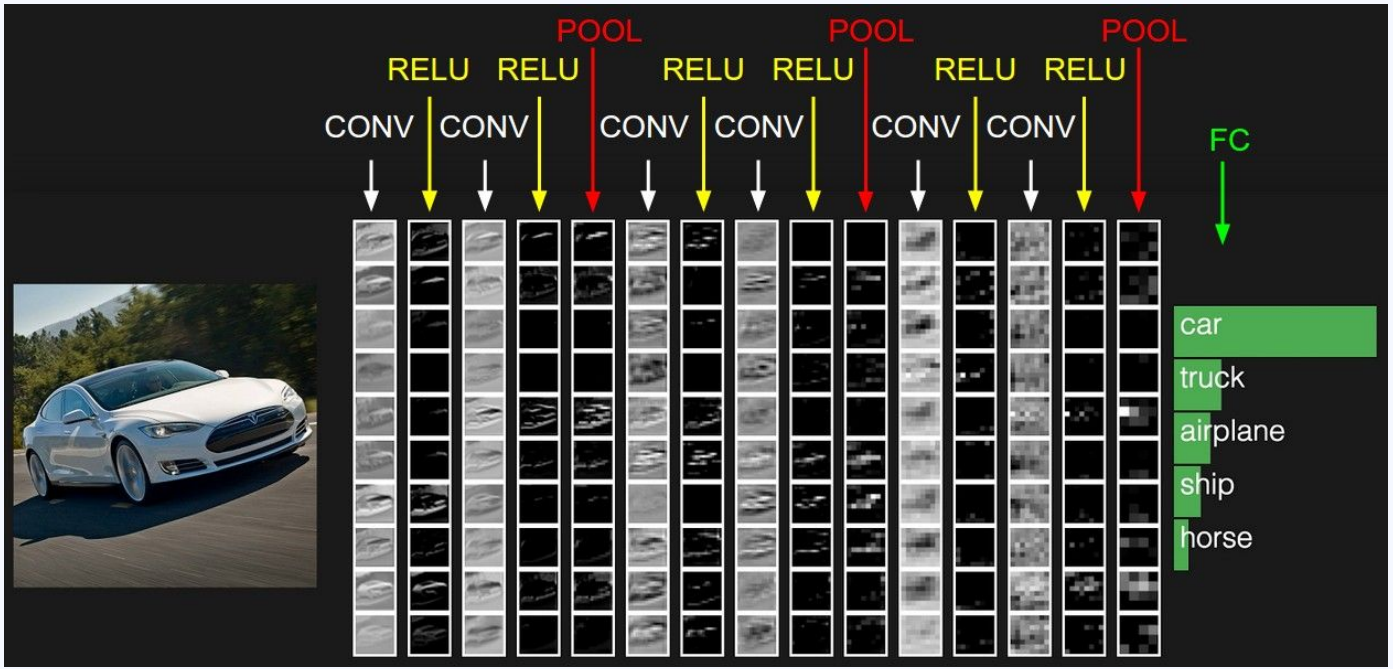
Basic Image Classification

Image Classification

- Categorize an image into one label out of multiple possible labels



Source: https://cs231n.github.io/convolutional-networks/

Image Classification Pipeline(1)

- Convert the image into a tensor

- Shape: [B, C, H, W] (B: batch size; C: channels; H: height; W: width)

Image Classification Pipeline(1)

- The tensor conversion is handled with torchvision.transforms.ToTensor()
- It also normalizes the tensor into a range from 0 to 1
- And adjusts the tensor values to normalize the mean and standard deviation of each channel (torchvision.transforms.Normalize)

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

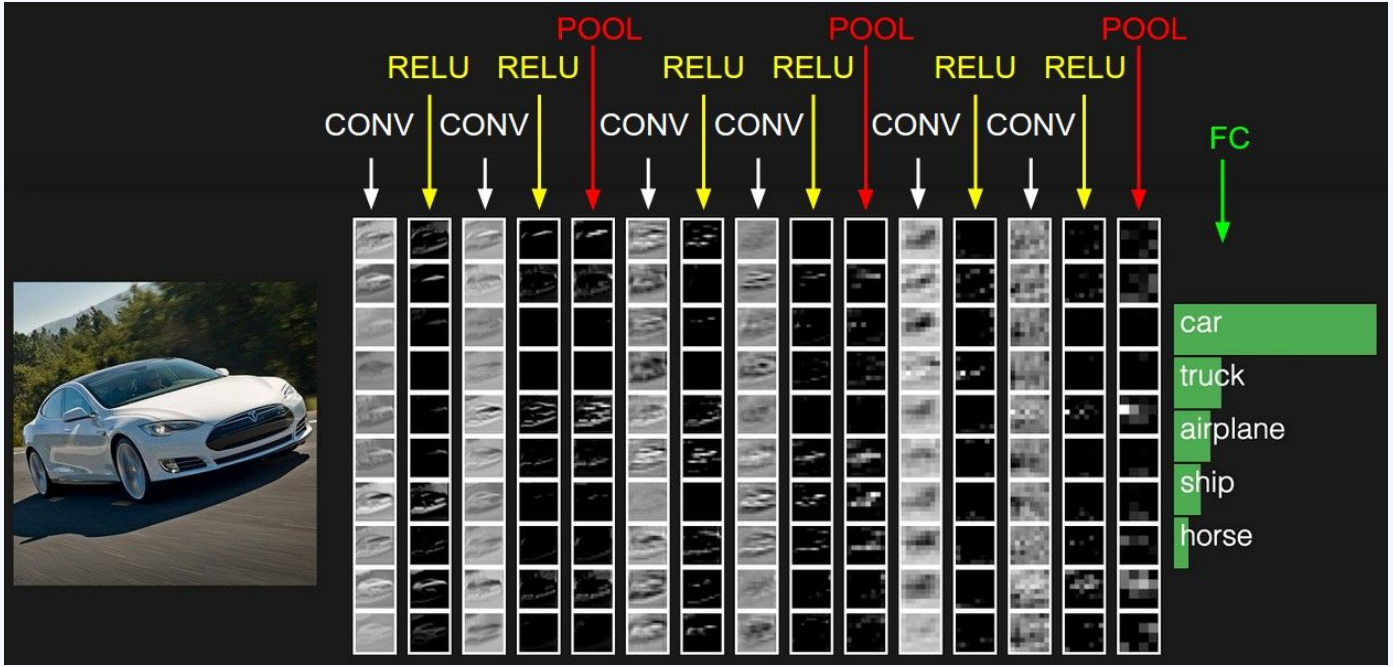
Image Classification Pipeline(2)

- The batch dimension is responsible for stacking multiple images into a batch
- The images are fed into the model one batch at a time during training
- It is handled by torch.utils.data.DataLoader

```
batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

Convolutional Neural Networks (CNNs)

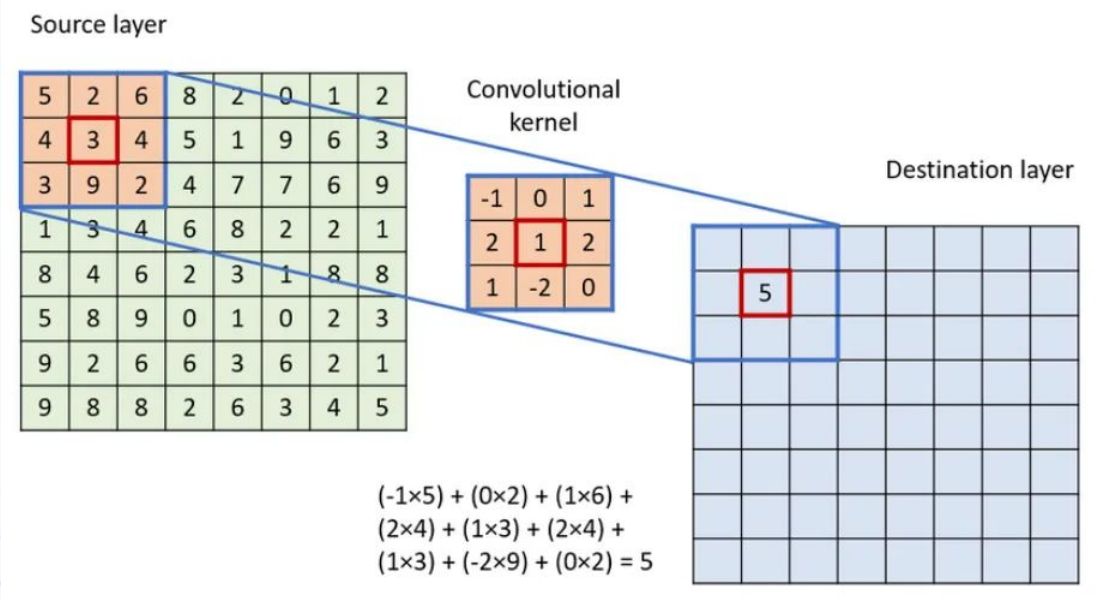
- Follows: Convolution layer—>activation function—>pooling layer—>fully connected layers



Source: https://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks (CNNs)

- One of the mainstream ways for deep learning systems to process images is through convolutional neural networks



Convolutional Neural Networks: Kernel

- "Sliding window" approach: a kernel is "slid" through the image
- Each time it slides into a new set of corresponding pixels in the original image, it computes a cross-correlation of the kernel and the corresponding pixels and attaches it onto the output image

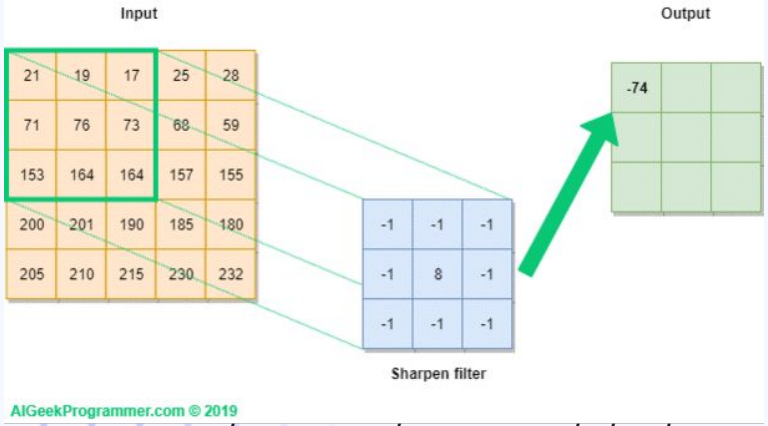
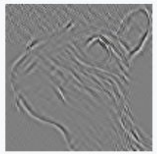
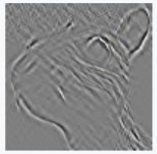


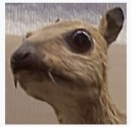
Image source: https://upload.wikimedia.org/wikipedia/commons/9/90/CNN-filter-animation-1.gif

Convolutional Neural Networks: Kernel

- Convolutional kernels extract features from the image

Ridge or **edge detection**	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

cessi

Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Convolutional Neural Networks: Kernel

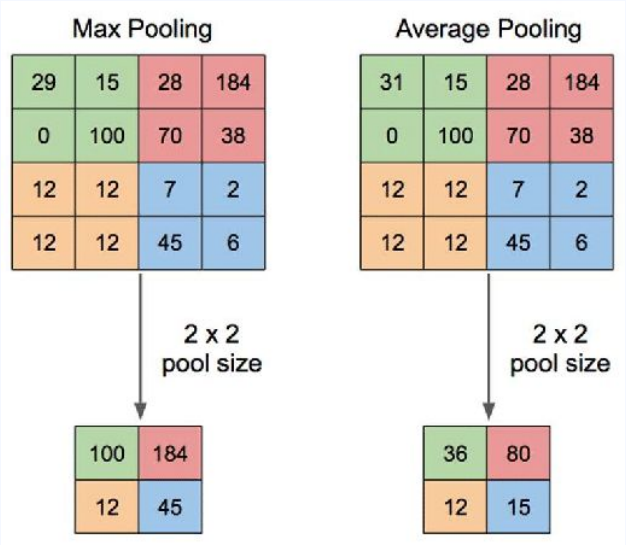
- Earlier convolution layers tend to extract lower-level local features such as edges and corners
- Later convolution layers tend to extract higher-level global features, gathering information regarding the shape of the objects in the image

Convolutional Neural Networks: Kernel

- The convolutional kernels are the learnable weights in convolutional layers
- Therefore the purposes of the convolution kernels are not as clear cut, they are all learned from data!

Convolutional Neural Networks: Pooling

- Another key component of CNNs
- Most common forms: max pooling and average pooling



Source: https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

Convolutional Neural Networks:Pooling

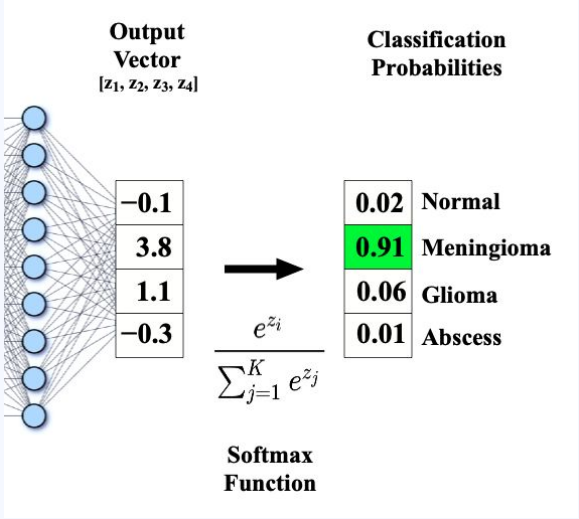
- To reduce the image dimension while preserving semantically useful information
- Eventually the image size is small enough that the convolutional kernel is of a significant size, and global features can be extracted more easily

Fully Connected Layer

- After the feature maps are extracted, the image is flattened into one vector (for each image)
- Shape after flattening: [B, final_feature_map.numel()] # B: batch size
- The resulting vector can be passed into models of choice (classification models or nn.Linear layers)

Fully Connected Layer

- After flattening, apply a linear layer or multilayer perceptron to the flattened feature maps to produce logits for each class
- Softmax is applied to the logits to make it a proper probability distribution over the classes

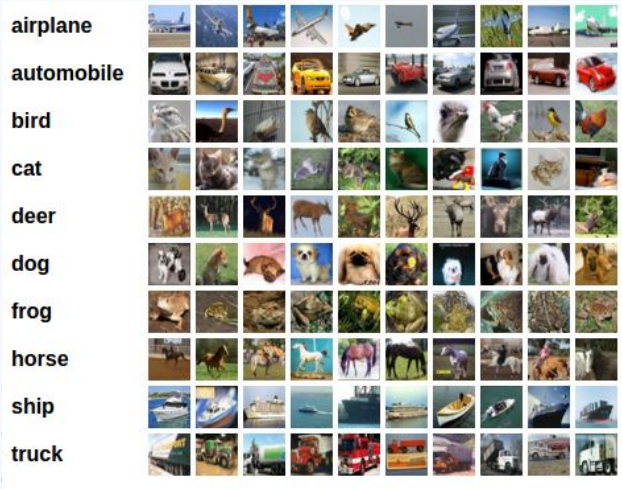


Baseline Architecture

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5) # convolutional layer
        self.pool = nn.MaxPool2d(2, 2) # pooling
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5) # convolutional layer
        self.fc1 = nn.Linear(16 * 5 * 5, 120) # fully connected layer 1
        self.fc2 = nn.Linear(120, 84) # fully connected layer 2
        self.fc3 = nn.Linear(84, 10) # fully connected layer 3

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
net = net.to(device)
```

Dataset

- CIFAR-10
- A benchmark dataset for evaluating model performance
- 60000 32x32 images in 10 classes of 6000 each
- 50000 images (5000 per class) on the train set, 10000 images (1000 per class) on the test set



Basic Image Classification

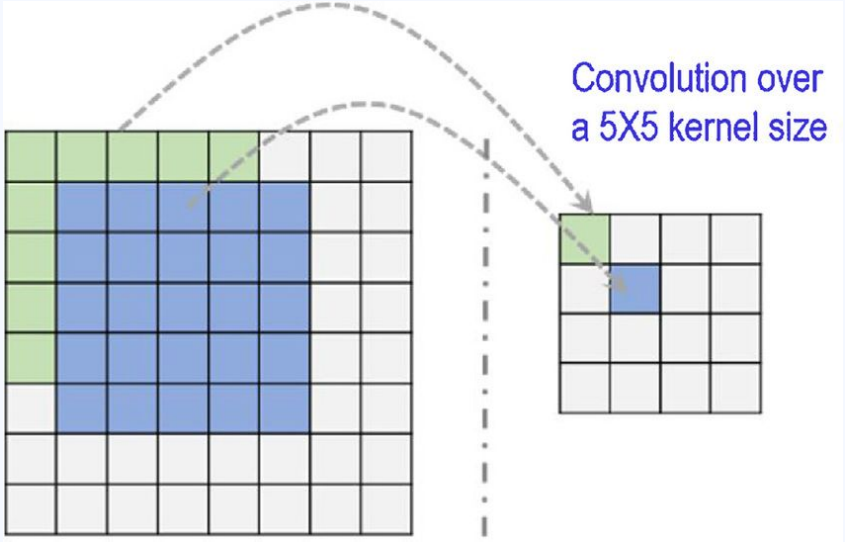
- https://colab.research.google.com/drive/1hCW7c_2KqvbdTkHqlidq7hGBfdnqb3LX
- https://www.kaggle.com/code/carsoncheng/cifar10-classification
- Run through the image classification pipeline and try to improve the accuracy!

Calculating Image Shapes

- It’s important to calculate image shapes through the convolution layers when dealing with convolutional neural networks
- To make sure your network can receive an image of your required size and to make sure you can extract both local and global features

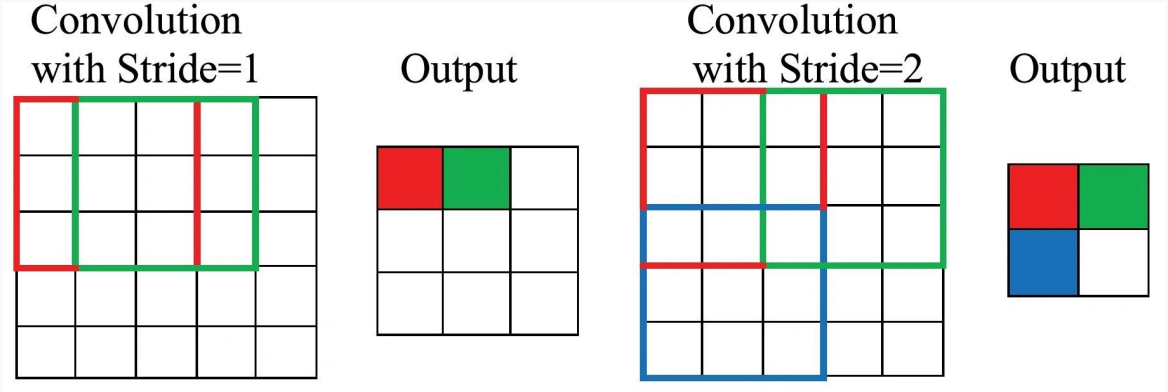
Convolutional Layer Parameters

- kernel_size: the size of the kernel slid through the image



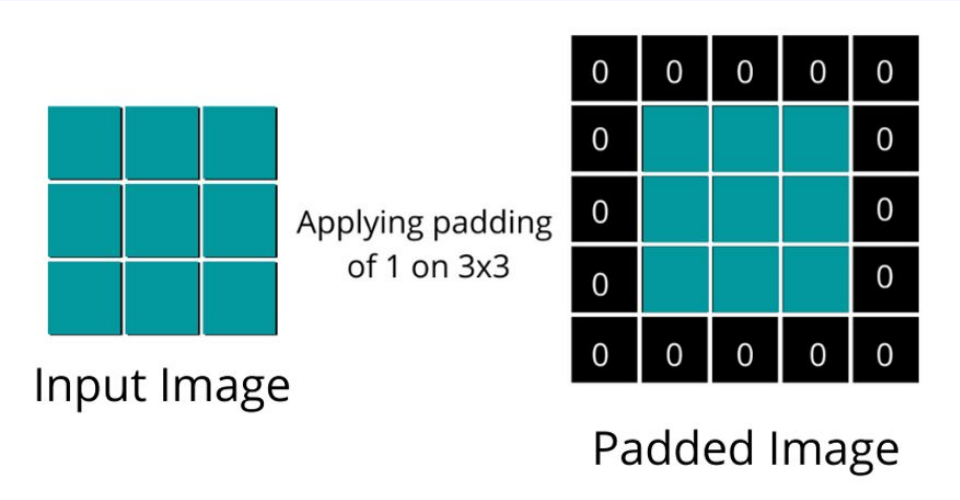
Convolutional Layer Parameters

- stride: the step size of sliding the kernel through the image



Convolutional Layer Parameters

- padding: by how much is the image “padded” (usually with zeros) to increase the resultant image size to the required size



Calculating Image Shapes

- If padding="same", the output image shape is the same as the input image shape
- Otherwise, for each layer, use the formula:

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Calculating Image Shapes

- Example: if an image has shape [1, 3, 32, 32], if it passes through a Conv2d layer with in_channels=3, out_channels=3, kernel_size=(3, 3), stride=1, padding=0, dilation=1, what is the output image size?
- Final shape = [B, C, H, W]
- Answer: B = original batch size = 1
- C = out_channels = 3
- H (height): (32 + 2 * 0 - 1 * (3 - 1) - 1) / 1 + 1 = 30
- W (width): (32 + 2 * 0 - 1 * (3 - 1) - 1) / 1 + 1 = 30
- Therefore, the final shape is [1, 3, 30, 30]

Calculating Image Shapes

- Can you pass a 5x5 convolution kernel on a 4x4 image?
- (yes / no) (remove the incorrect option)