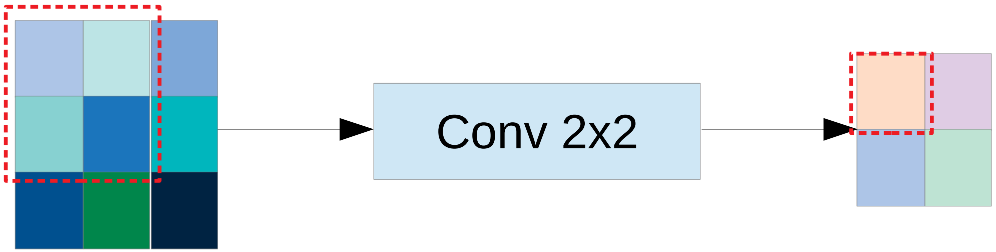
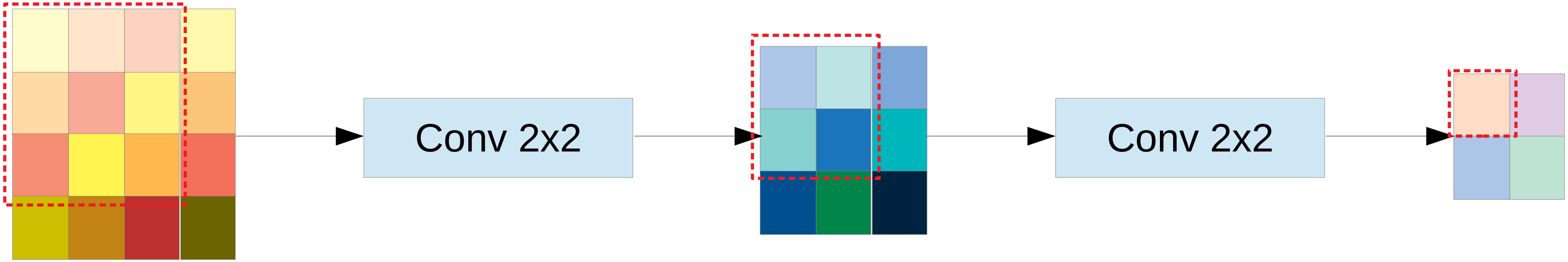
CNN Rules of Thumb

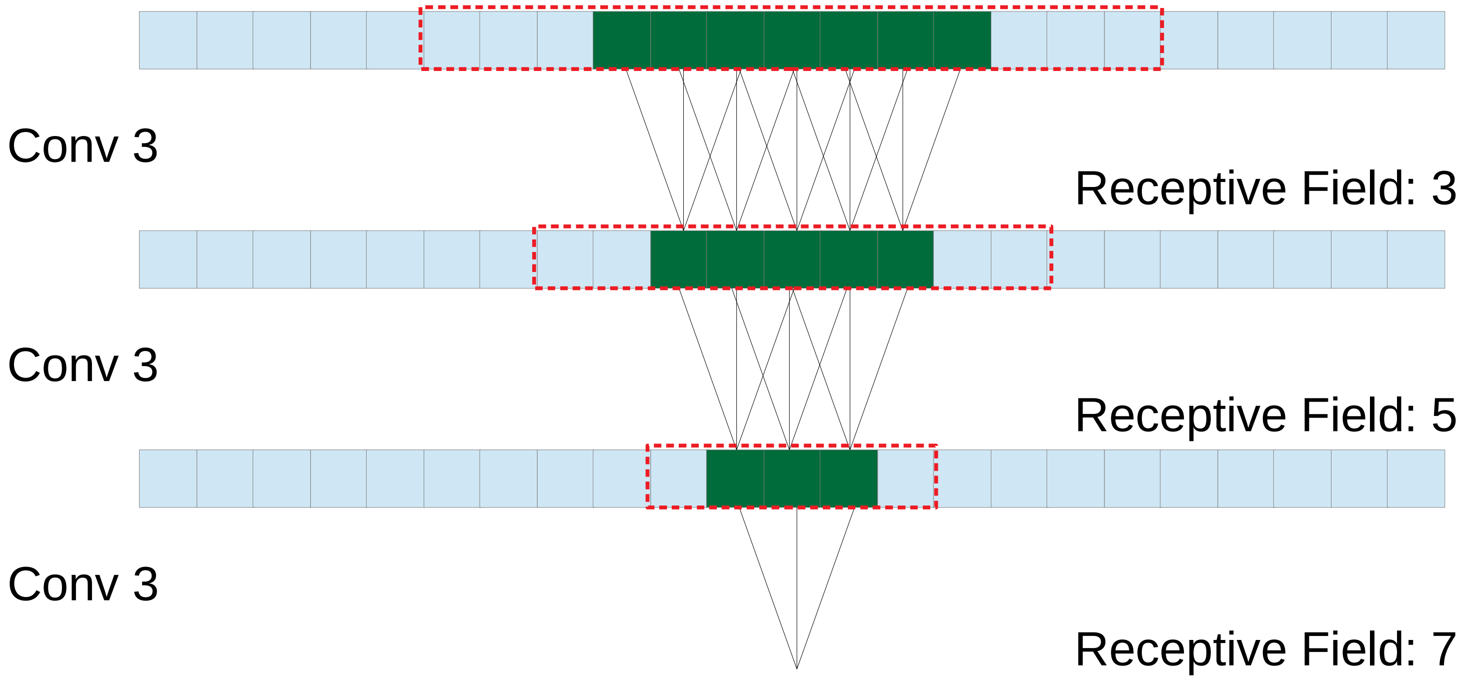
Receptive Field

Does the input X[a, b, c] affect the output Y[i, j, k]

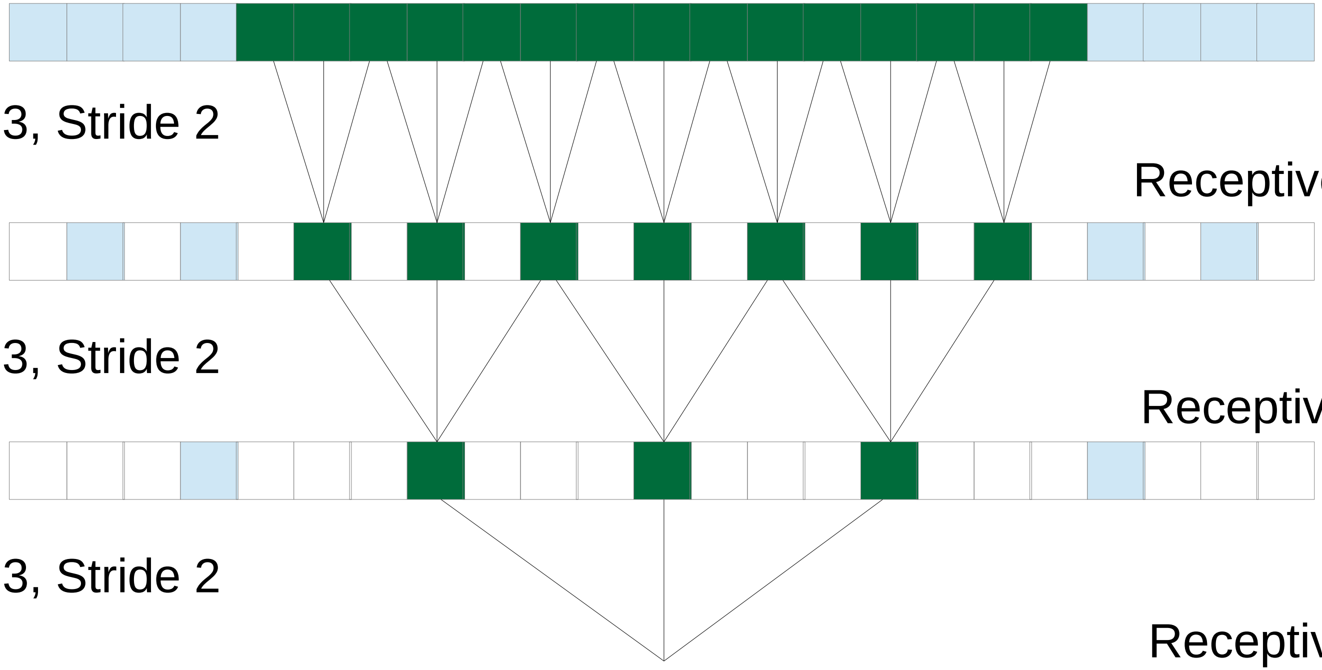




Receptive Field



Receptive Field – Striding



Conv 3, Stride 2

Receptive Field: 3

Conv 3, Stride 2

Receptive Field: 7

Conv 3, Stride 2

Receptive Field: 15

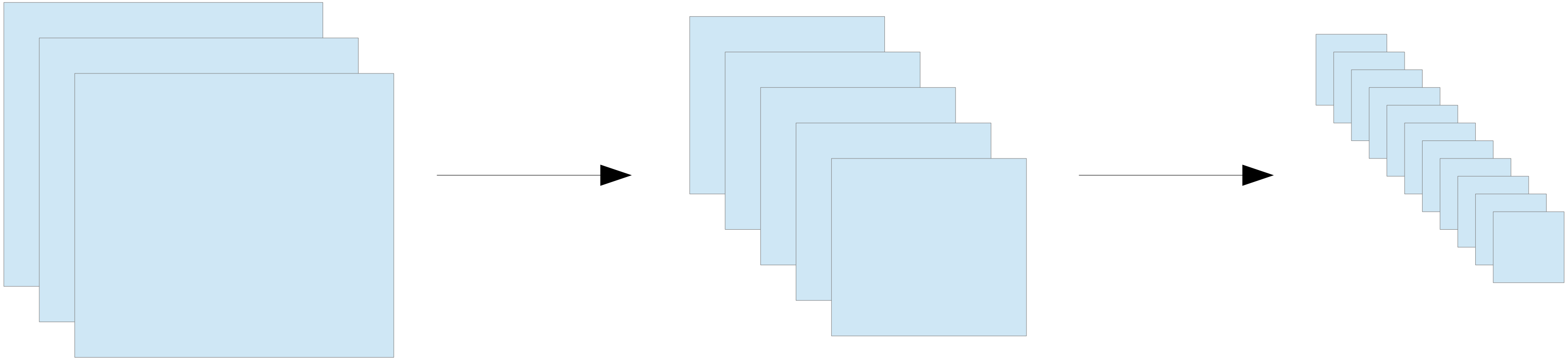
Stride and Increase Channels

Trade spatial resolution for channels

Balance computation

Stride = 2, $C_o = 2 * C_i$

Special case: first layer
$C_i = 3$, $C_o = 8, 16, 32, ...$

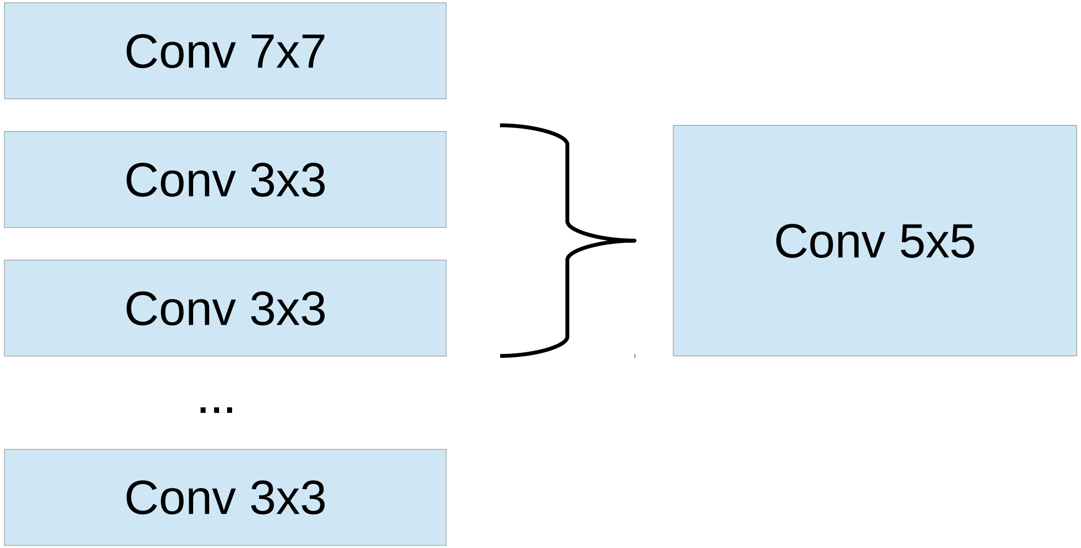


Small Kernels

- Almost always 3x3
- The first layer can be larger, up to 7x7

Two 3x3 layers:
2 * 3 * 3 = 18 parameters

One 5x5 layer:
1 * 5 * 5 = 25 parameters



Conv 7x7

Conv 3x3

Conv 3x3

...

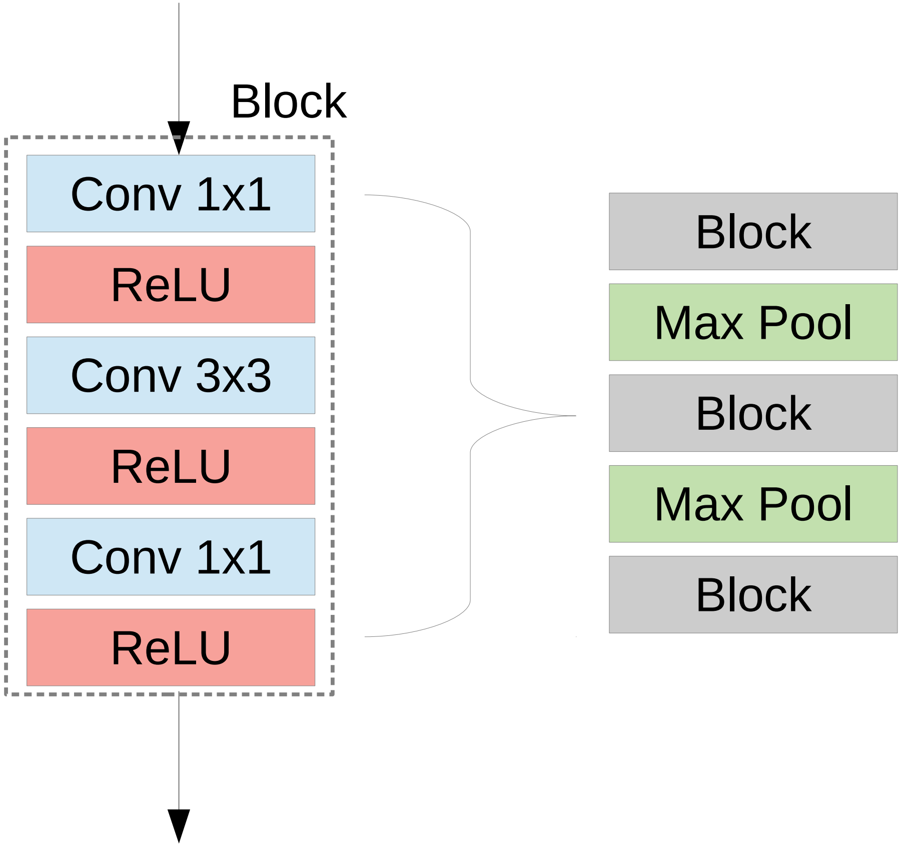
Conv 3x3

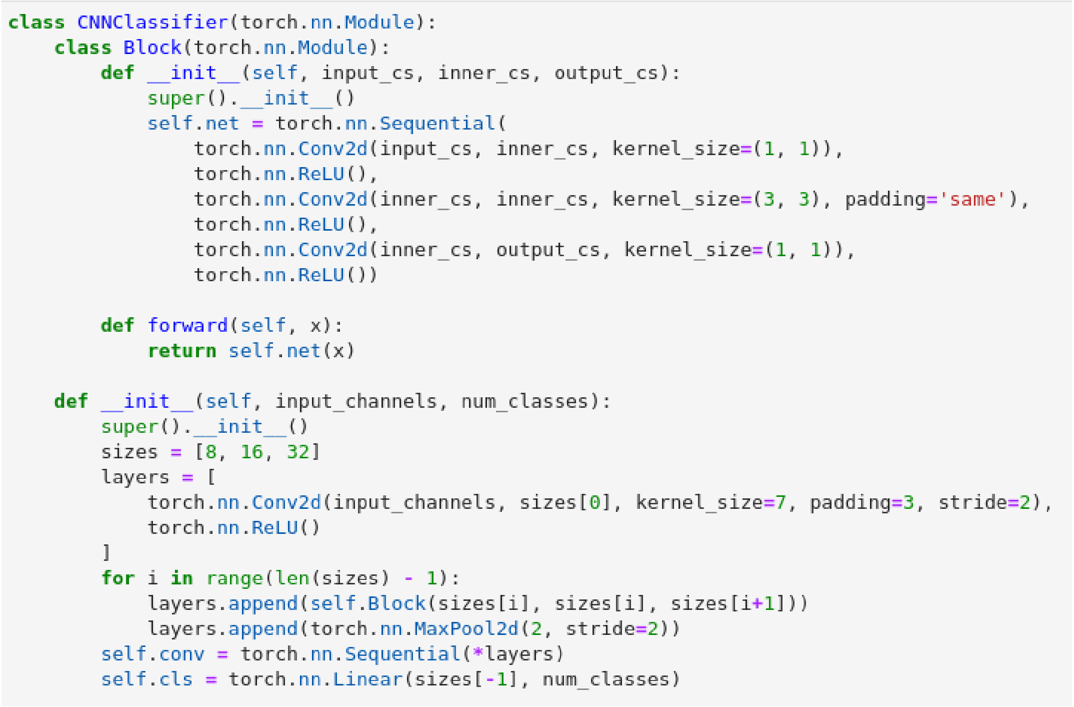
Conv 5x5

...But Not Too Small

- 2x2 layers are not centered – Padding is asymmetrical
- Two 2x2 layers is only one less parameter than a 3x3 layer.
- 1x1 kernels are a special case
  - Two 1x1 layers does not add up to a larger kernel

Repeat Patterns



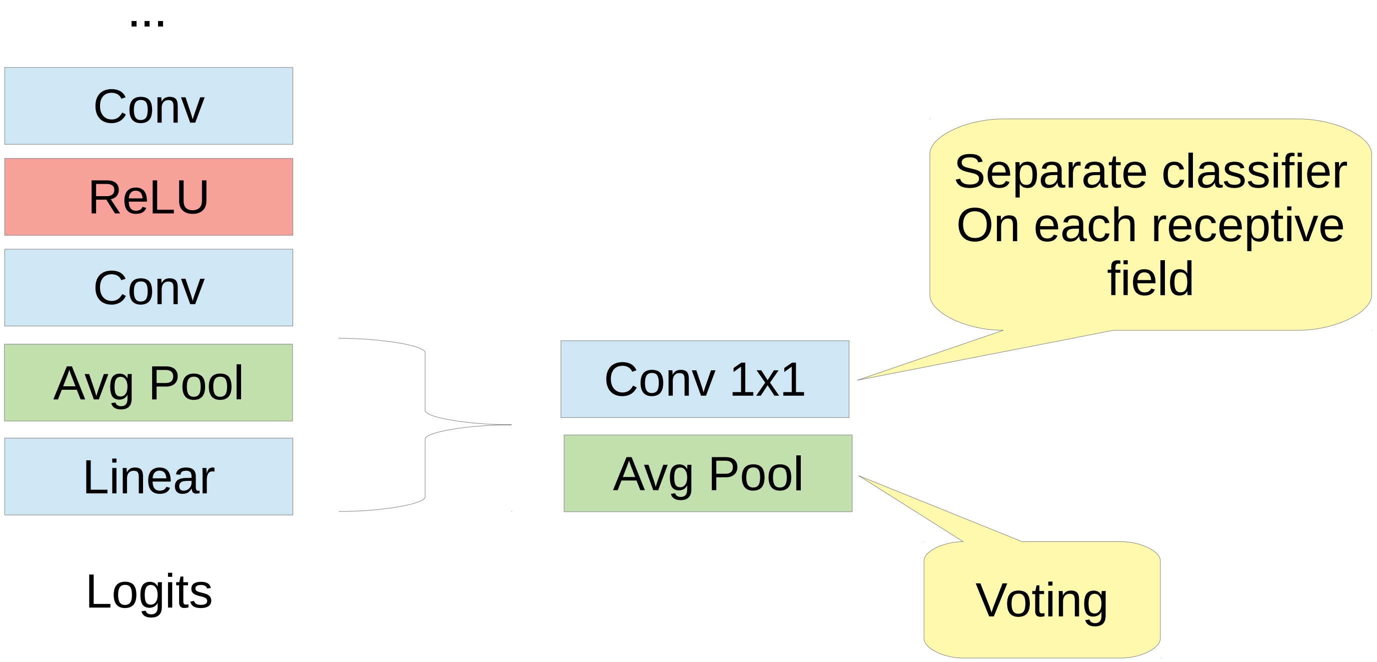


```python
class CNNClassifier(torch.nn.Module):
    class Block(torch.nn.Module):
        def __init__(self, input_cs, inner_cs, output_cs):
            super().__init__()
            self.net = torch.nn.Sequential(
                torch.nn.Conv2d(input_cs, inner_cs, kernel_size=(1, 1)),
                torch.nn.ReLU(),
                torch.nn.Conv2d(inner_cs, inner_cs, kernel_size=(3, 3), padding='same'),
                torch.nn.ReLU(),
                torch.nn.Conv2d(inner_cs, output_cs, kernel_size=(1, 1)),
                torch.nn.ReLU())

        def forward(self, x):
            return self.net(x)

    def __init__(self, input_channels, num_classes):
        super().__init__()
        sizes = [8, 16, 32]
        layers = [
            torch.nn.Conv2d(input_channels, sizes[0], kernel_size=7, padding=3, stride=2),
            torch.nn.ReLU()
        ]
        for i in range(len(sizes) - 1):
            layers.append(self.Block(sizes[i], sizes[i], sizes[i+1]))
            layers.append(torch.nn.MaxPool2d(2, stride=2))
        self.conv = torch.nn.Sequential(*layers)
        self.cls = torch.nn.Linear(sizes[-1], num_classes)
```

Only Convolutions



...
Conv
ReLU
Conv
Avg Pool
Linear
Logits
Conv 1x1
Avg Pool
Separate classifier
On each receptive
field
Voting