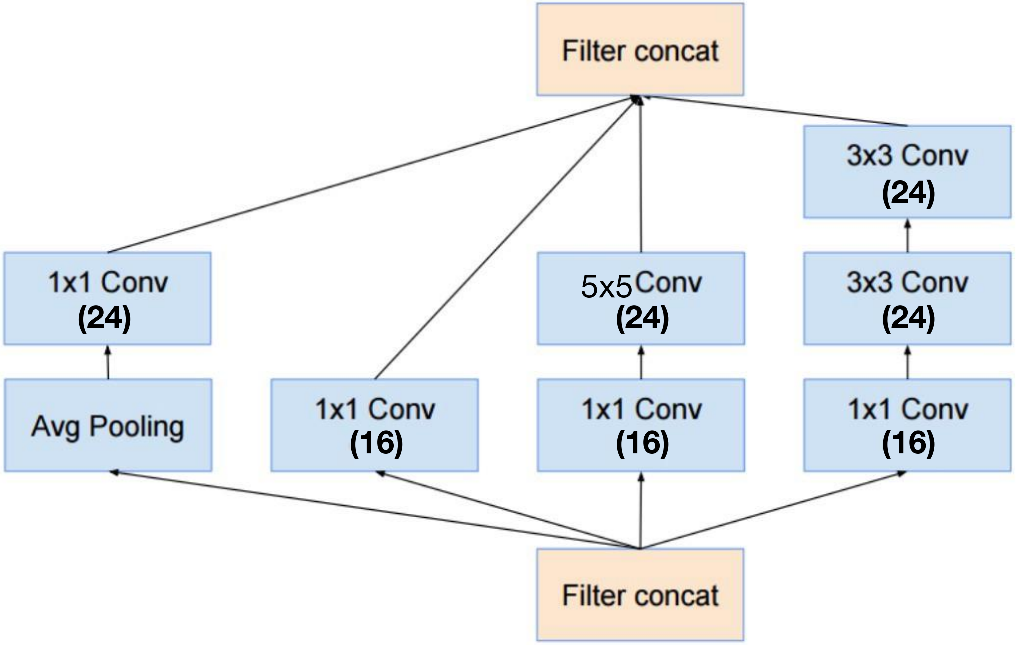
https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

INCEPTION MODULES

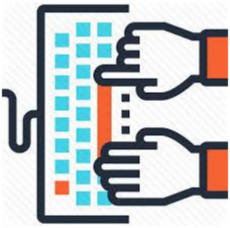


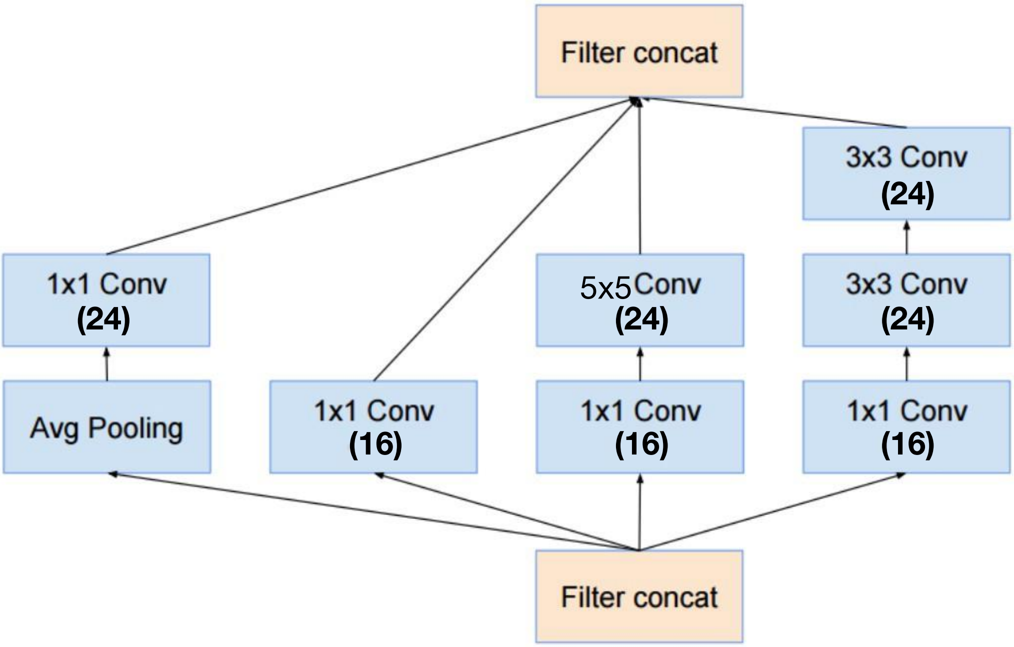
https://www.youtube.com/watch?v=VxhSouuSZDY

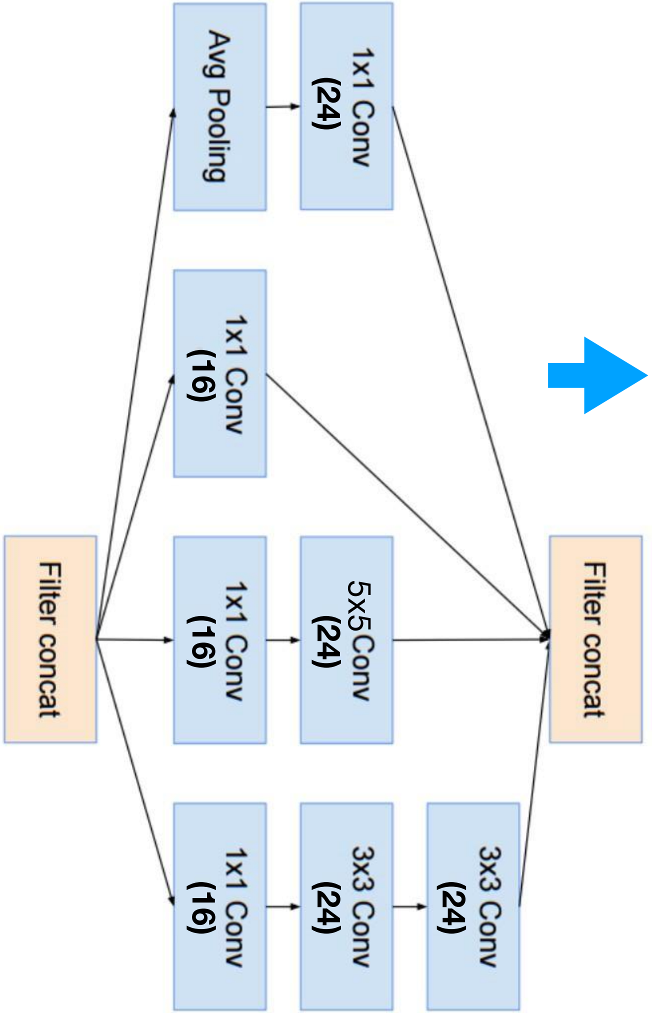
Inception Module



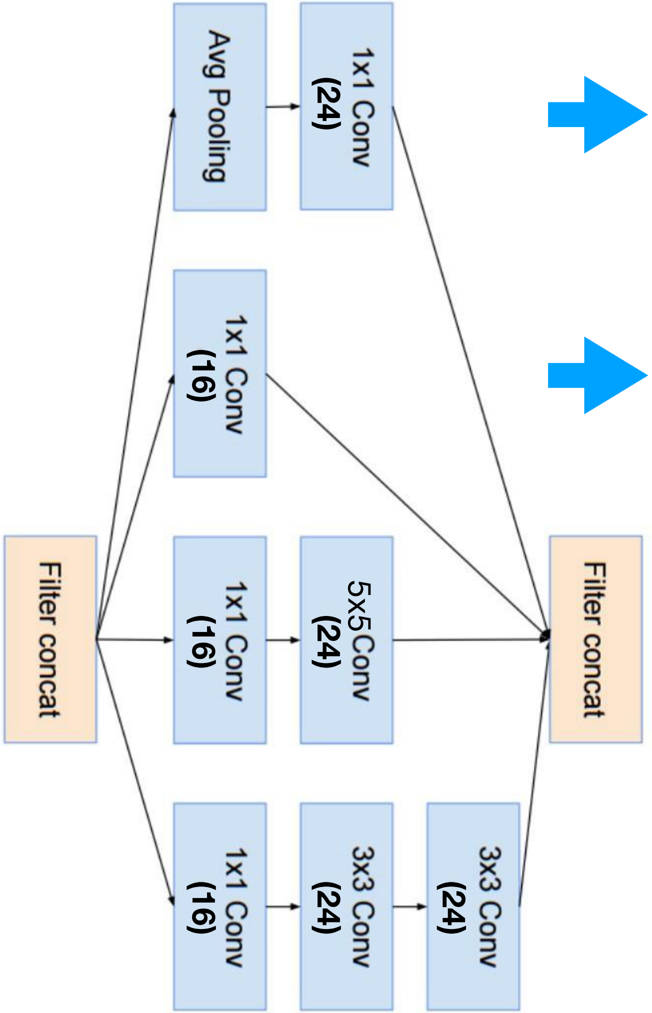
https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/




Filter concat
3x3 Conv
(24)
1x1 Conv
(24)
5x5 Conv
(24)
3x3 Conv
(24)
Avg Pooling
1x1 Conv
(16)
1x1 Conv
(16)
1x1 Conv
(16)
Filter concat

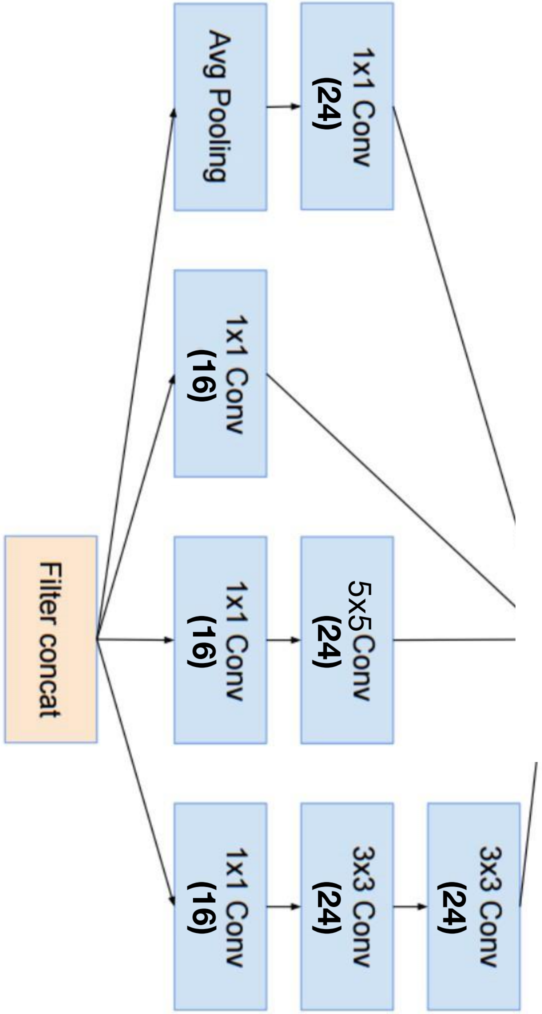



```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```




```
self.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)
```

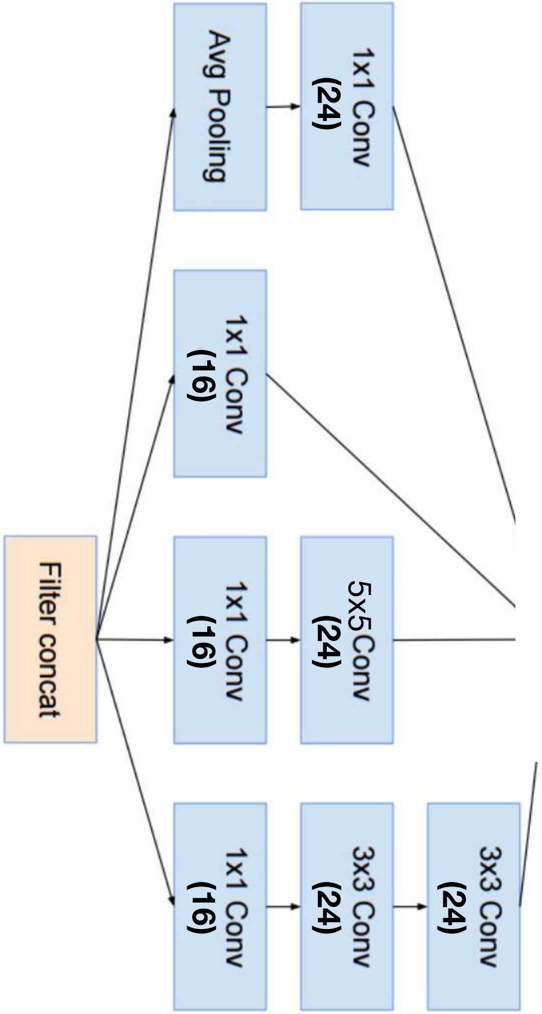
```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

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

branch1x1 = self.branch1x1(x)
```

```
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)
```

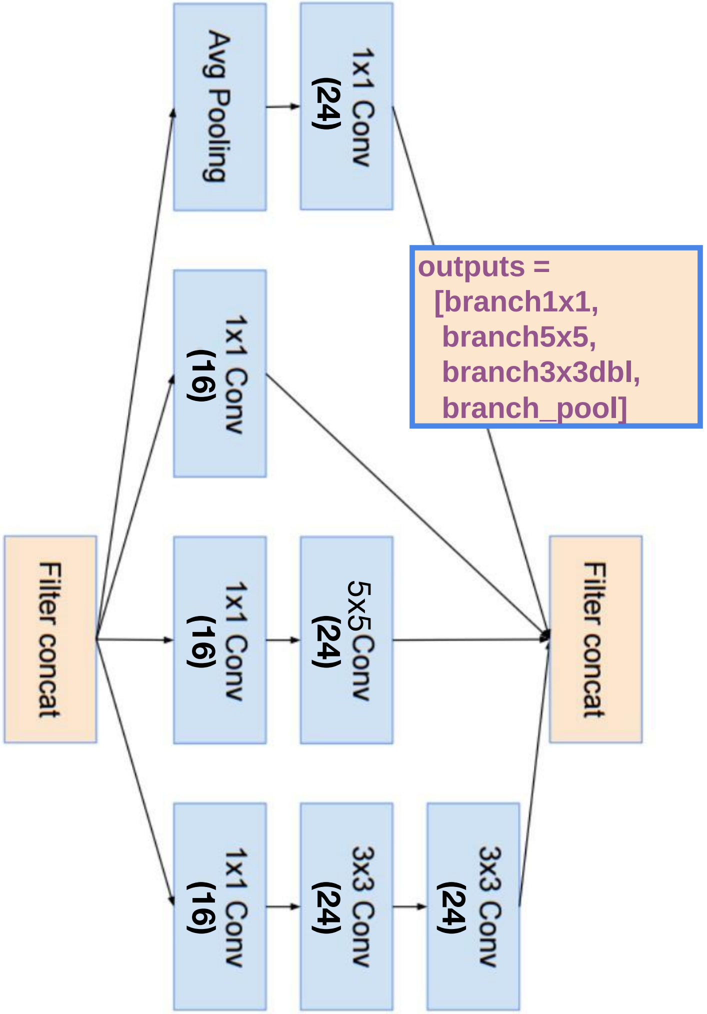



```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

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

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

branch1x1 = self.branch1x1(x)
```

```
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.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)
```



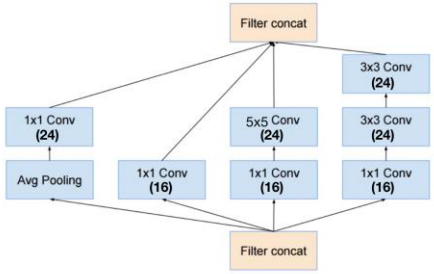

```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

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

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

branch1x1 = self.branch1x1(x)
```

```
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.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)
```

Inception Module

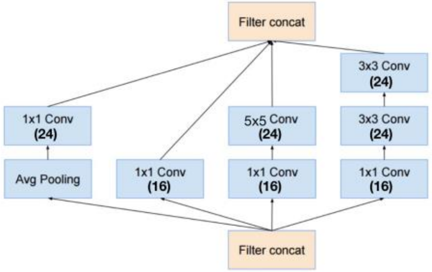



```python
class InceptionA(nn.Module):
    def __init__(self, in_channels):
        super(InceptionA, self).__init__()
        self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

        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

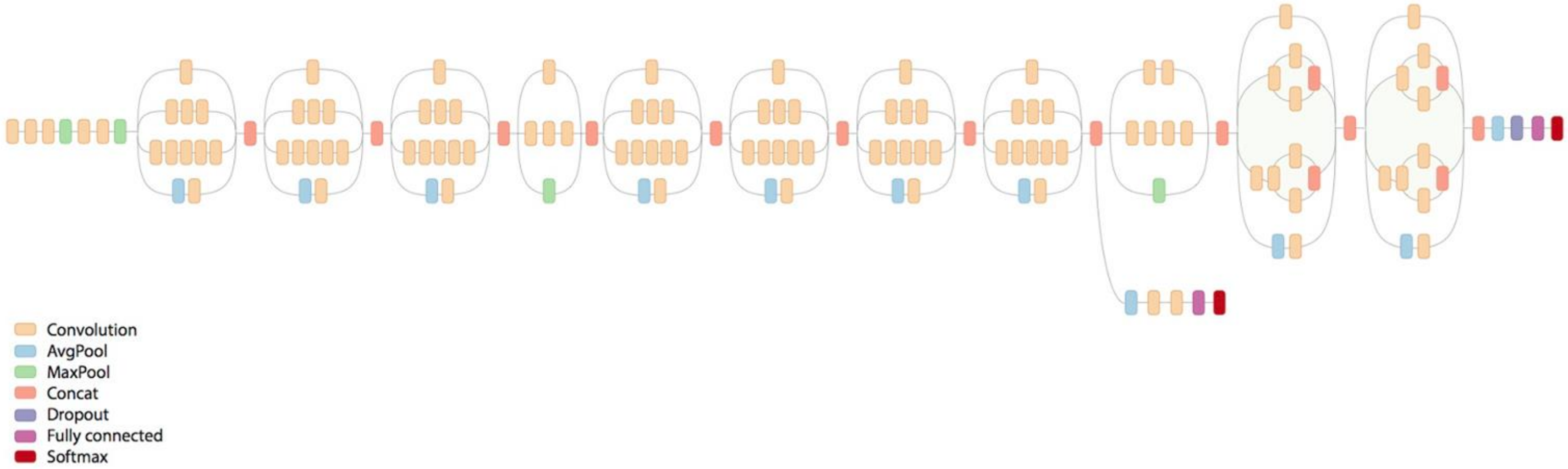


Train Epoch: 9 [44800/60000 (75%)] Loss: 0.064180
Train Epoch: 9 [45440/60000 (76%)] Loss: 0.020339
Train Epoch: 9 [46080/60000 (77%)] Loss: 0.061476
Train Epoch: 9 [46720/60000 (78%)] Loss: 0.039662
Train Epoch: 9 [47360/60000 (79%)] Loss: 0.026798
Train Epoch: 9 [48000/60000 (80%)] Loss: 0.071569
Train 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

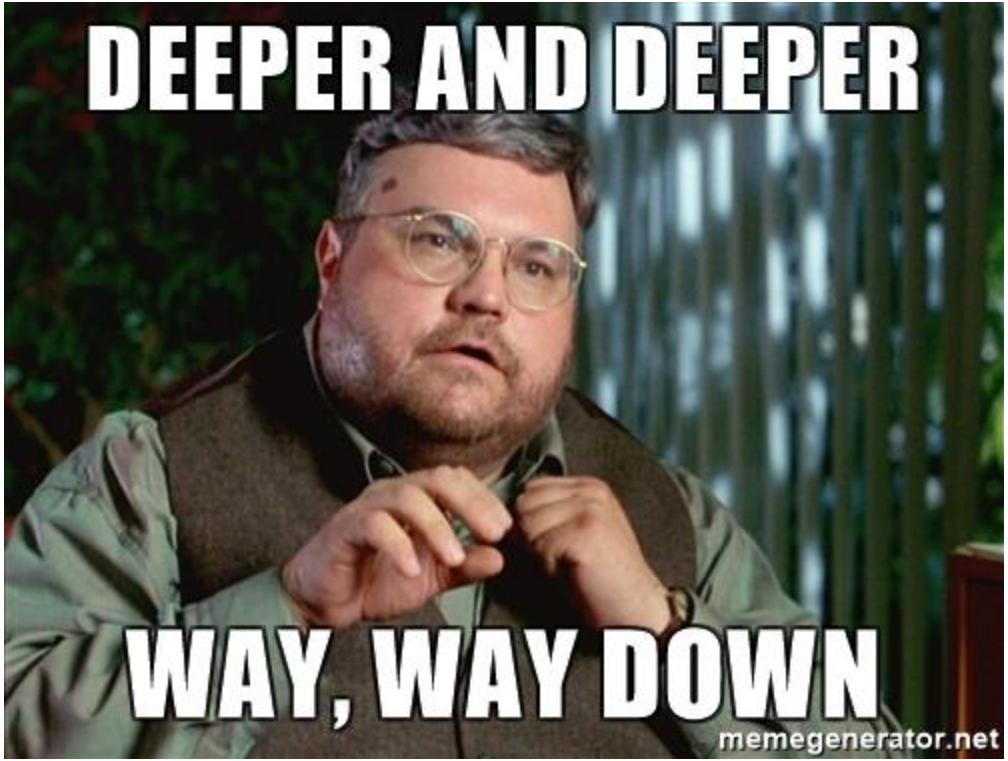
Test set: Average loss: 0.0470, Accuracy: 9866/10000 (99%)

```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 11-1: Implement full inception v3/v4

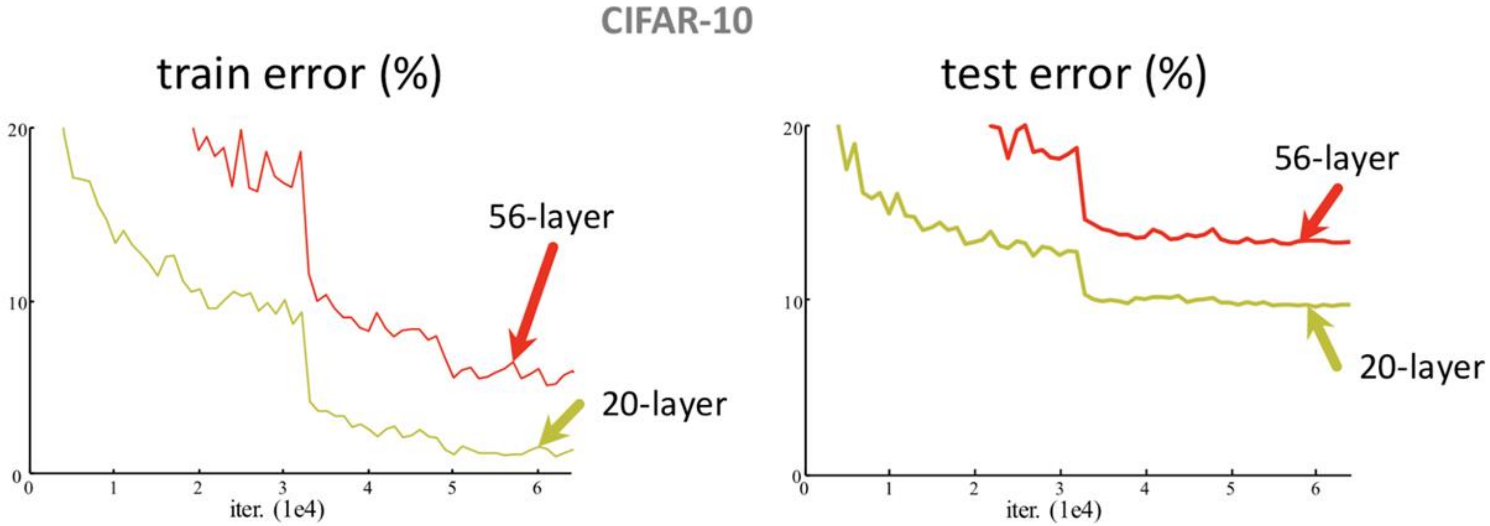


https://research.googleblog.com/2016/08/improving-inception-and-image.html



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

Can we just go deeper, keep stacking layers?

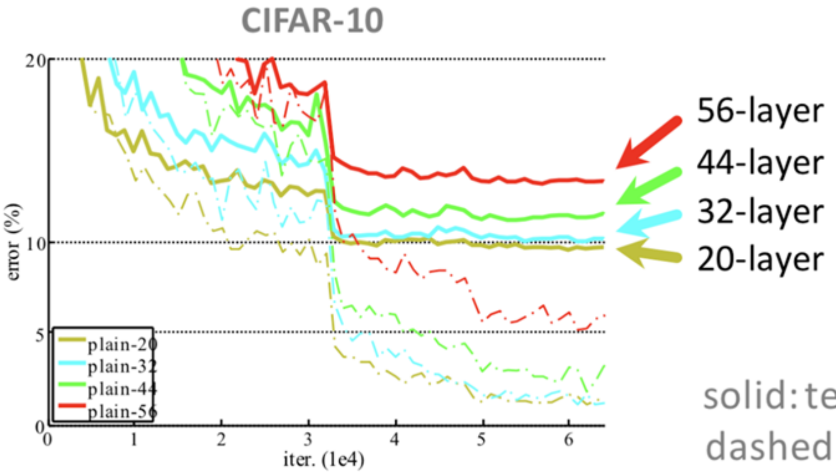


- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

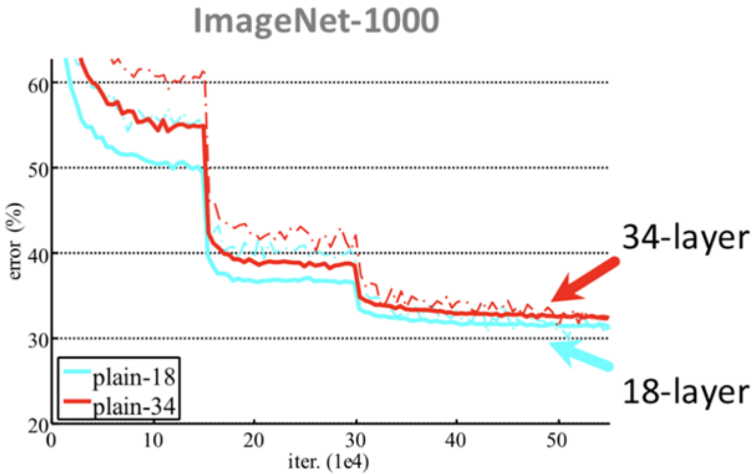
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

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

Can we just go deeper, keep stacking layers?



solid: test/val
dashed: train



- “Overly deep” plain nets have **higher training error**
- A general phenomenon, observed in many datasets

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

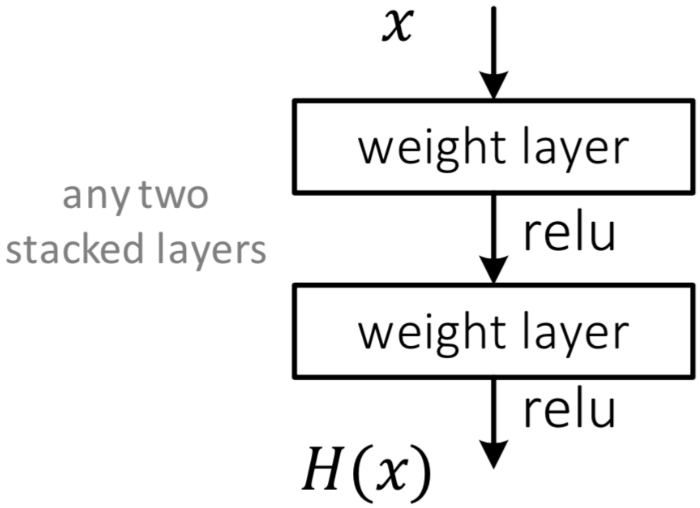
http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Problems with stacking layers (TBA)

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

Deep Residual Learning

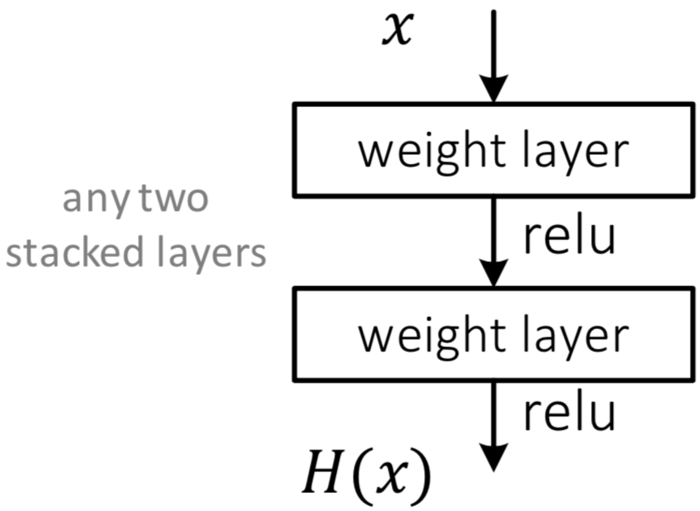
- Plaint net



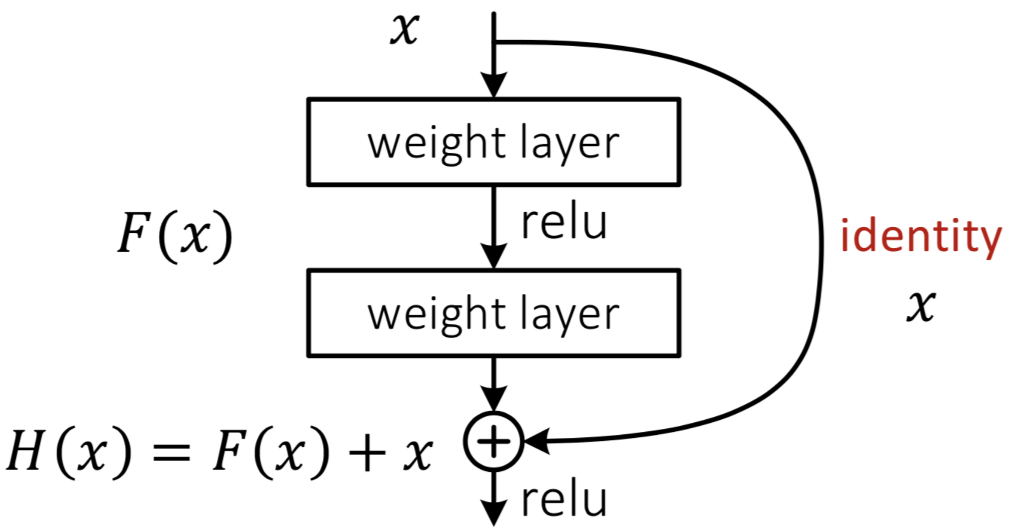
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

Deep Residual Learning

- Plaint net



- Residual net

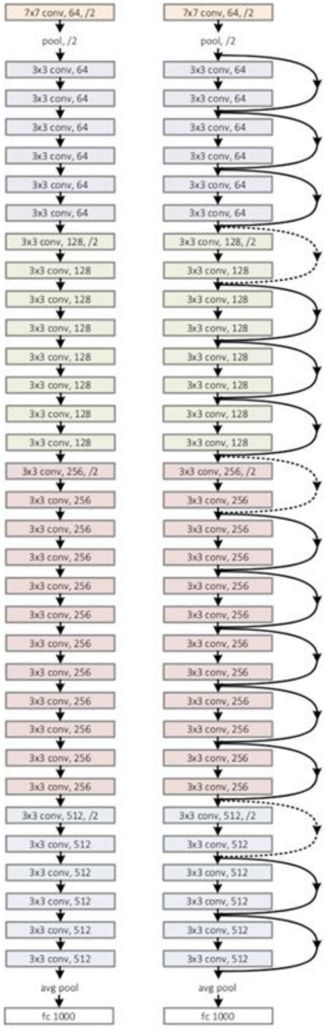


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

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



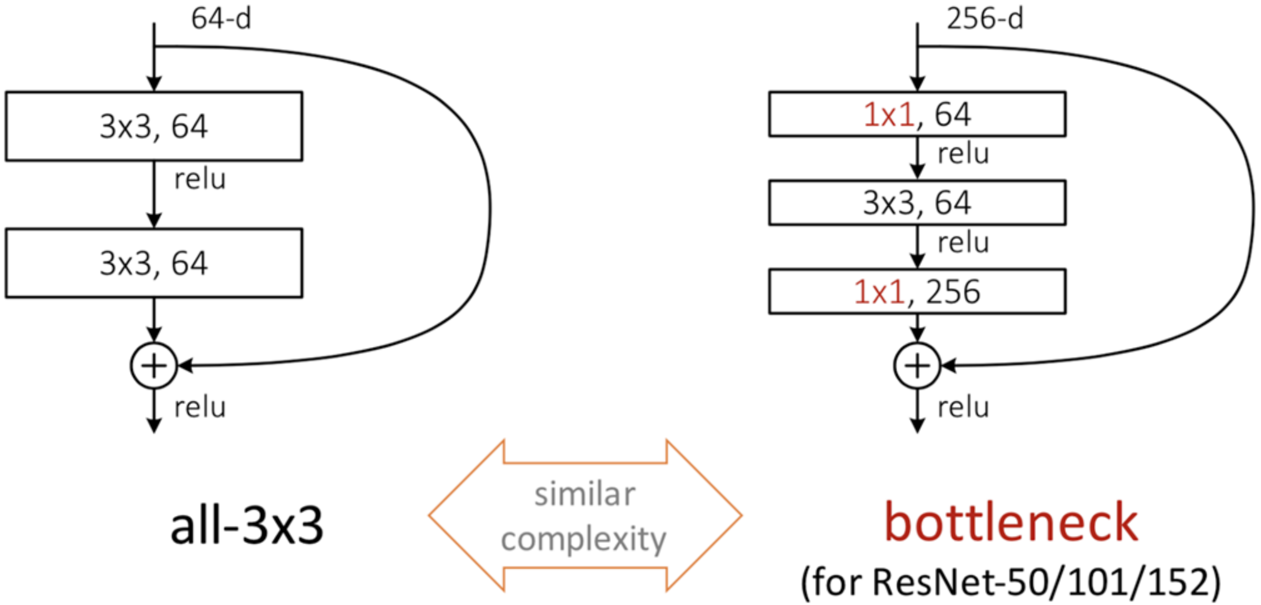
ResNet

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

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

ImageNet experiments

- A practical design of 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

Many more experiments!

* ResNet-110 on CIFAR-10

$h(x) = x$

error: 6.6%

$h(x) = 0.5x$

error: 12.4%

$h(x) = \text{gate} \cdot x$

error: 8.7%

*similar to "Highway Network"

$h(x) = \text{gate} \cdot x$

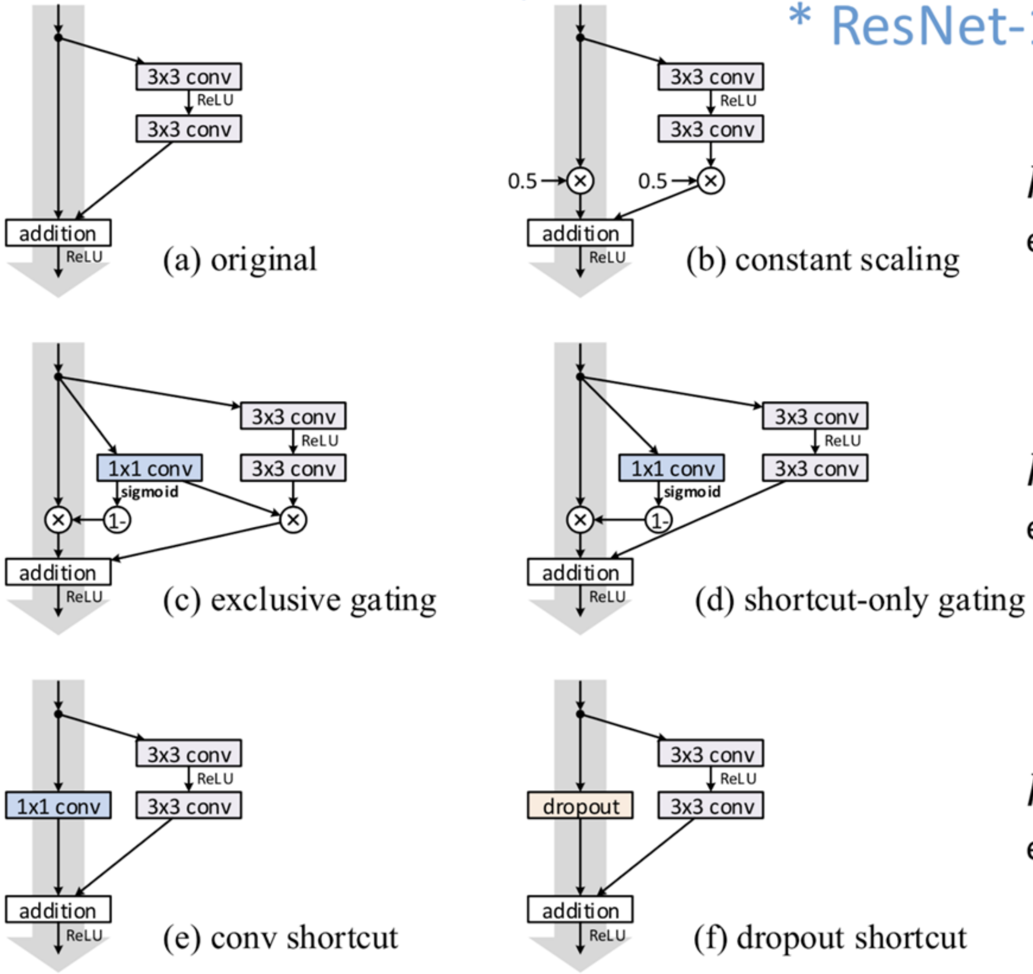
error: 12.9%

$h(x) = \text{conv}(x)$

error: 12.2%

$h(x) = \text{dropout}(x)$

error: > 20%



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

ResNets @ ILSVRC & COCO 2015 Competitions

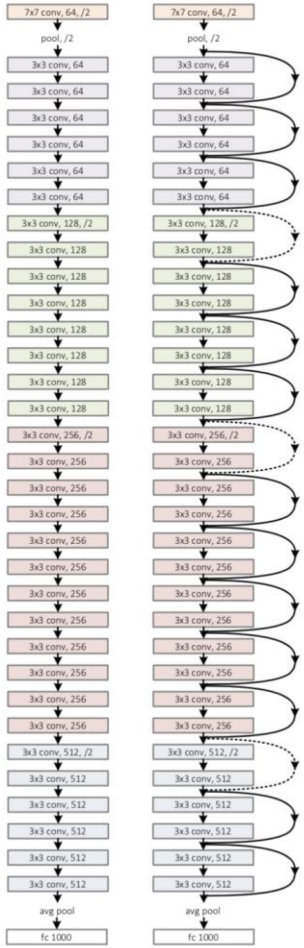
- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Exercise 11-2: Implement ResNet

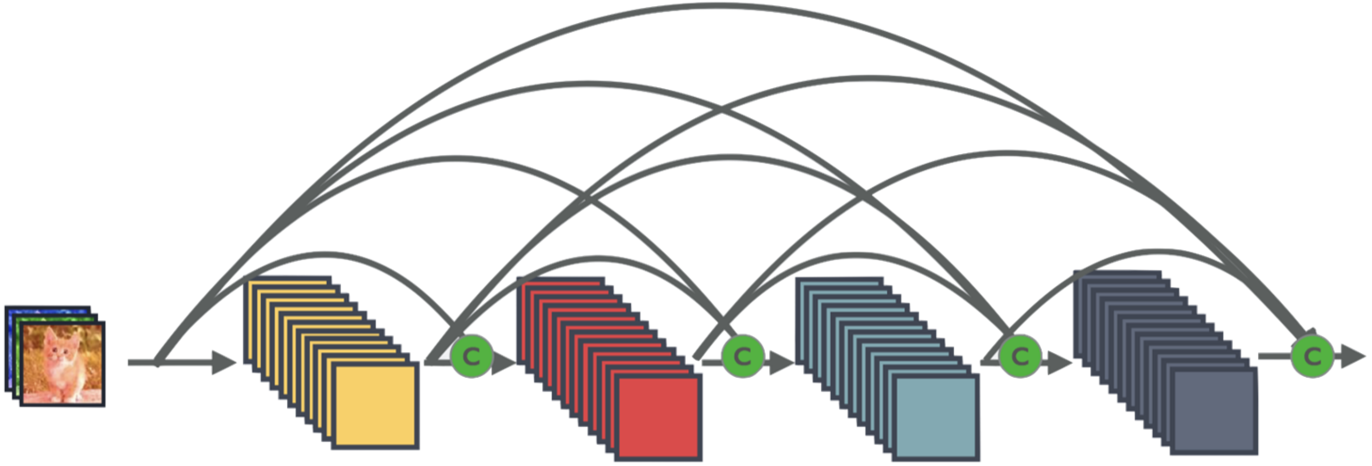
plain net



ResNet

- Try to implement from scratch
- Deep Residual Learning for Image Recognition: https://arxiv.org/abs/1512.03385
- Identity Mappings in Deep Residual Networks: https://arxiv.org/abs/1603.05027
- http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

Exercise 11-3: Implement DenseNet



https://arxiv.org/abs/1608.06993

WHAT NEXT

Lecture 12: RNN