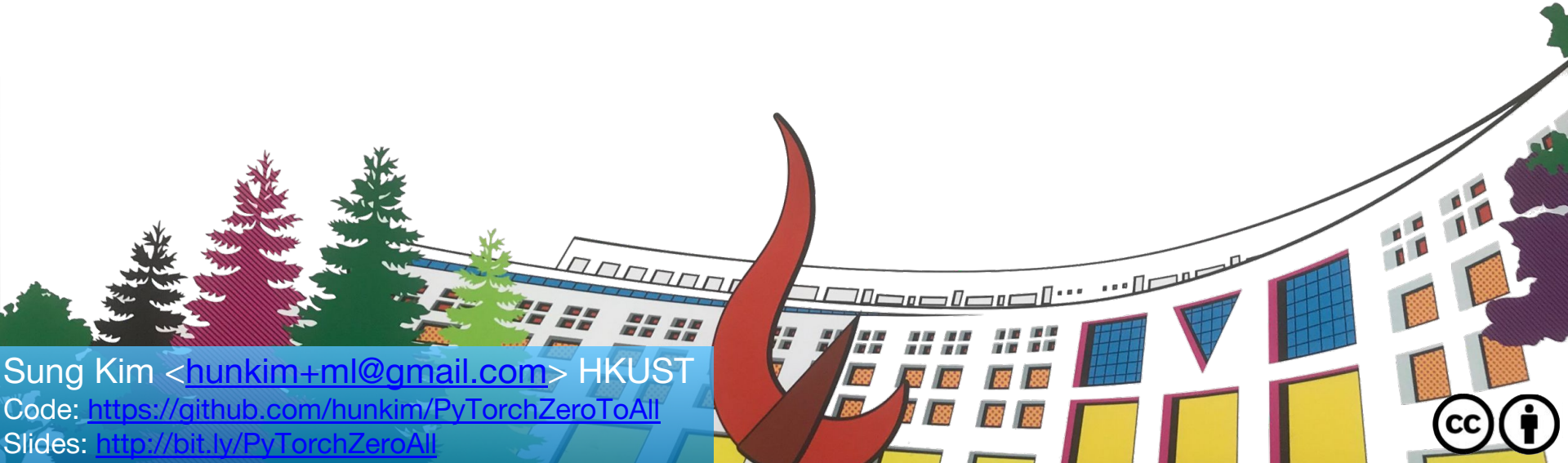
ML/DL for Everyone with PYTORCH

Lecture 10: CNN



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

Call for Comments

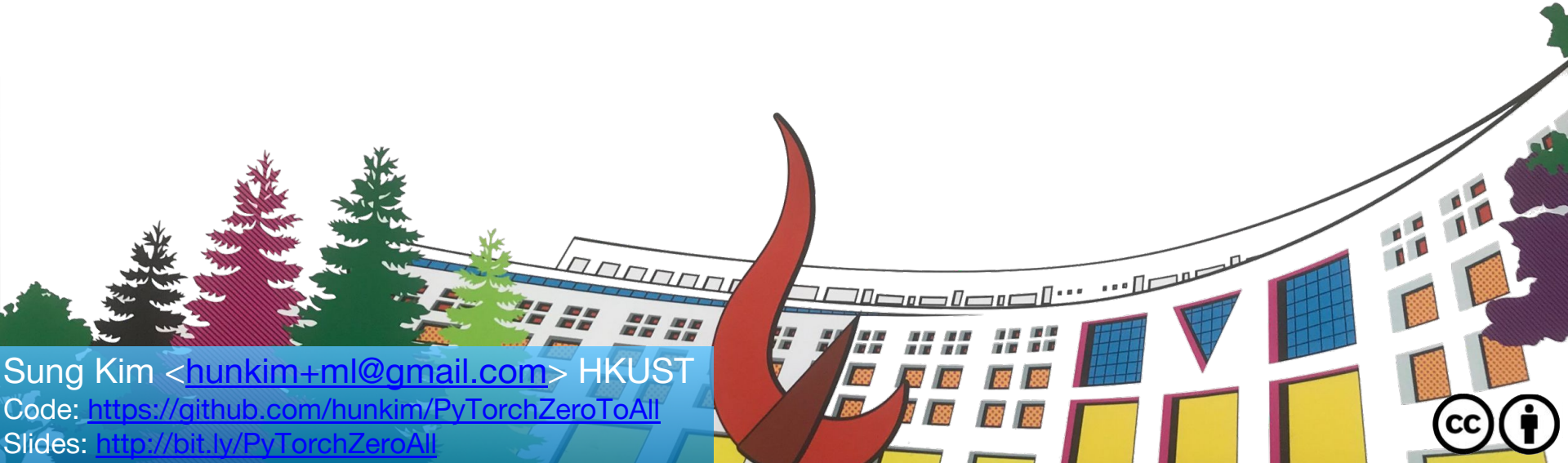
Please feel free to add comments directly on these slides.

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

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

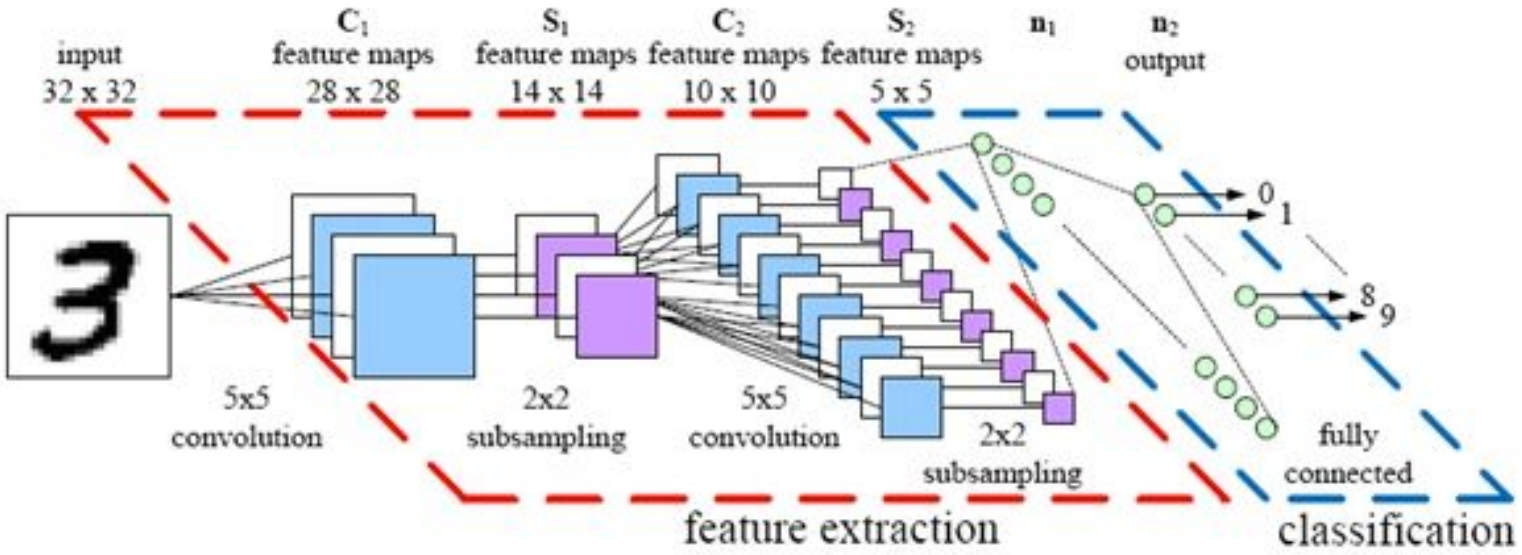
ML/DL for Everyone with PYTORCH

Lecture 10: CNN



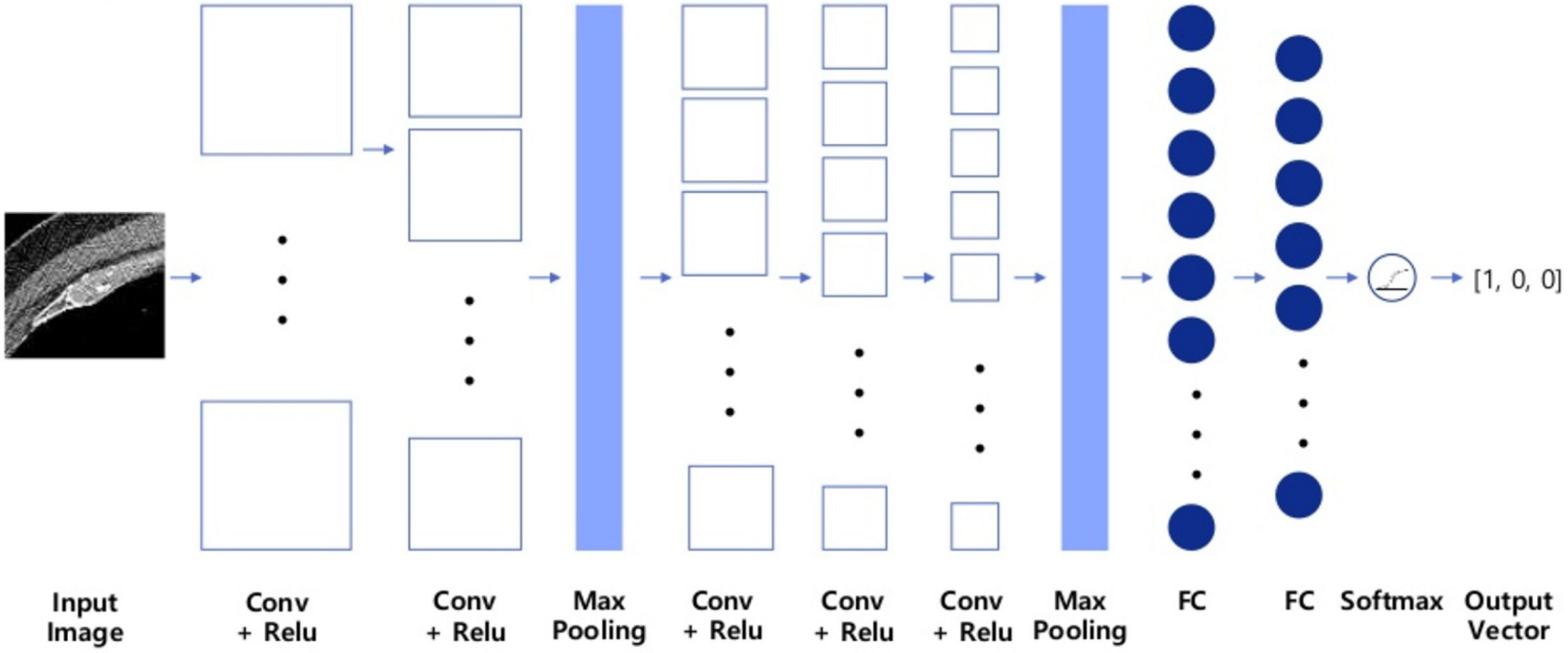
Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

CNN



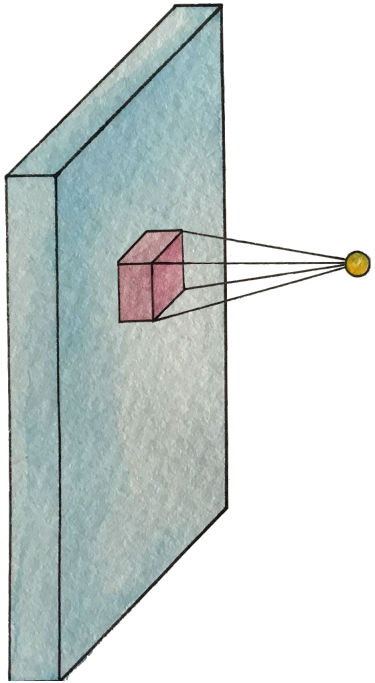
http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

CNN for CT images



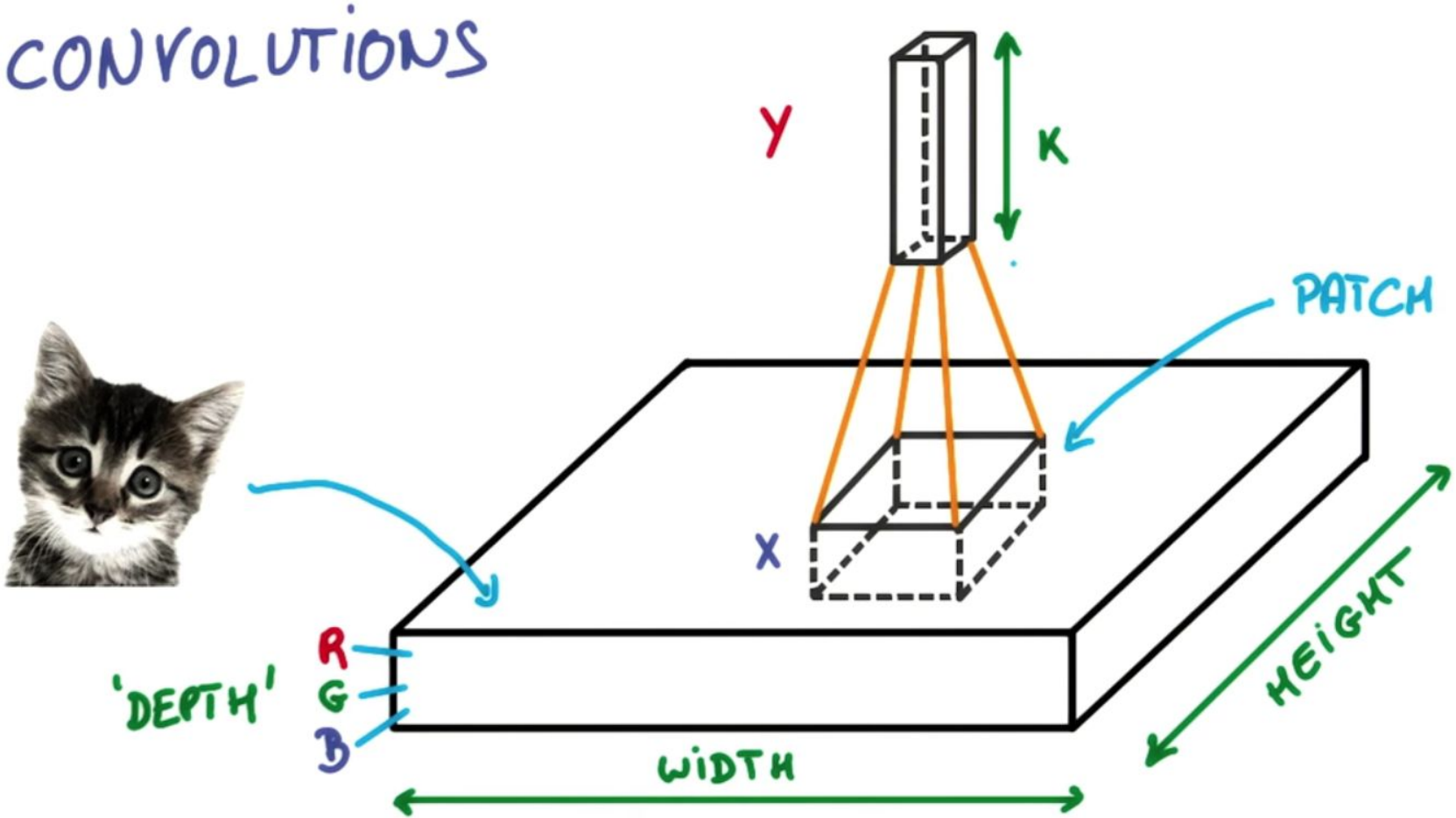
Asan Medical Center & Microsoft Medical Bigdata Contest Winner by GeunYoung Lee and Alex Kim
https://www.slideshare.net/GYLee3/ss-72966495

Convolution layer and max pooling



Single depth slice

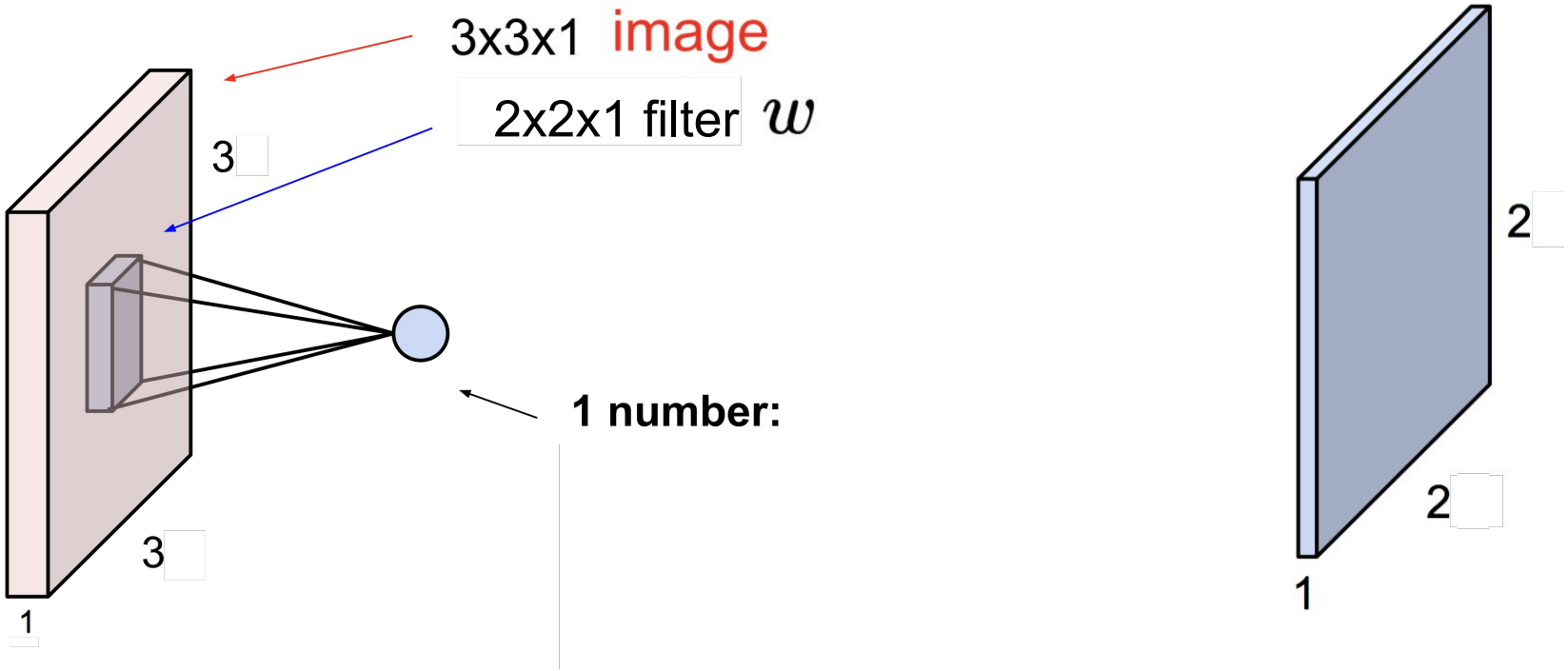
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4



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

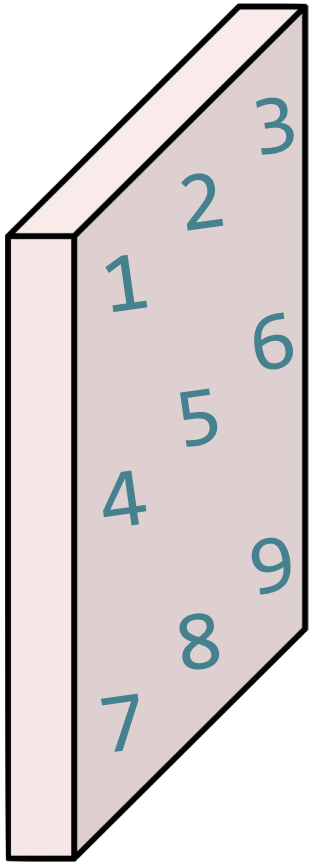
Simple convolution layer

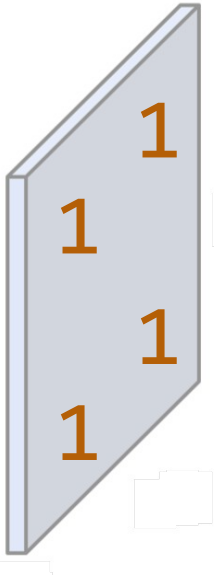
`Stride: 1x1`

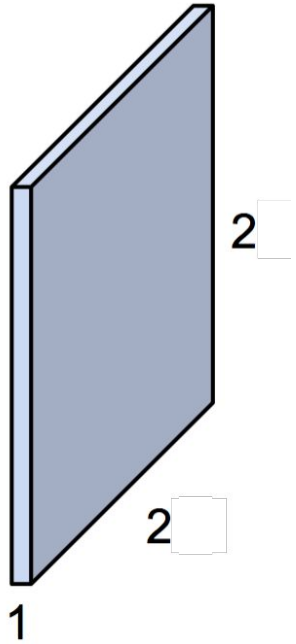


Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

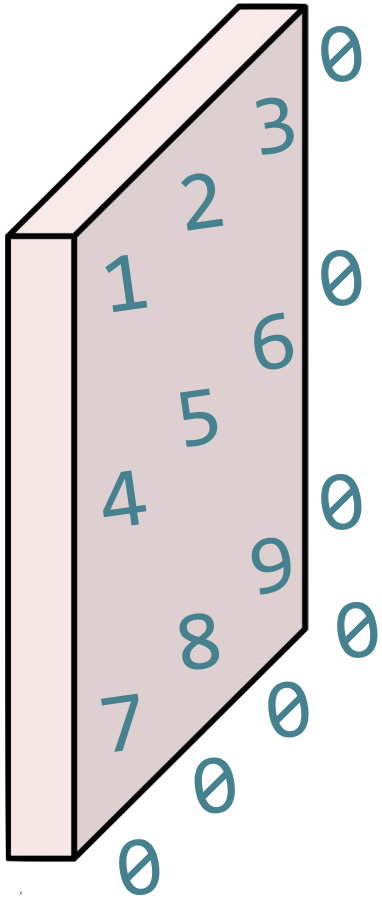


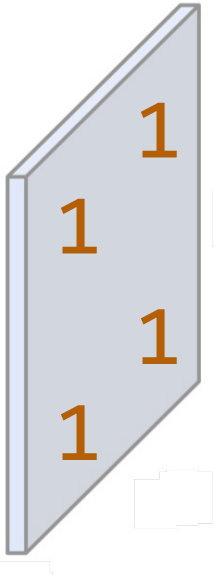


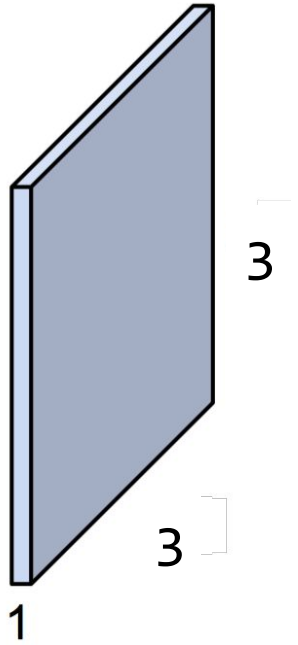


Simple convolution layer

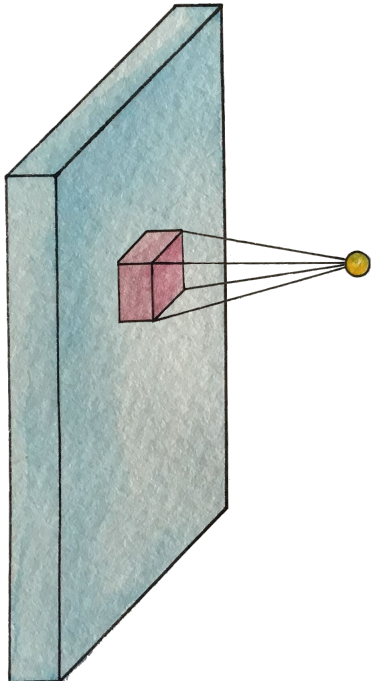
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: **SAME**







Max pooling



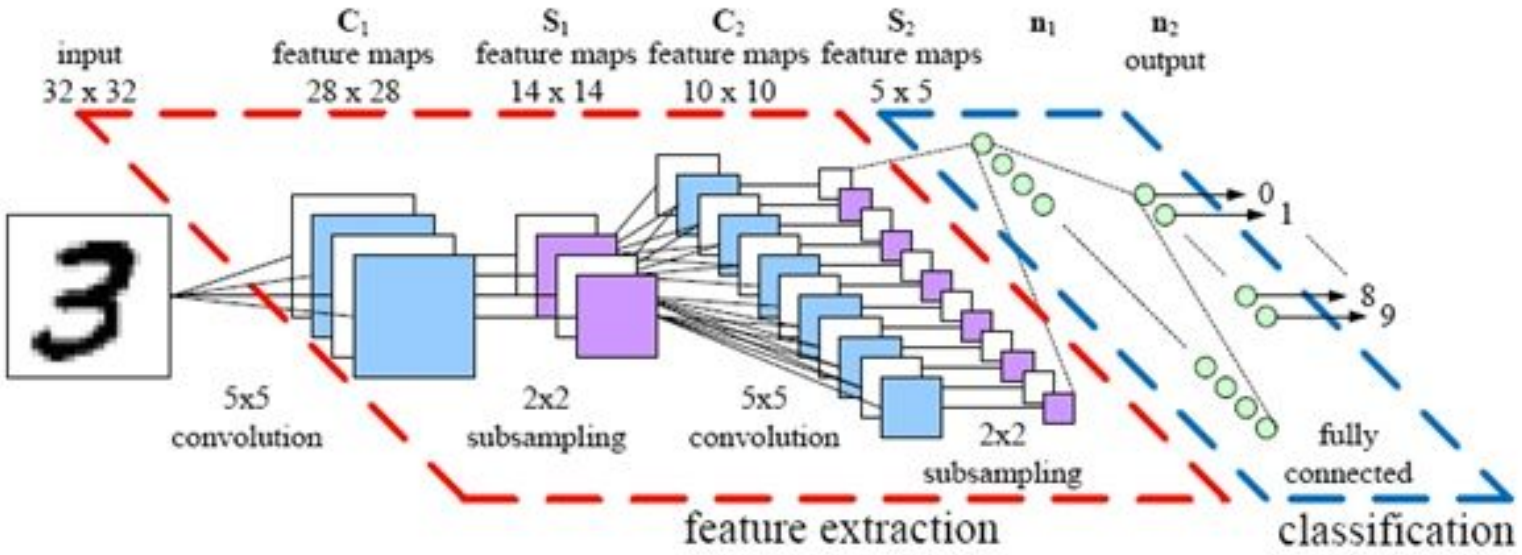
Single depth slice

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

max pool with 2x2 filters
and stride 2

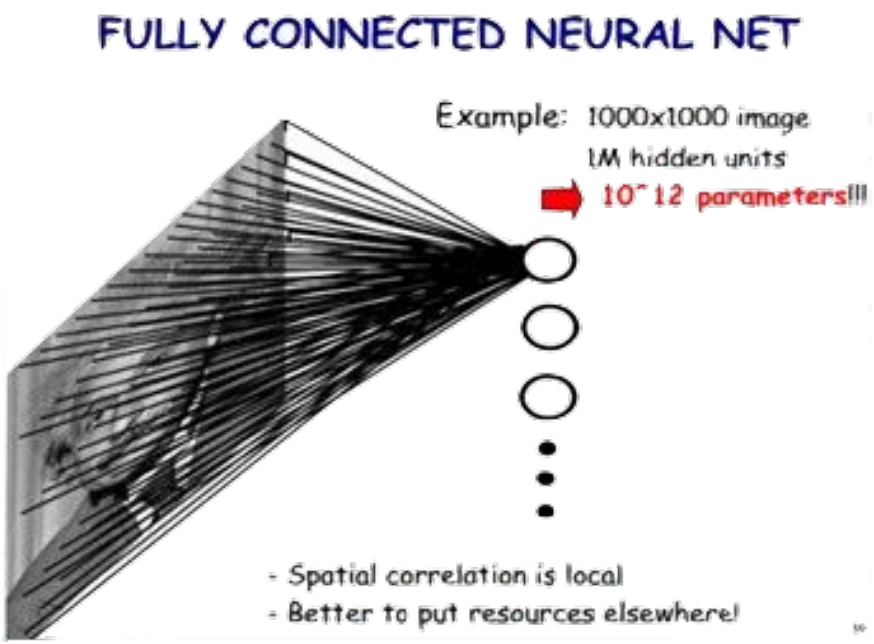
6	8
3	4

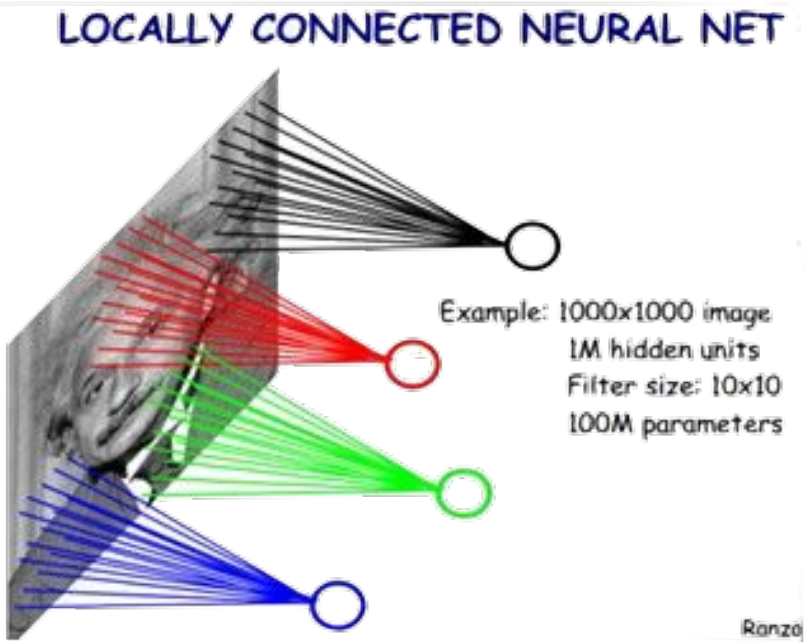
CNN



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

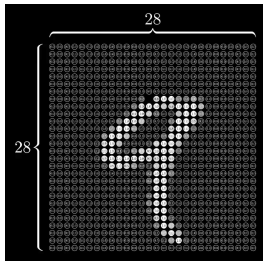
Locally Connected Features

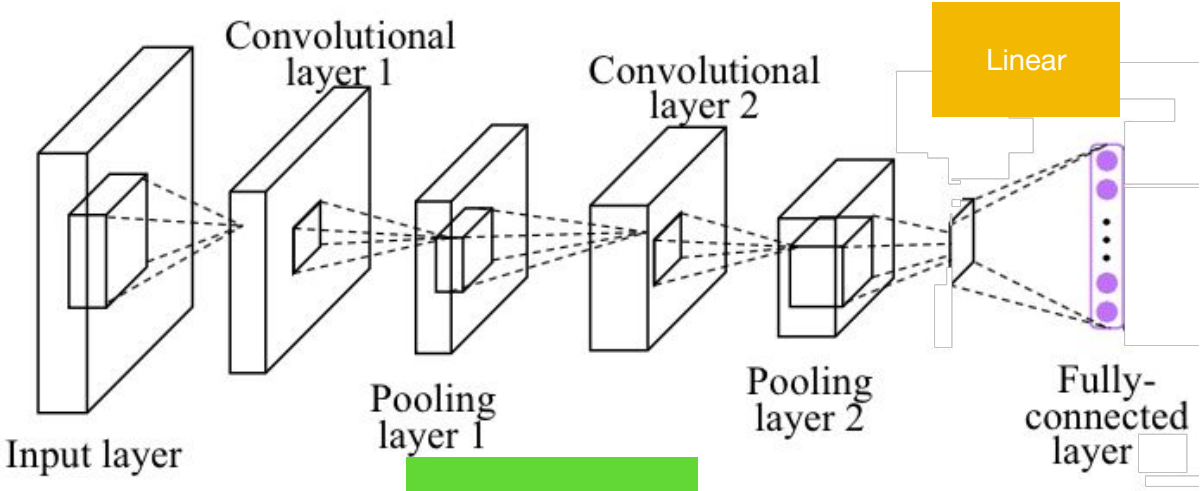




https://ireneli.eu/2016/02/03/deep-learning-05-talk-about-convolutional-neural-network%EF%BC%88cnn%EF%BC%89/

Simple CNN





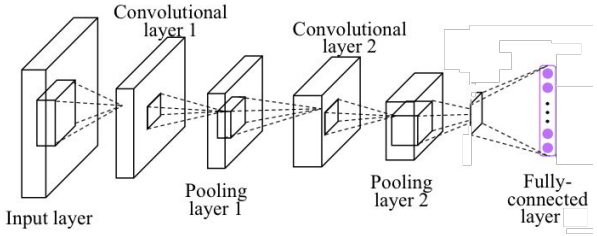

```
self.fc = nn.Linear(320, 10)
```

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

nn.MaxPool2d (kernel_size)

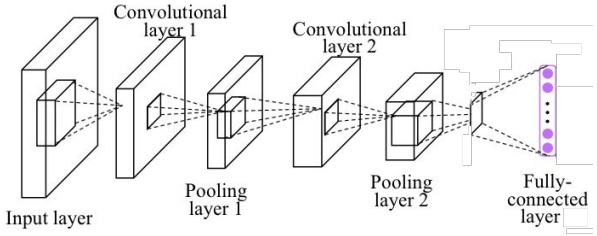
```
self.mp = nn.MaxPool2d(2)
```

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



Simple CNN

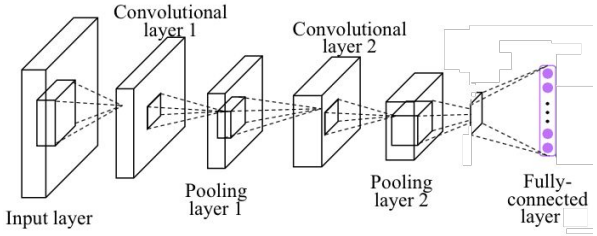
```
class Net(nn.Module):

    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)
```

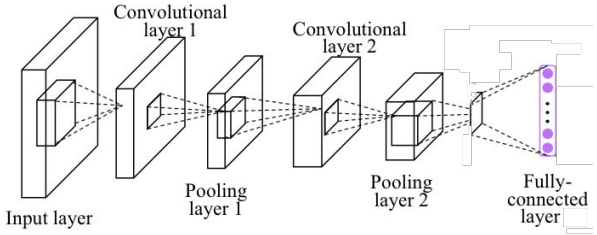
RuntimeError: size mismatch, m1: [64 x **320**], m2: [100 x 10]



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]

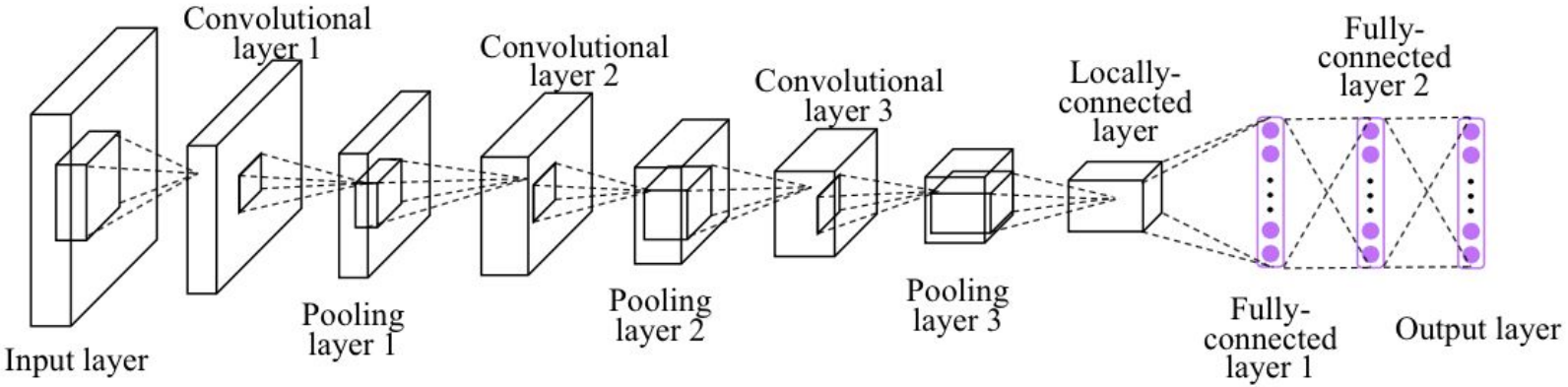


Simple CNN

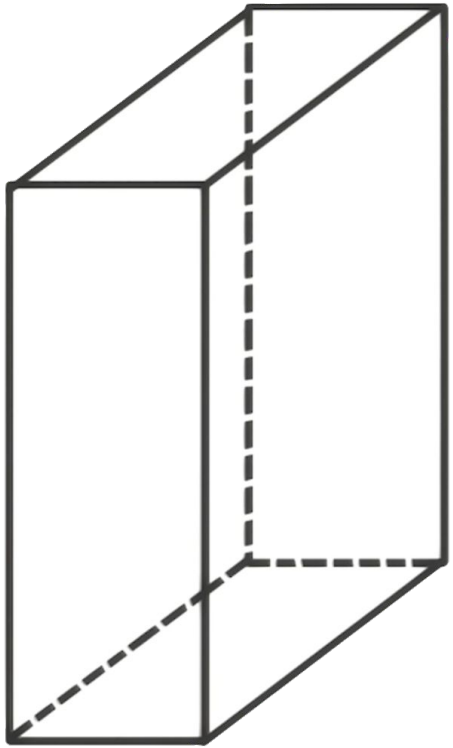
```
class Net(nn.Module):

    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

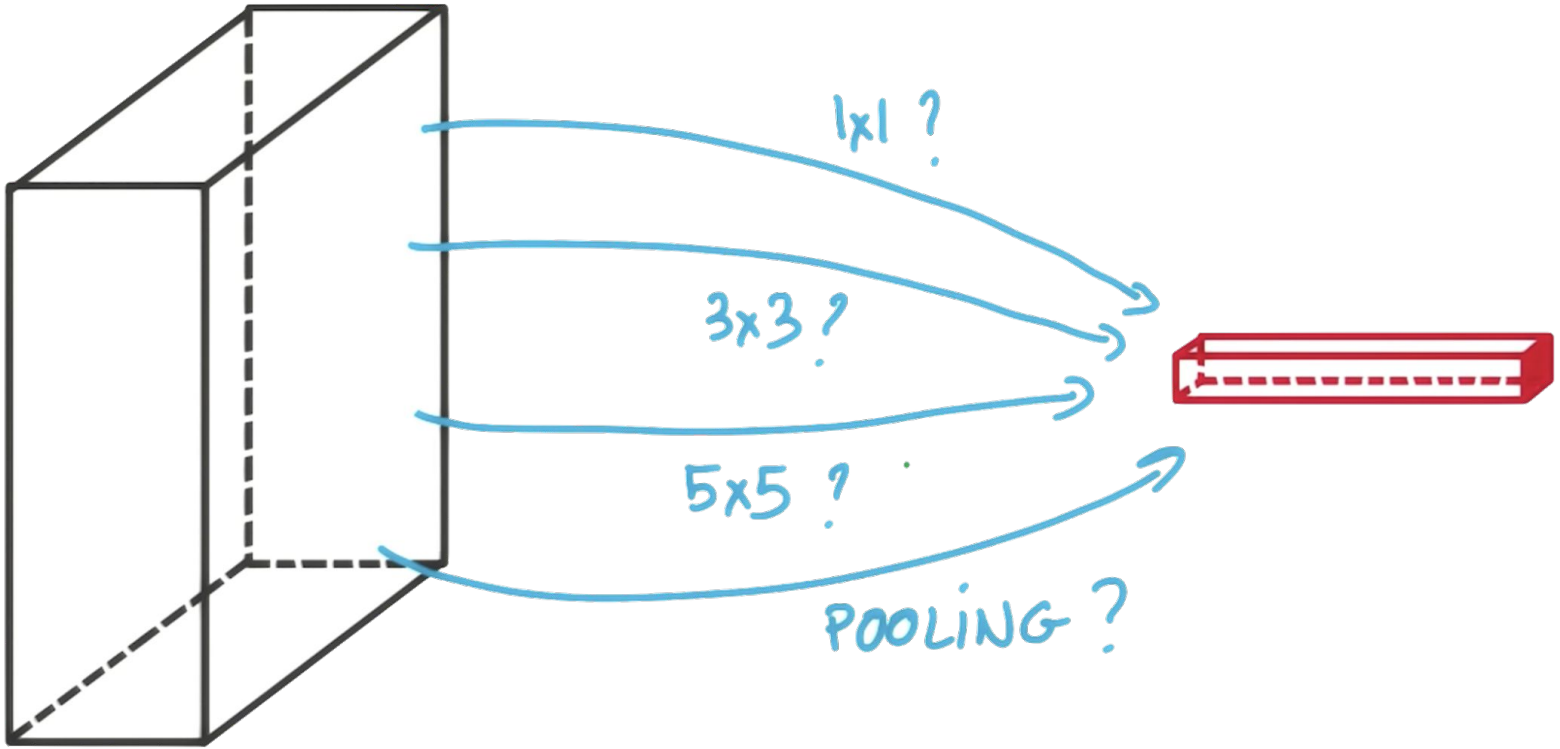


INCEPTION MODULES

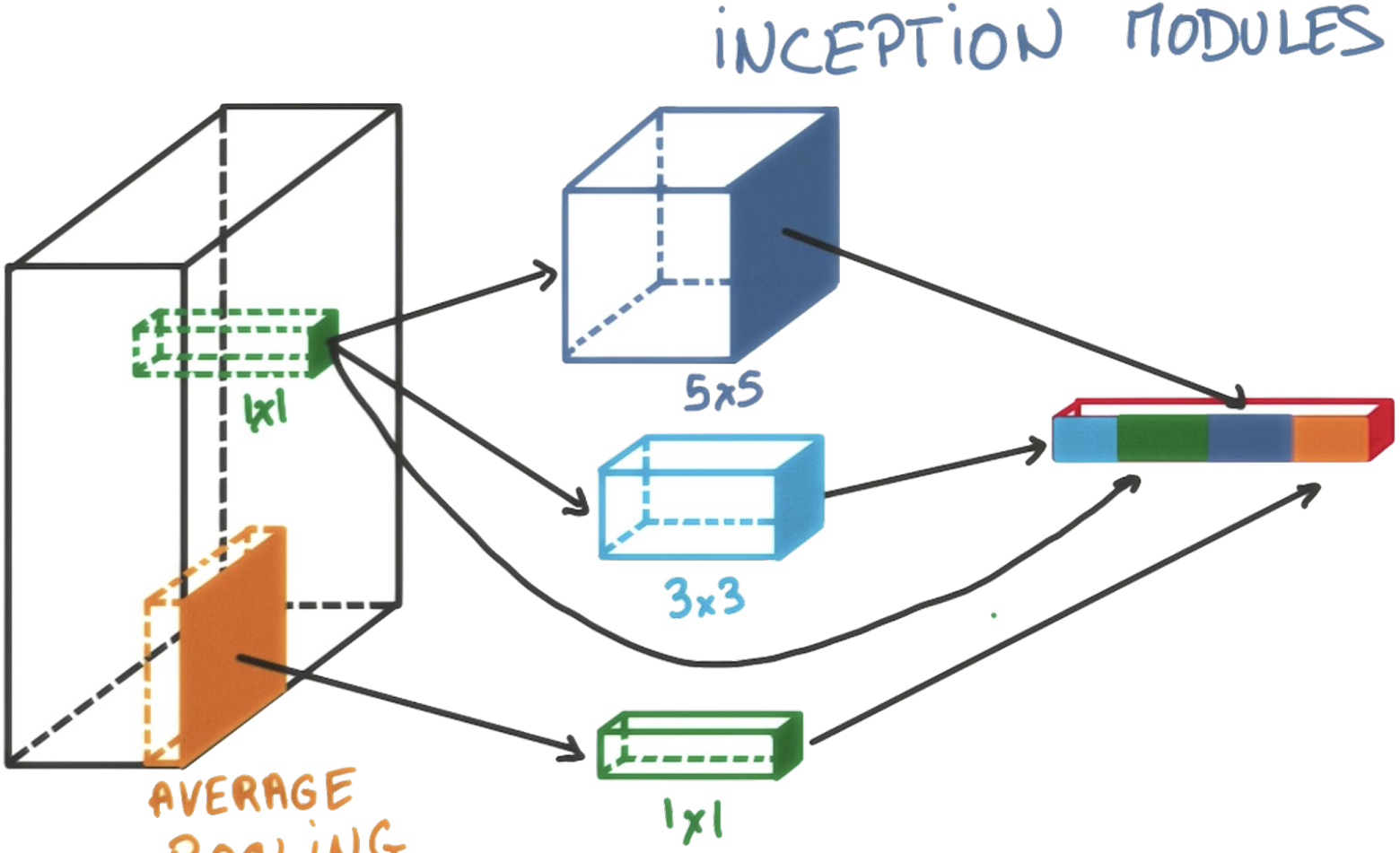


https://www.youtube.com/watch?v=VxhSouuSZDY

INCEPTION MODULES

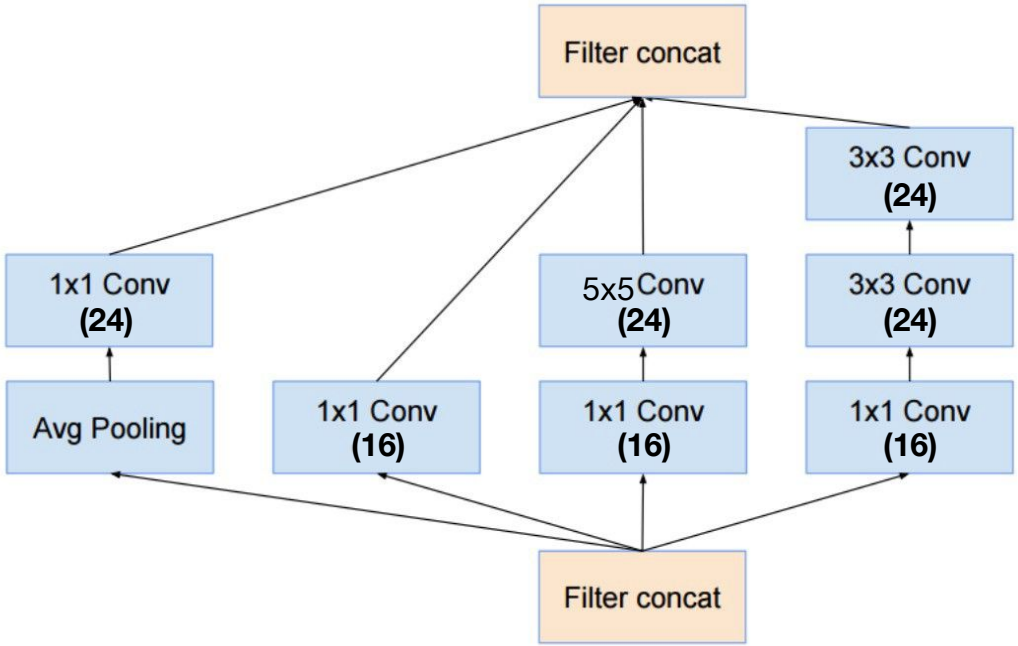


https://www.youtube.com/watch?v=VxhSouuSZDY

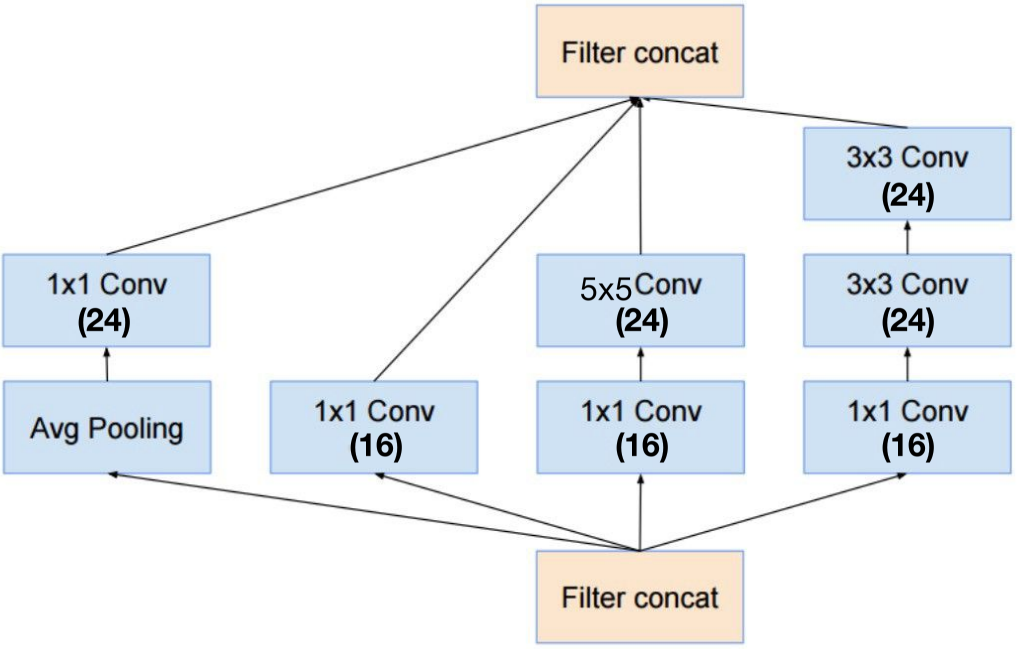


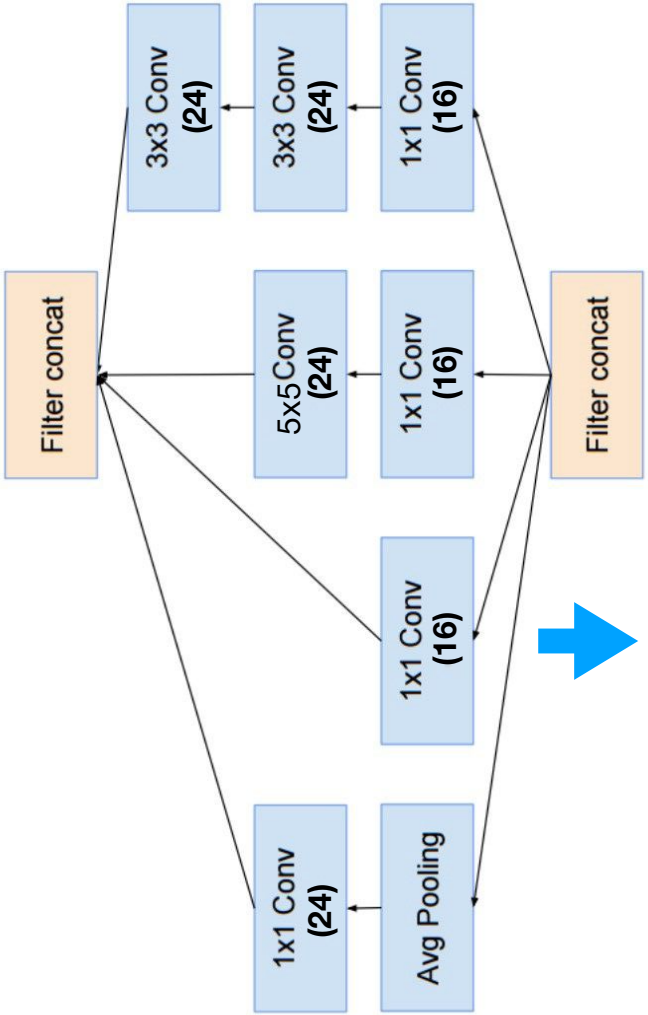
https://www.youtube.com/watch?v=VxhSouuSZDY

Inception Module

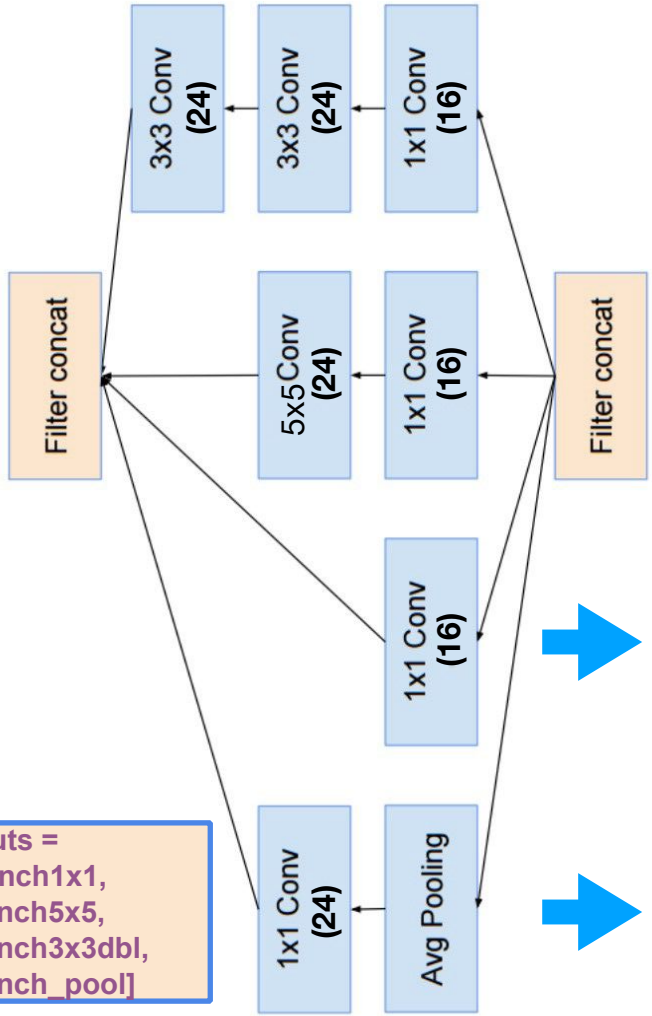


https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/


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



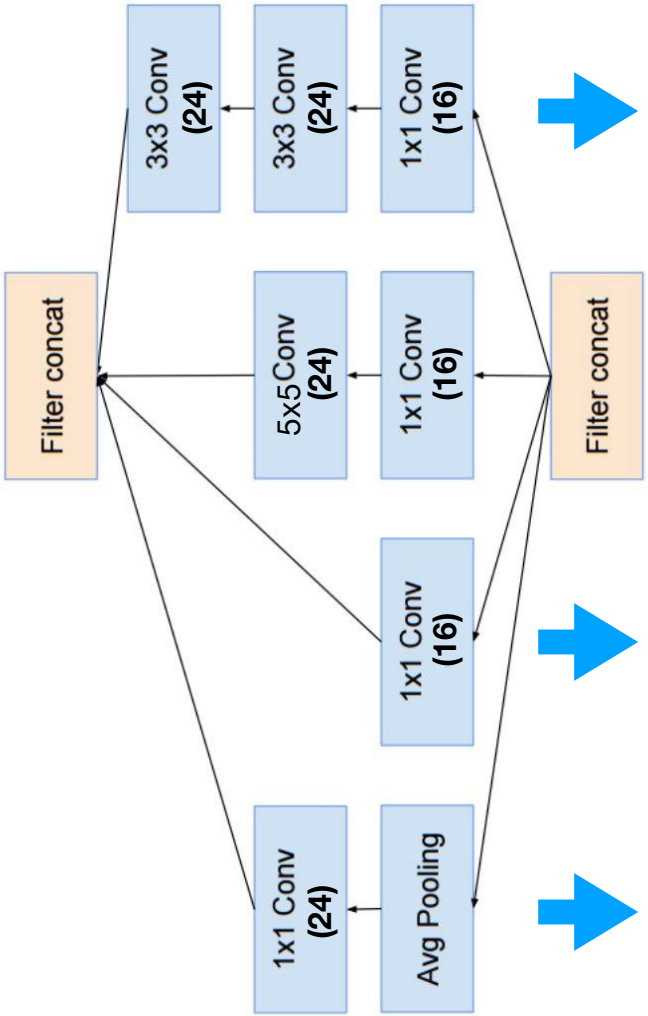

```
self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

branch1x1 = self.branch1x1(x)
```




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

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

branch1x1 = self.branch1x1(x)
```

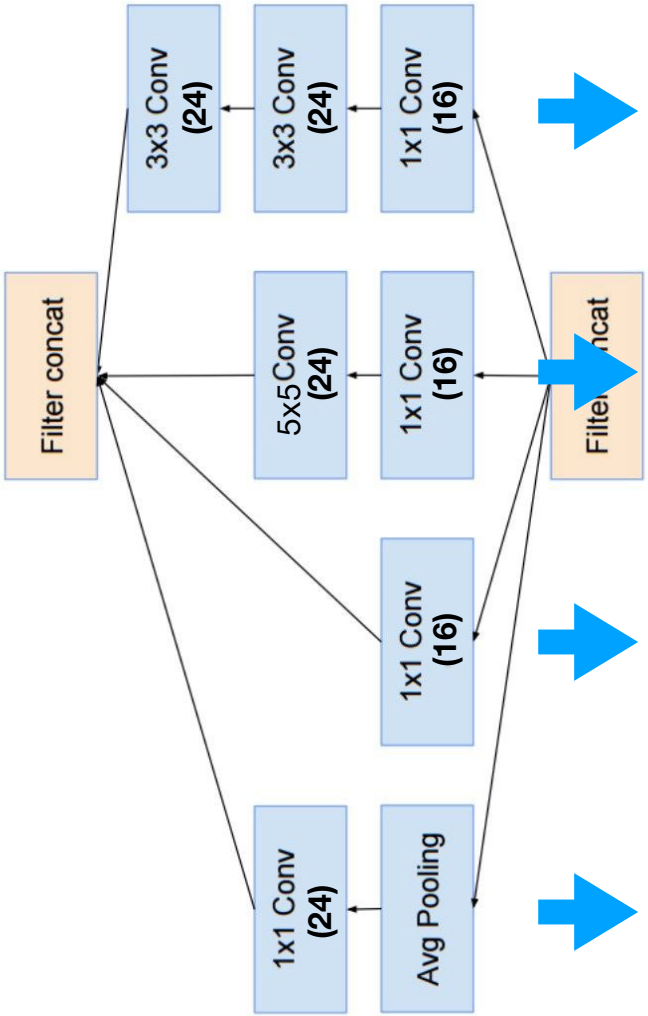
```
self.branch_pool = nn.Conv2d(in_channels, 24, kernel_size=1)

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




```
self.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)
```

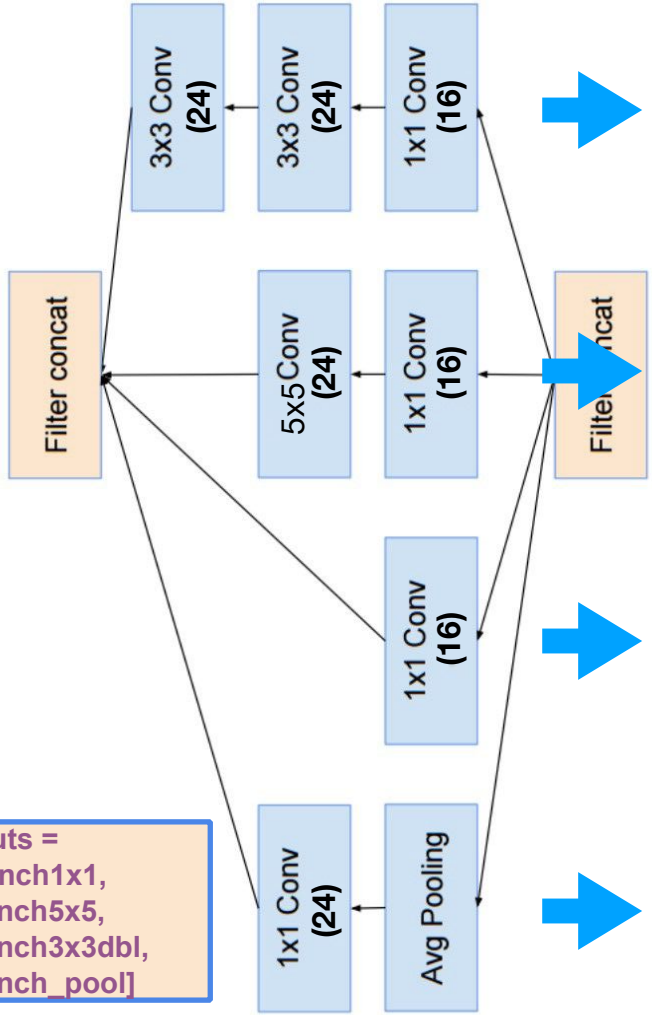



```
self.branch3x3dbl_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch3x3dbl_2 = nn.Conv2d(16, 24, kernel_size=3, padding=1)
self.branch3x3dbl_3 = nn.Conv2d(24, 24, kernel_size=3, padding=1)

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)
```




```
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)
```

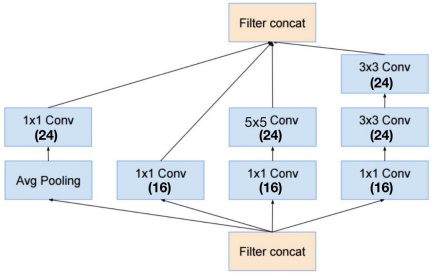
```
self.branch5x5_1 = nn.Conv2d(in_channels, 16, kernel_size=1)
self.branch5x5_2 = nn.Conv2d(16, 24, kernel_size=5, padding=2)

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)
```

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

Inception Module

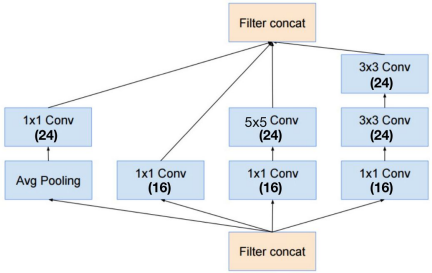



```python
class InceptionA(nn.Module):
    def __init__(self, in_channels):
        super(InceptionA, self).__init__()
        self.branch1x1 = nn.Conv2d(in_channels, 16, kernel_size=1)

        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)
```

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

```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)
```

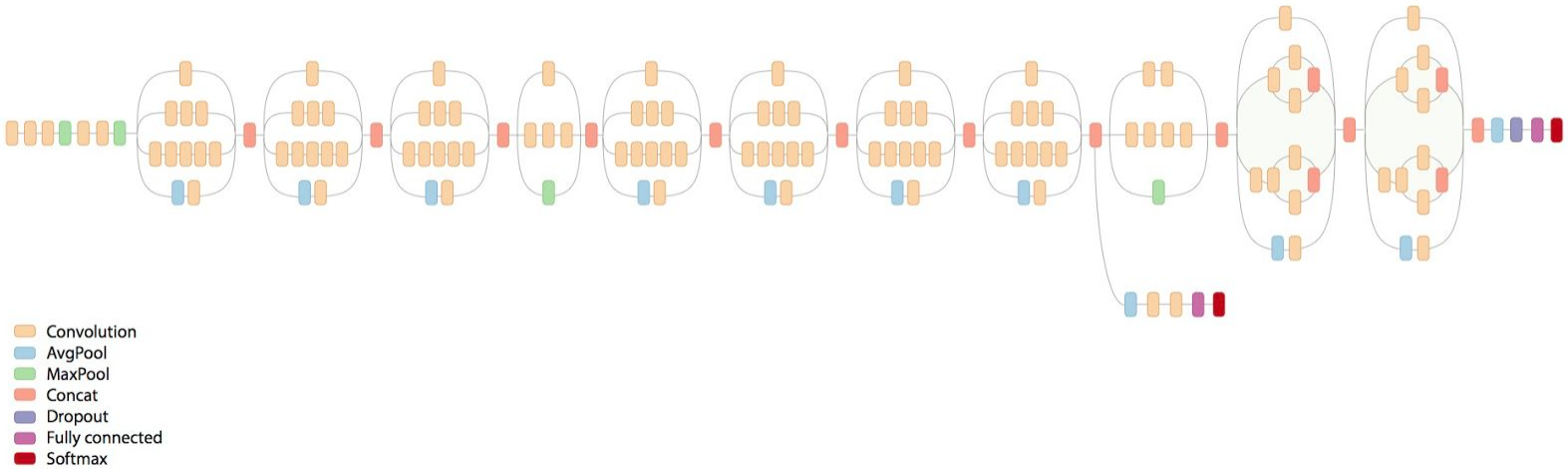
Inception Module




```
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%)
```

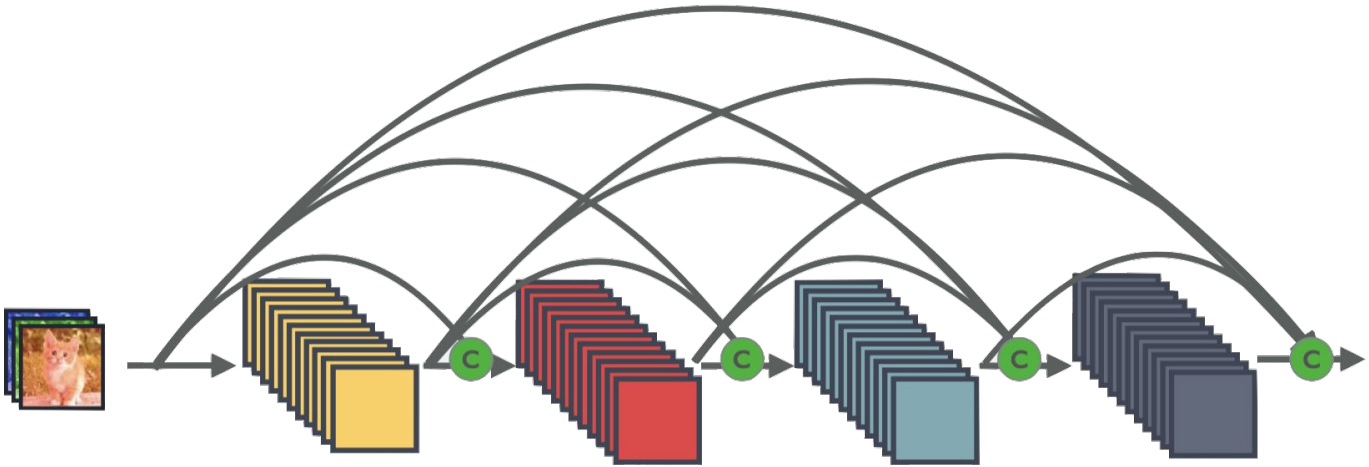
```python
class Net(nn.Module):

    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

Exercise 10-3: Implement DenseNet



https://arxiv.org/abs/1608.06993

Lecture 11: RNN