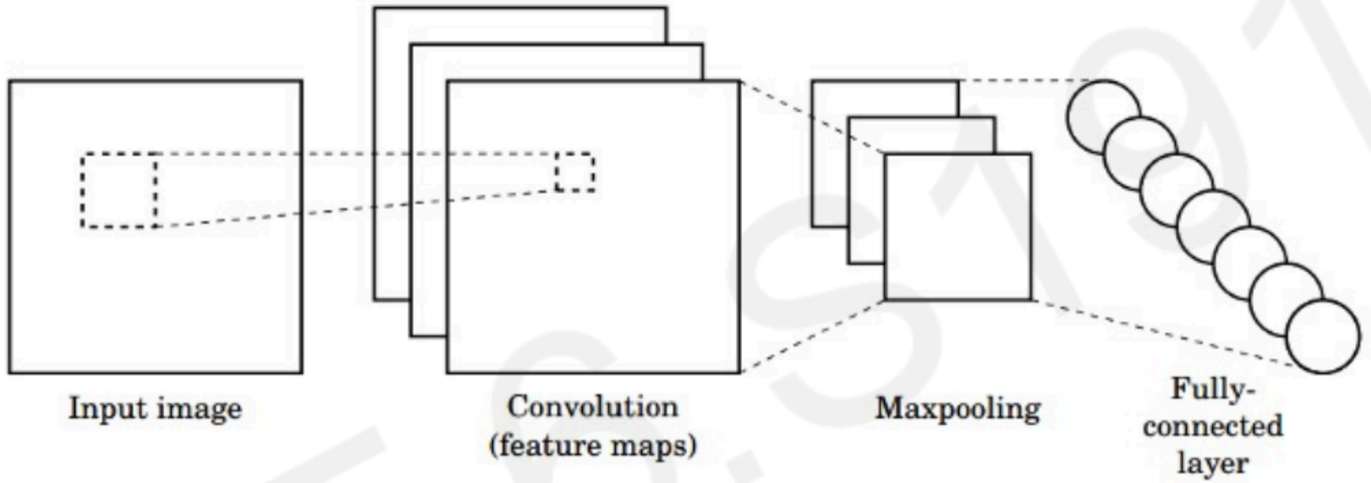
Convolutional Neural Networks (CNNs)

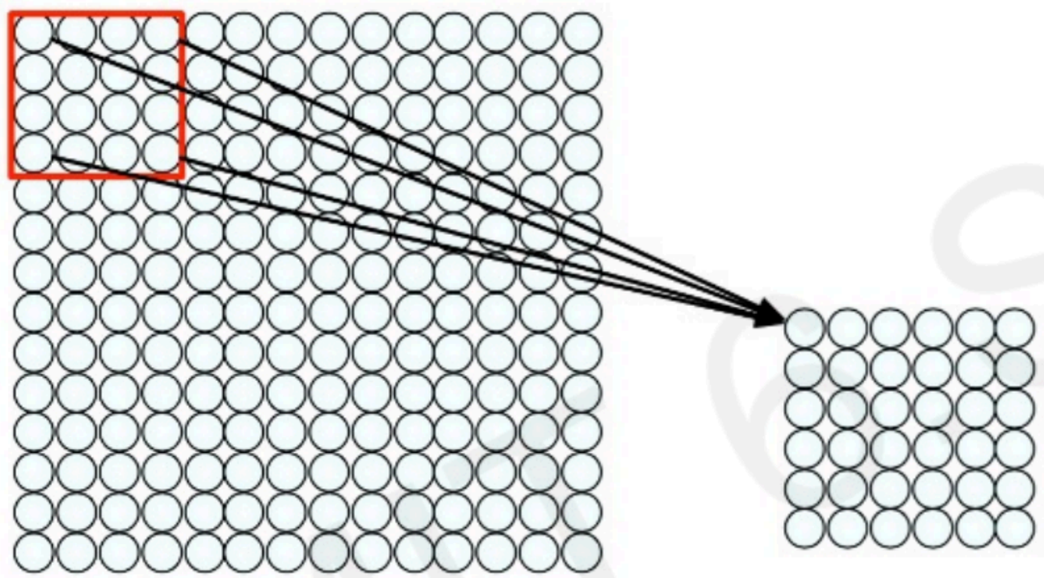
CNNs for Classification



`tf.keras.layers.Conv2D`	1. **Convolution**: Apply filters to generate feature maps.	`torch.nn.Conv2d`
`tf.keras.activations.*`	2. **Non-linearity**: Often ReLU.	`torch.nn.ReLU,…`
`tf.keras.layers.MaxPool2D`	3. **Pooling**: Downsampling operation on each feature map.	`torch.nn.MaxPool2d`

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity



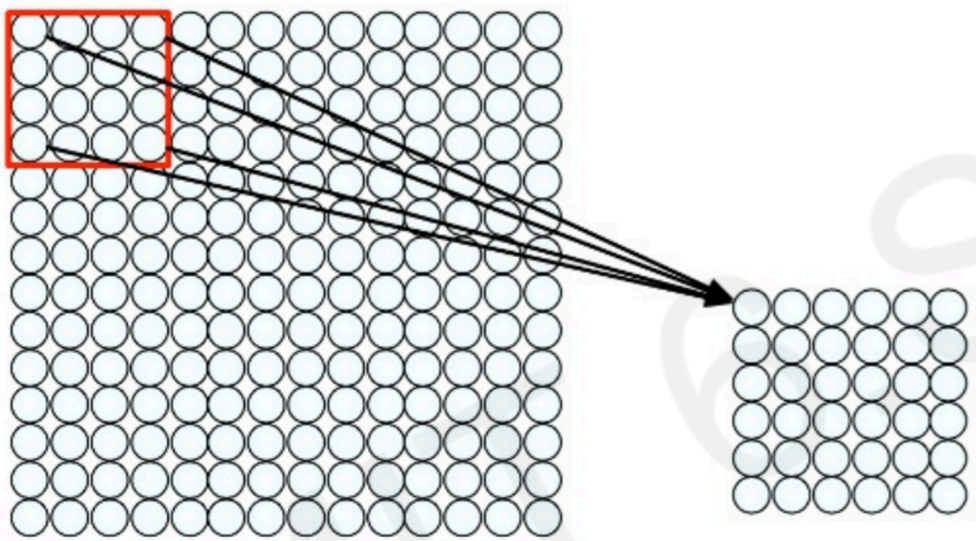
`tf.keras.layers.Conv2D`

`torch.nn.Conv2d`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity



`tf.keras.layers.Conv2D`

`torch.nn.Conv2d`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

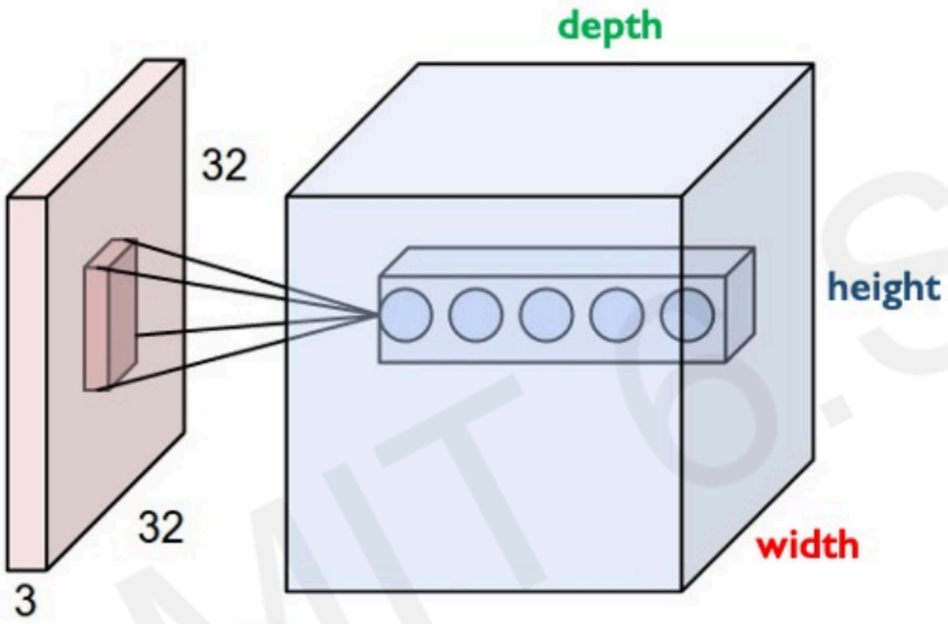
4x4 filter: matrix of weights w_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

1) applying a window of weights
2) computing linear combinations
3) activating with non-linear function

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \; x \; w \; x \; d$

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

Filter step size

Receptive Field:

Locations in input image that a node is path connected to

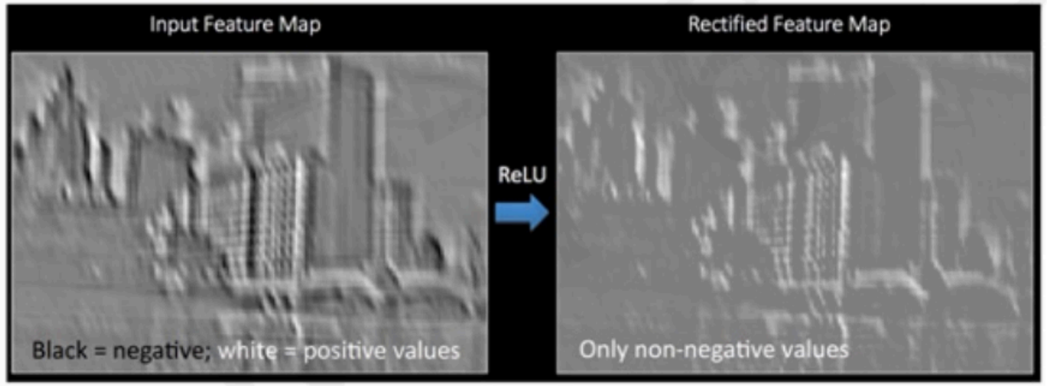
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

```
torch.nn.Conv2d( in_channels=3, out_channels=d, kernel_size=(h,w), stride=s )
```

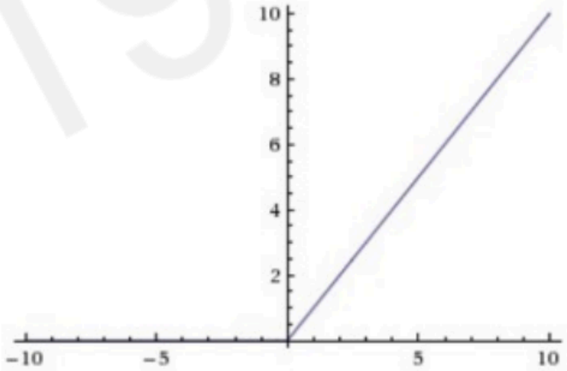
Need to specify input dimensionality!

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



Rectified Linear Unit (ReLU)

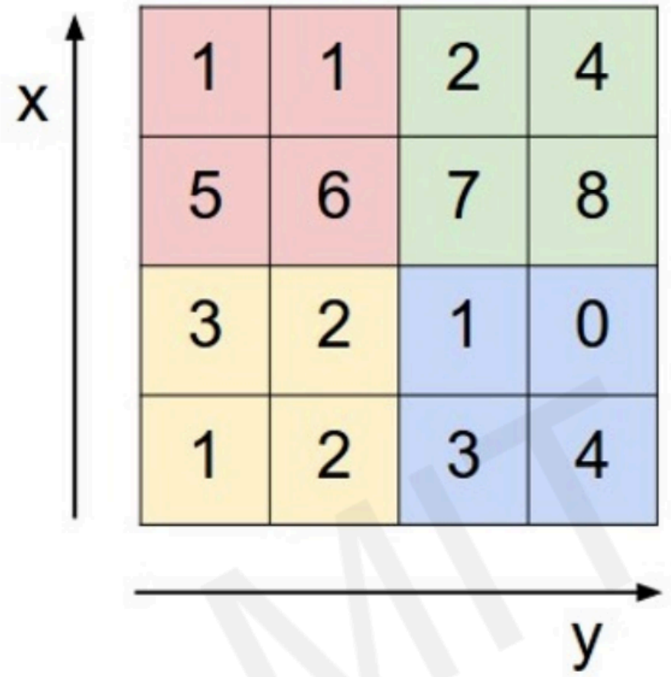


$$g(z) = \max(0, z)$$

`tf.keras.layers.ReLU`

`torch.nn.ReLU`

Pooling



max pool with 2x2 filters and stride 2

```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

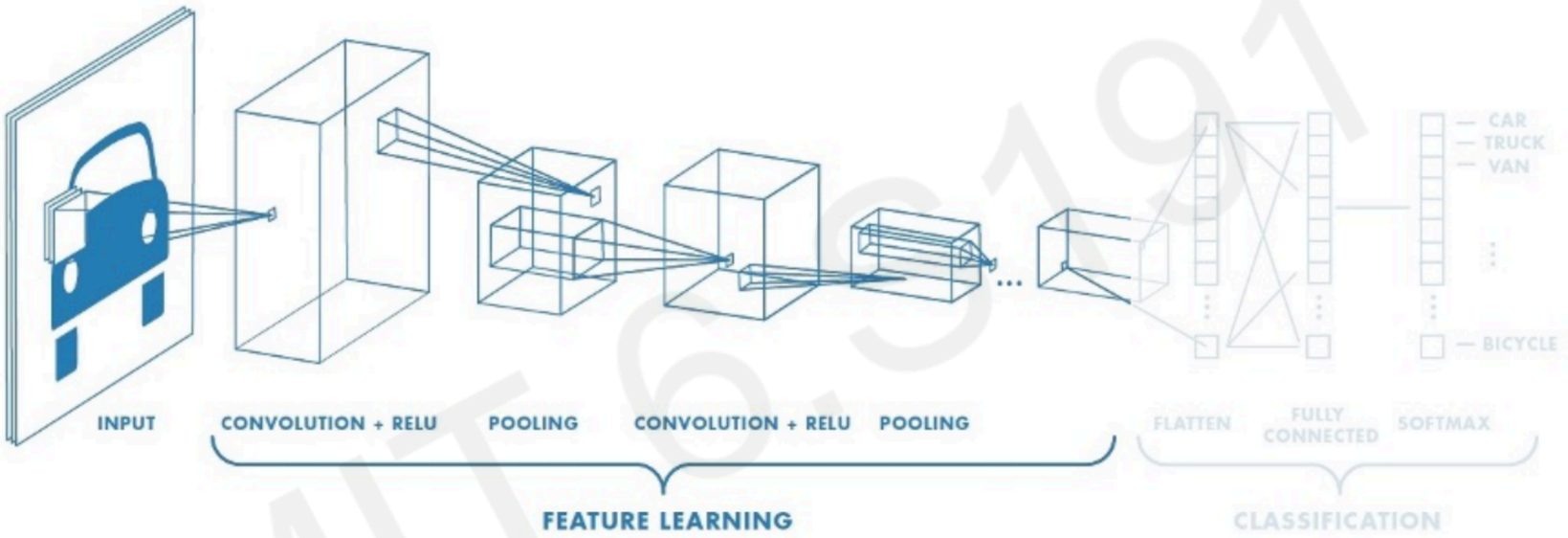
```
torch.nn.MaxPool2d(
    kernel_size=(2,2),
    stride=2
)
```

6	8
3	4

1) Reduced dimensionality
2) Spatial invariance

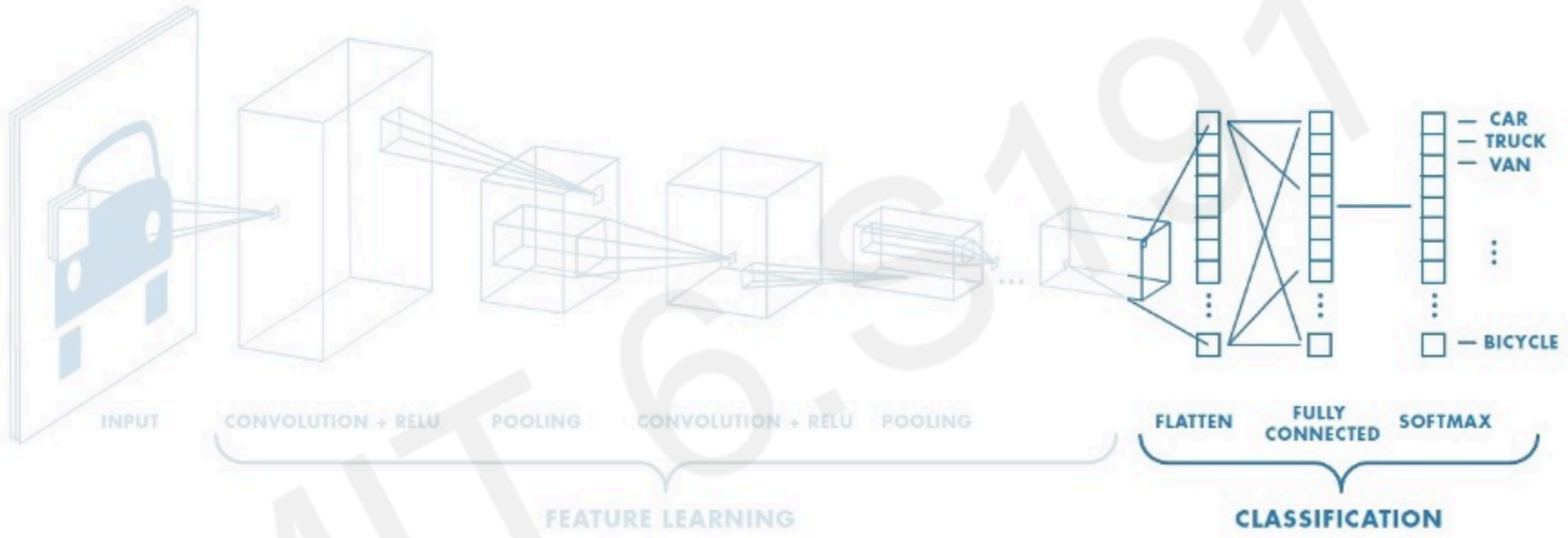
How else can we downsample and preserve spatial invariance?

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$