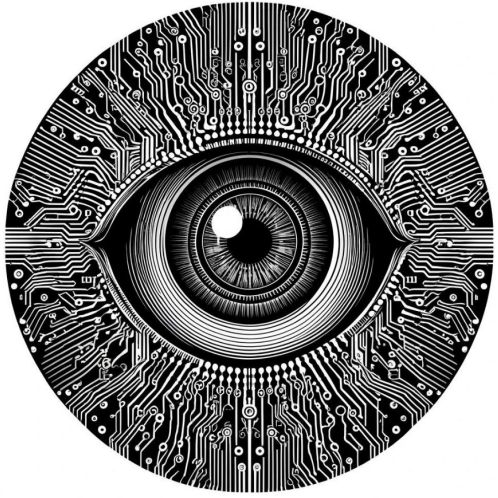
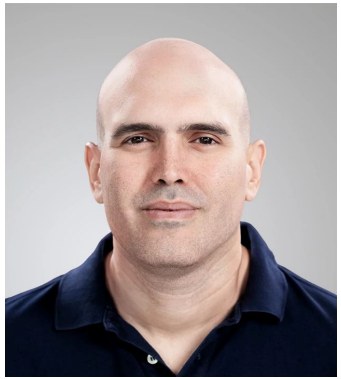
Workshop 4 - Introduction to Convolutional Neural Networks



Antonio Rueda-Toicen

About me

- AI Researcher at Hasso Plattner Institute, AI Engineer & DevRel for Voxel51
- Organizer of the Berlin Computer Vision Group
- Instructor at Nvidia's Deep Learning Institute and Berlin's Data Science Retreat
- Preparing a MOOC for OpenHPI (to be published in May)



LinkedIn

Agenda

- Fundamentals of convolutions
- Pooling in neural networks

Notebooks

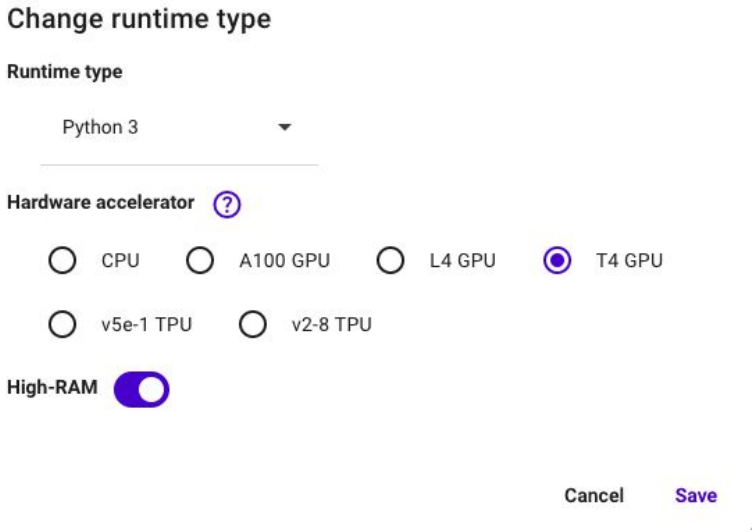
- Digit recognition on the MNIST dataset with the LeNet5 model
  - Kaggle competition notebook - GitHub link
- Visualizing convolutions and pooling in LeNet5
  - Kaggle notebook - GitHub link

Setup for today

- Setting up a Weights and Biases API token
- Setting up free GPU and TPU access in Kaggle Notebooks

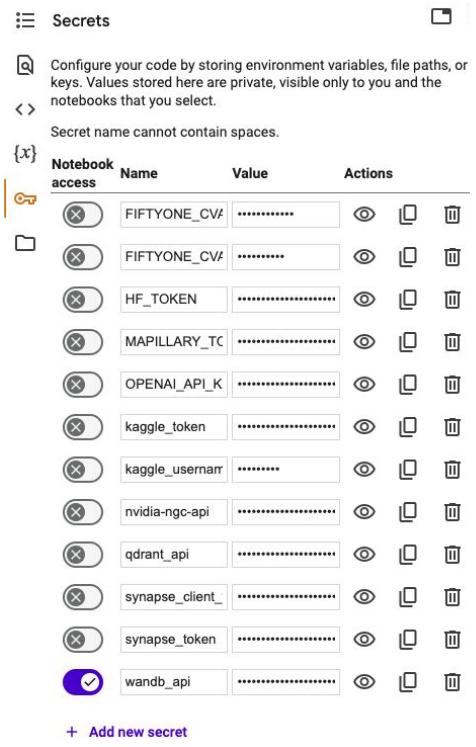
Google Colab Runtime Setup for Training

- Runtime -> Change Runtime type

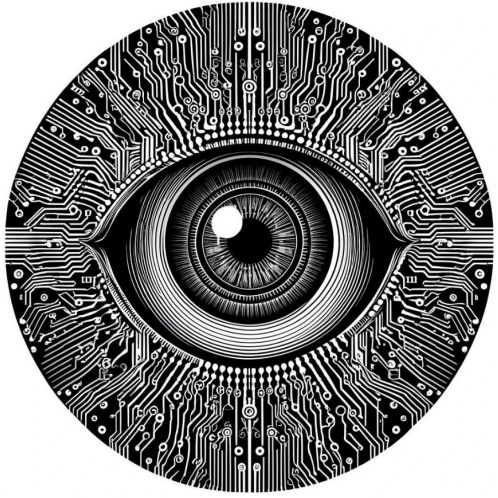


Google Colab Setup for wandb API keys

- Go to wandb.ai/authorize to find your API key
- Go to Secrets -> “Add New Secret”
  in your Colab environment



Classifying Handwritten Digits



The problem: recognizing zip codes

Throwback to 1989

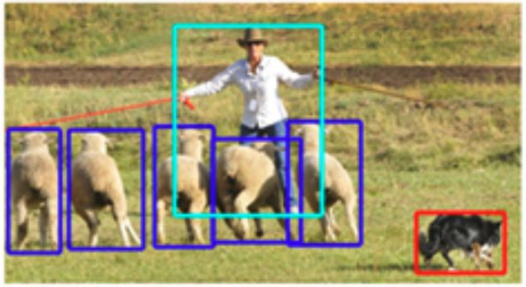
UNITED STATES POSTAL SERVICE

📫

LeNet’s history on Wikipedia

Localized vs Non-localized Image Classification

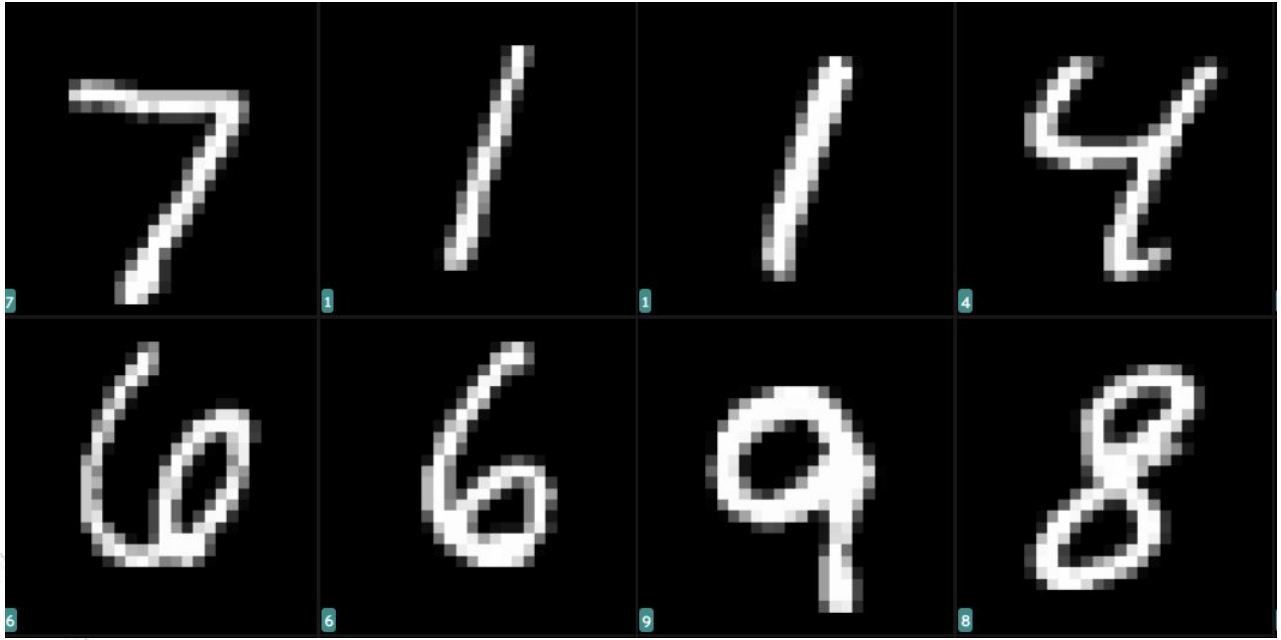
Classification



Object Detection

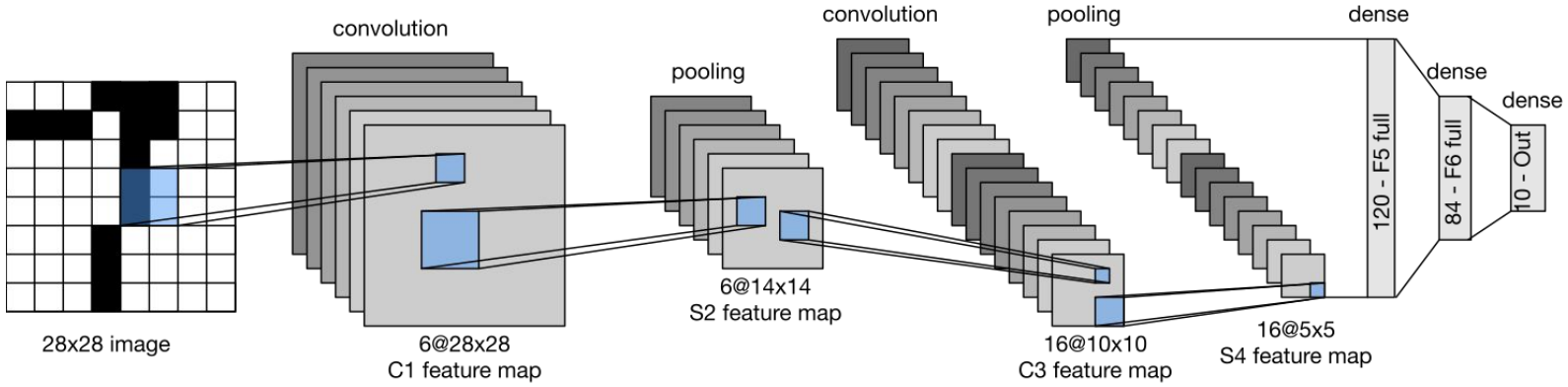
Instance Segmentation

Classifying handwritten digits



Explore the MNIST dataset in https://try.fiftyone.ai/datasets/mnist/samples

The LeNet-5 architecture



Kaggle’s digit recognition challenge

KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING

Submit Prediction

Digit Recognizer

Learn computer vision fundamentals with the famous MNIST data

https://www.kaggle.com/c/digit-recognizer

The “top” leaderboard 🤔

Digit Recognizer

Submit Prediction ⋯

Overview | Data | Code | Models | Discussion | Leaderboard | Rules | Team | Submissions

#	Team	Members	Score	Entries	Last
1	Gurleenk2		1.00000	3	2mo
2	Jayaraju Gorle		1.00000	6	2mo
3	Dhwani Goyal		1.00000	2	2mo
4	dkaoki		1.00000	1	2mo
5	JAY RATHOD		1.00000	27	2mo
6	Muhammad Ehsan		1.00000	4	1mo

https://www.kaggle.com/c/digit-recognizer/leaderboard

Labeling errors

MNIST given label:
6

Cleanlab guessed: **4**

MTurk consensus: **Neither 6 nor 4**

ID: 3520

MNIST given label:
9

Cleanlab guessed: **4**

MTurk consensus: **4**

ID: 1901

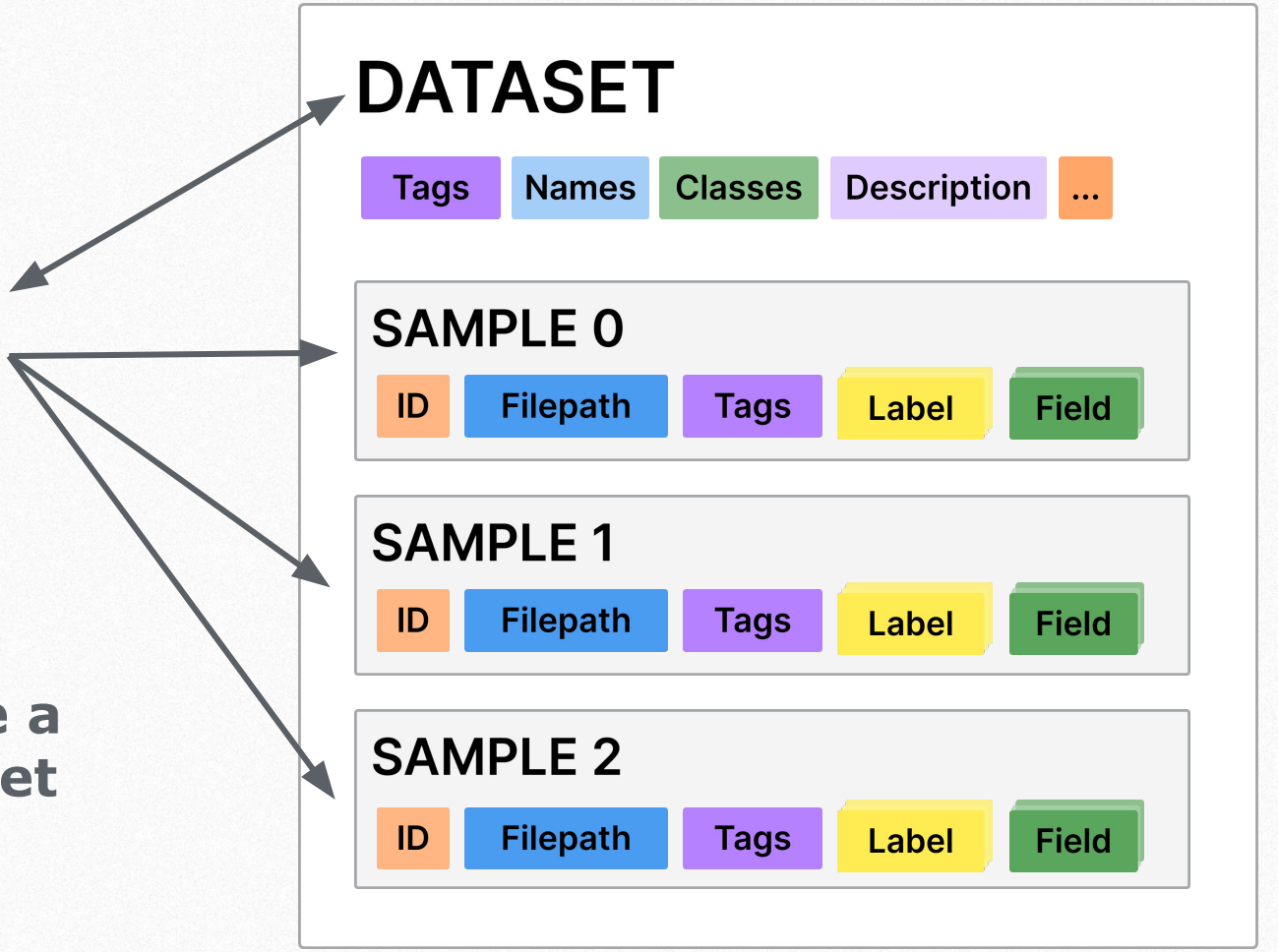
MNIST given label:
6

Cleanlab guessed: **1**

MTurk consensus: **Neither 6 nor 1**

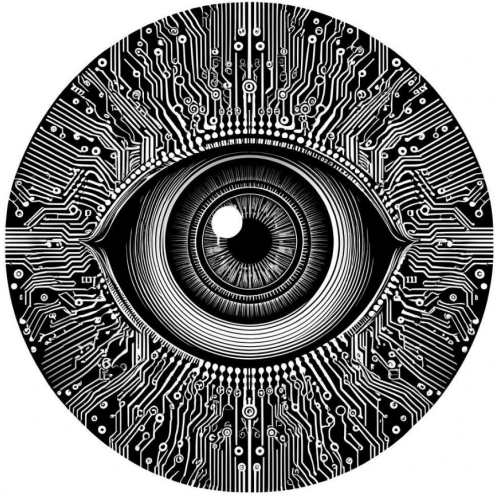
ID: 2654

https://labelerrors.com/



How we create a FiftyOne dataset

Fundamentals of Convolutions



Learning goals

- Understand the role of convolutions in producing image features
- Evaluate effect of stride, padding, and filter size on output images

Convolution filters create new images

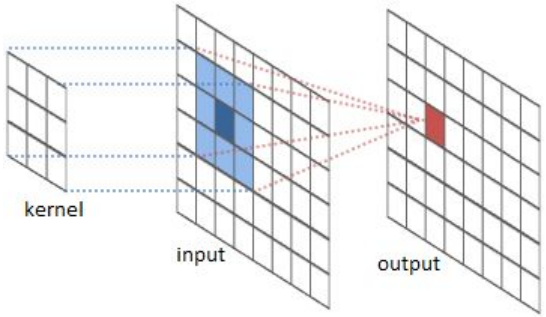
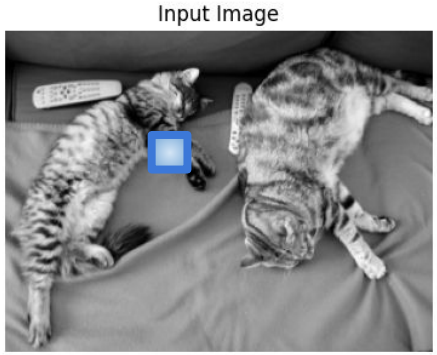
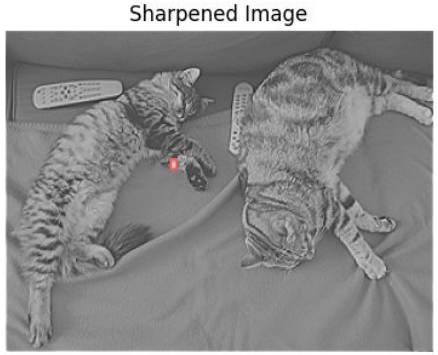


Image from source





$$\text{Sharpening Kernel} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \text{Image Patch (Input)} = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

$$\text{Output} = (0 \cdot 10) + (-1 \cdot 20) + (0 \cdot 30) + (-1 \cdot 40) + (5 \cdot 50) + (-1 \cdot 60) + (0 \cdot 70) + (-1 \cdot 80) + (0 \cdot 90)$$

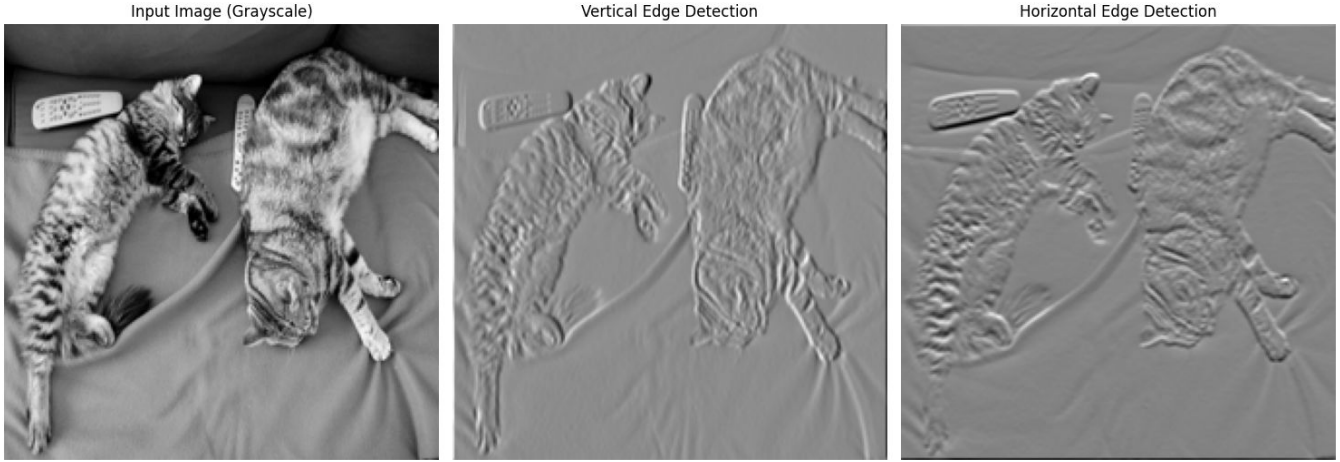
$$\text{Output} = 50$$

Convolutions were ‘traditionally’ handcrafted

$$\begin{bmatrix} 0.111 & 0.222 & 0.111 \\ 0.222 & 0.444 & 0.222 \\ 0.111 & 0.222 & 0.111 \end{bmatrix}$$

Gaussian blur kernel

Handcrafted convolution kernels



Vertical edge detection:

$$K_v = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal edge detection:

$$K_h = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The receptive field size controls how many neighboring pixels are considered

Input Image

Blurred Image (3x3)

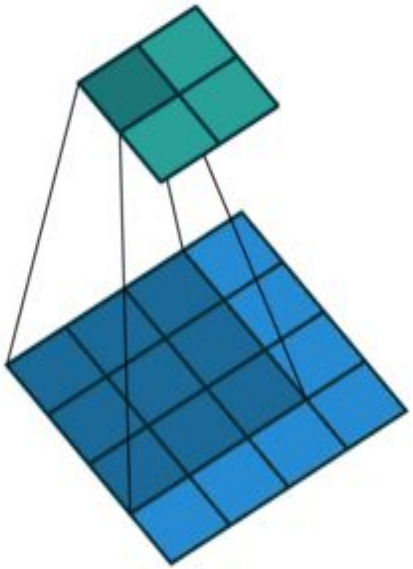
Blurred Image (5x5)

3x3 vs 5x5 uniform blur kernels

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \end{bmatrix}$$

Padding and stride effects



Convolution of 3x3 and stride = 1 without padding

Effect: the output loses one pixel on each dimension

Border padding

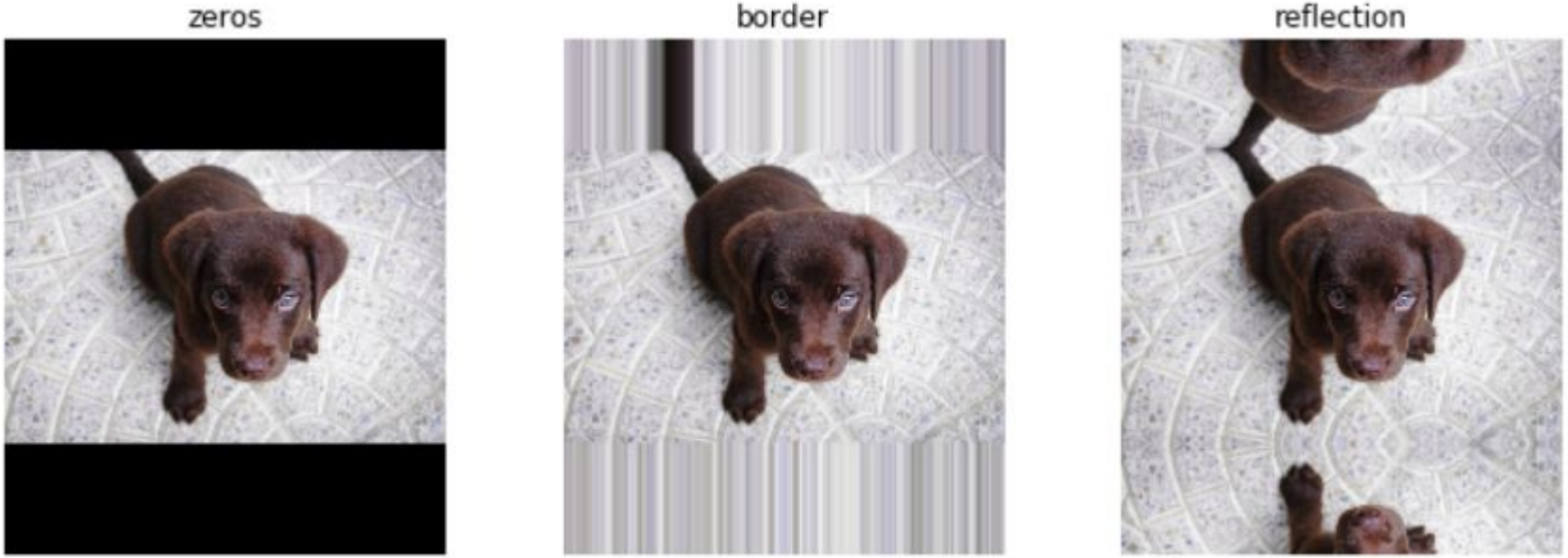
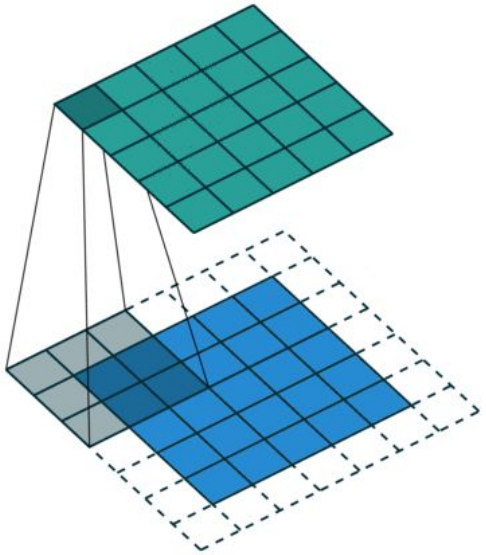


Image from the fastai documentation

“Automatic” feature learning

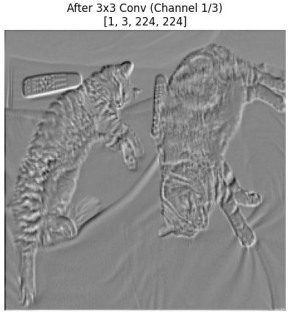


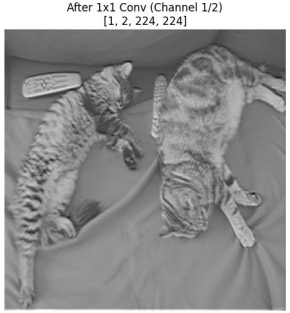
$$\begin{pmatrix} -0.64 & -0.74 & 0.91 \\ -0.96 & 0.48 & -0.46 \\ 0.56 & 0.67 & -0.08 \end{pmatrix}$$

$$\begin{pmatrix} -0.89 & 0.40 & 0.12 \\ -0.45 & 0.38 & 0.67 \\ 0.91 & -0.71 & -0.35 \end{pmatrix}$$

$$\begin{pmatrix} 0.65 & 0.39 & -0.75 \\ 0.36 & -0.57 & 0.34 \\ 0.61 & 0.69 & 0.43 \end{pmatrix}$$

$$\begin{pmatrix} -0.64 & -0.74 & 0.91 \\ -0.96 & 0.48 & -0.46 \\ 0.56 & 0.67 & -0.08 \end{pmatrix}$$





These weights are <u>adjusted to minimize the loss function</u>

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

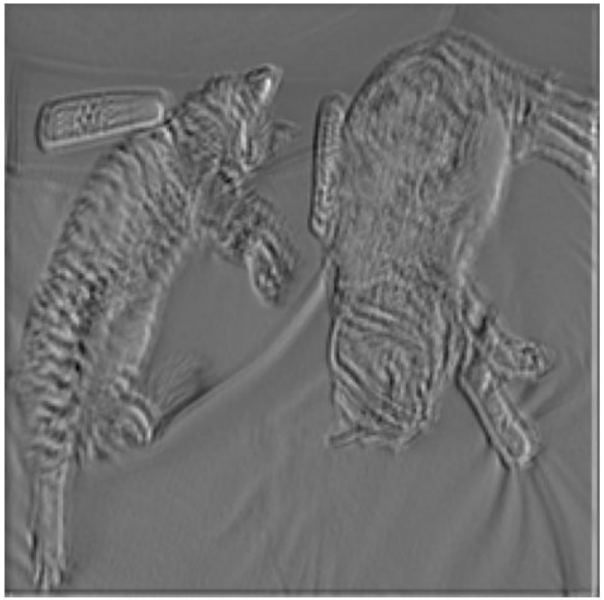
Convolutions of 3x3 and stride = 1 padded with zeros

Effect: the output preserves the original image size while producing a “different-looking” image (based on the values of the weights)

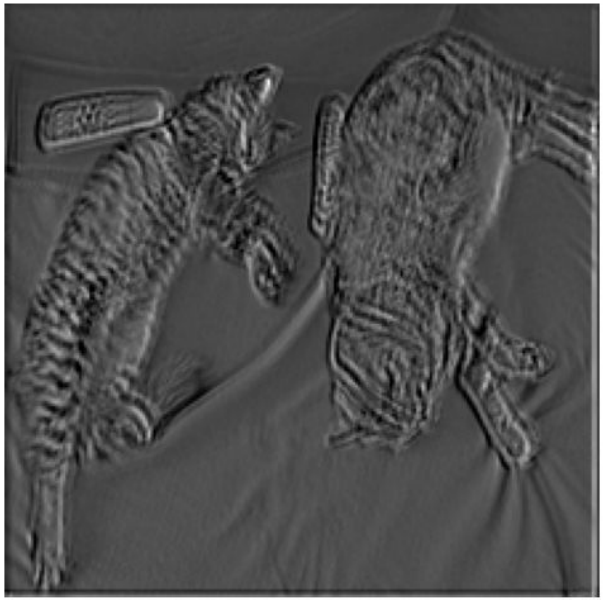
ReLU adds non-linearity to activations

Input with Padding

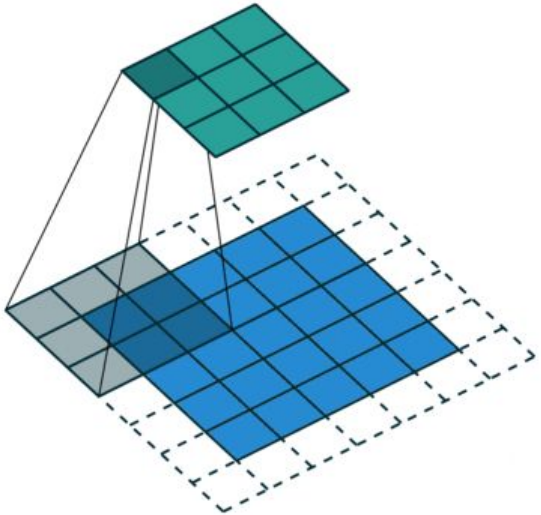
After Conv2d



After ReLU

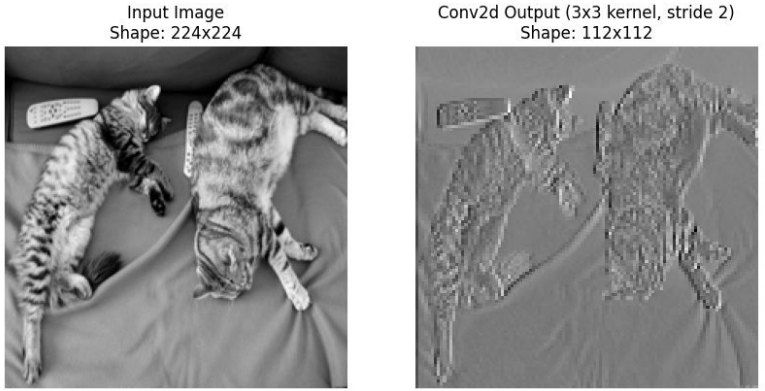


Downsampling images with stride > 1 convolutions



Convolution of 3x3 and stride = 2 padded with zeros

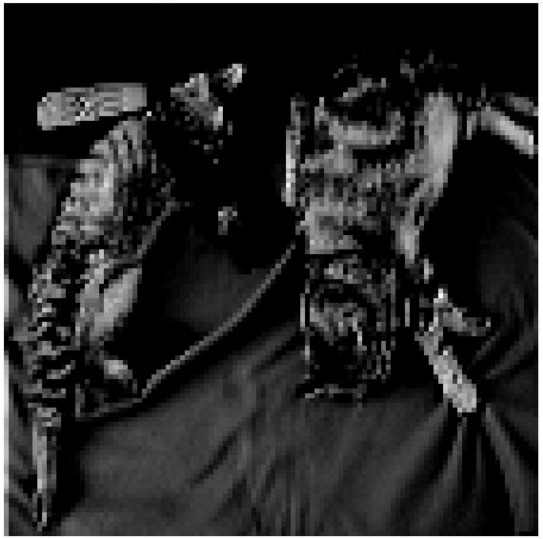
Effect: the output is downsampled to about half its size



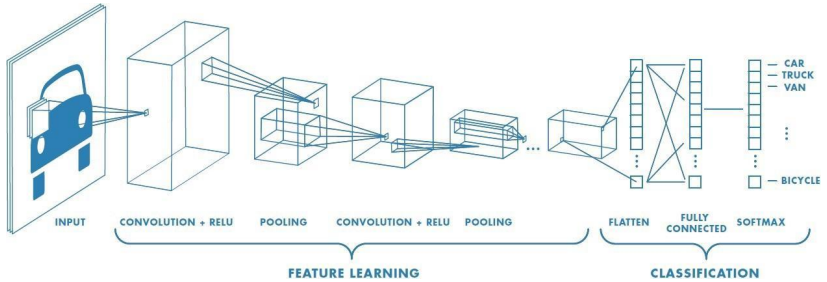
Convolutions are concatenated

Original Image
Shape: torch.Size([224, 224])

After First Conv + ReLU (stride = 2)
Shape: torch.Size([112, 112])

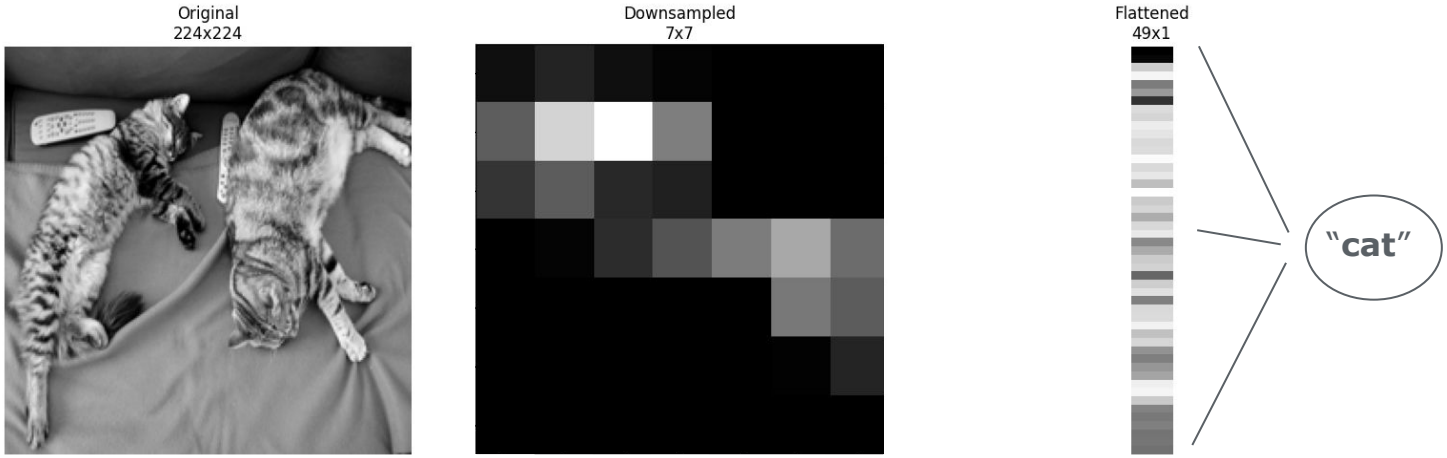


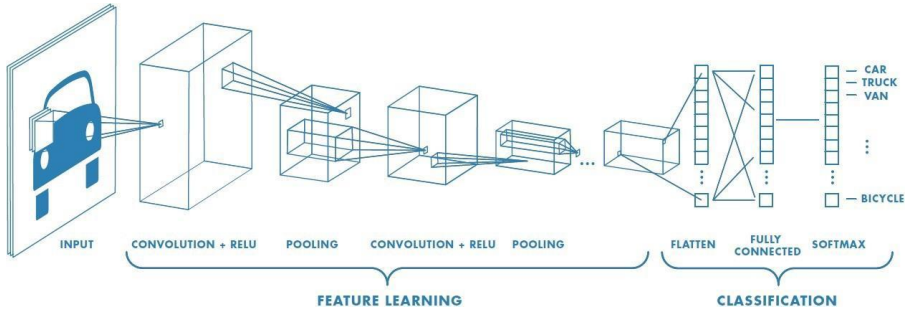
After Second Conv + ReLU (stride = 2)
Shape: torch.Size([56, 56])



$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

We use convolutions as “feature extractors” for classifiers





‘Unrolling’ convolutions

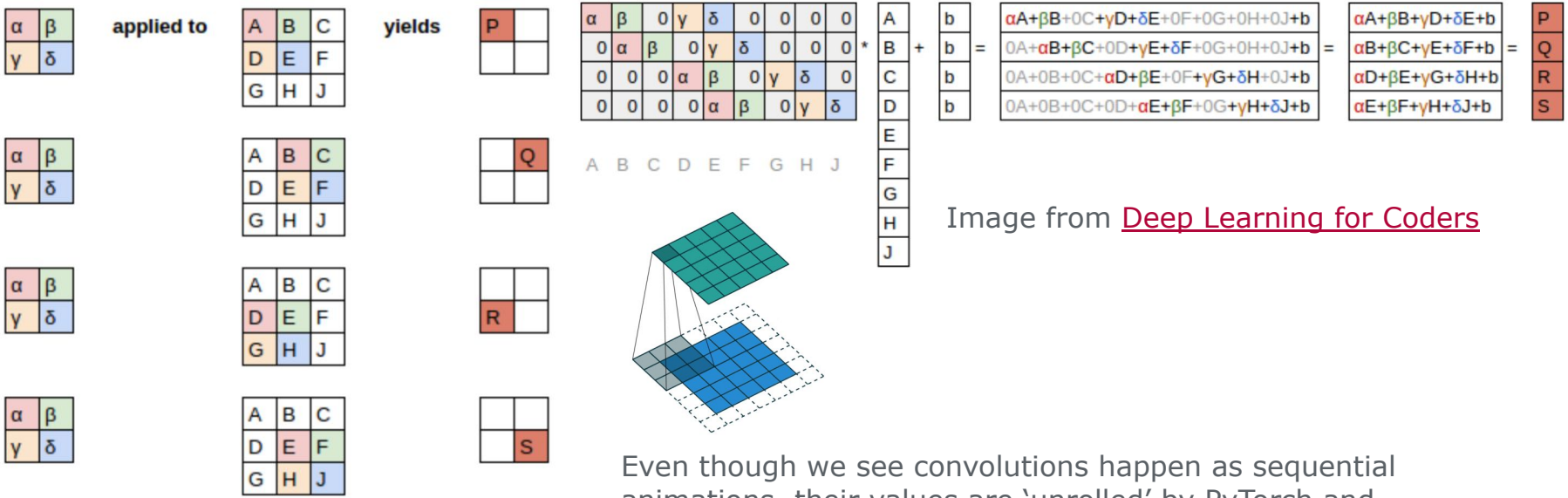


Image from [Deep Learning for Coders]

Even though we see convolutions happen as sequential animations, their values are ‘unrolled’ by PyTorch and applied as parallel matrix multiplications **simultaneously**

Summary

Convolutions and feature learning

- Convolutional layers create new images (“new features”) through learnable weights
- Convolutions were traditionally hand-crafted, now we use the neural network to set their weights
- Classifiers flatten these learned features before feeding them to a feedforward network
- Convolutions are “unrolled” by PyTorch in order to do parallel processing

Effects of receptive field size, padding, and stride

- The size of the receptive field influences how many pixels we consider
- Convolutions with padding preserve spatial dimensions
- Strided convolutions allow us to do learnable downsampling

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Summary

Dataset objects wrap images and labels together

- Provide a standard way to access data through `__getitem__` and `__len__` methods

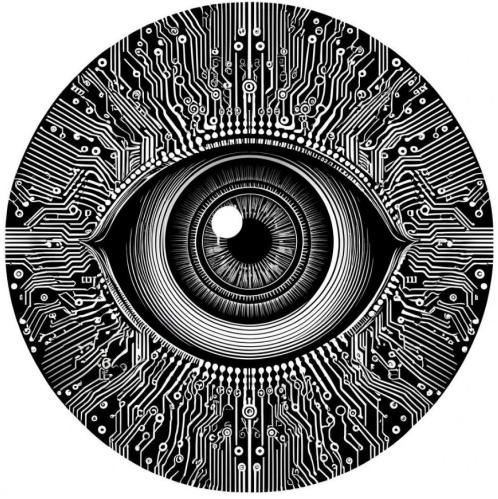
Dataloaders split datasets in batches

- Dataloaders shuffle the training data and maintain sequential order for validation and test sets

Dataloaders are fundamental to implement Stochastic Gradient Descent

- Their random sampling and shuffling of training samples provide the ‘stochastic’ part of Stochastic Gradient Descent

Pooling in Neural Networks

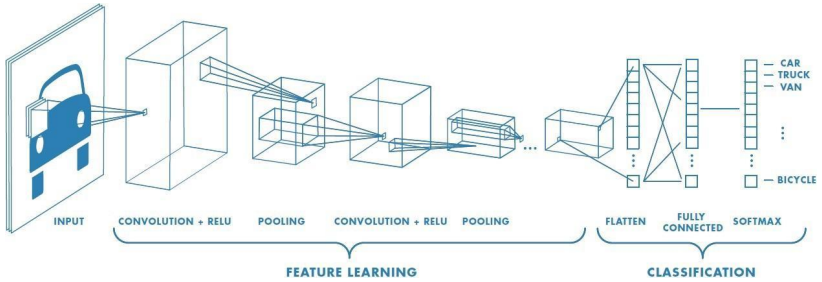


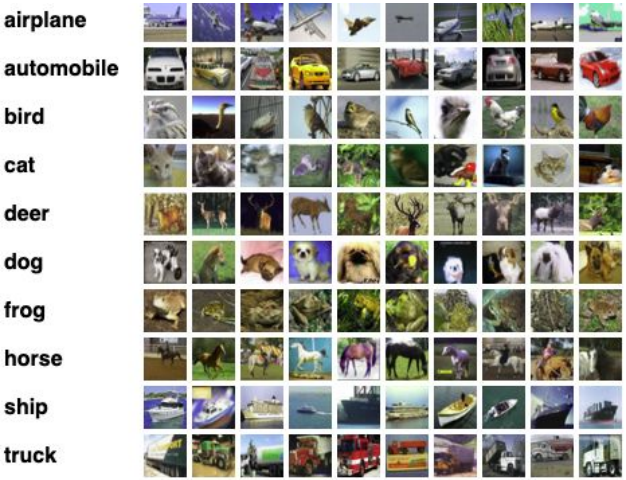
Learning goals

- Explore image downsampling and reduction of network parameters with mean, max, and global pooling

Architecture of a convolutional network with pooling

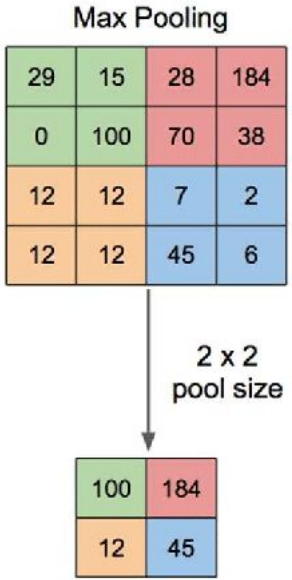
```
nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10), # 10 classes, we are working with CIFAR 10
        )
```



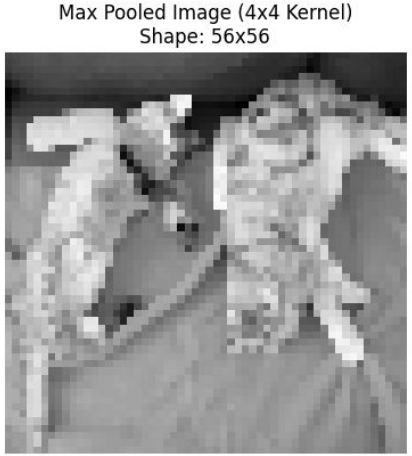


The CIFAR-10 dataset

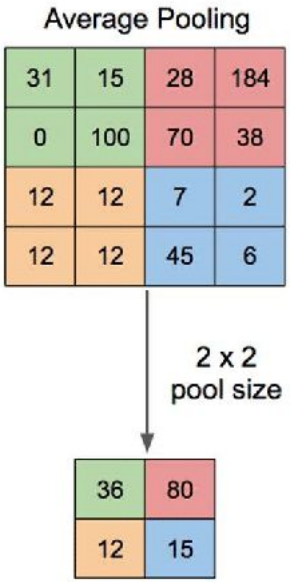
Max Pooling


Max Pooling
29
15
28
184
0
100
70
38
12
12
7
2
12
12
45
6
2 x 2
pool size
100
184
12
45


Input Image
Shape: 224x224

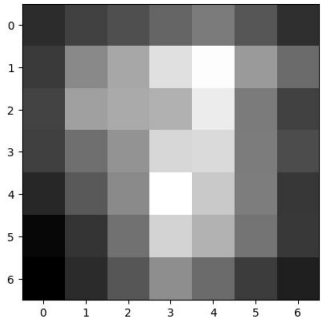

Max Pooled Image (4x4 Kernel)
Shape: 56x56

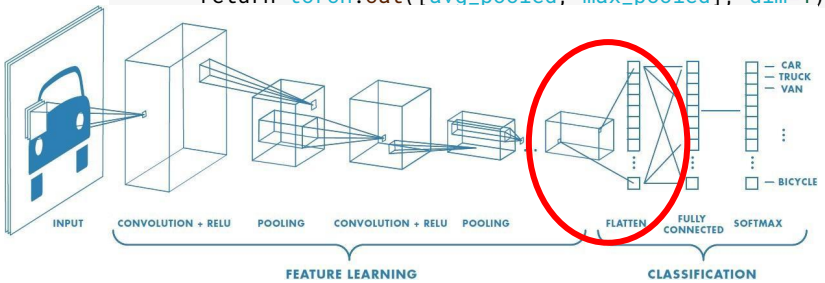
Mean Pooling

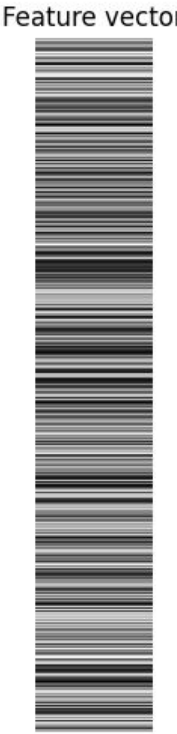


Global pooling aka adaptive pooling

```python
import torch.nn as nn


class GlobalPooling(nn.Module):
    def __init__(self):
        super().__init__()
        # Computes the average value of each activation map
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        # Selects the max value of each activation map
        self.max_pool = nn.AdaptiveMaxPool2d((1, 1))

    def forward(self, x):
        avg_pooled = self.avg_pool(x).flatten(1)
        max_pooled = self.max_pool(x).flatten(1)
        # Concatenates both poolings to produce a feature vector
        return torch.cat([avg_pooled, max_pooled], dim=1)
```

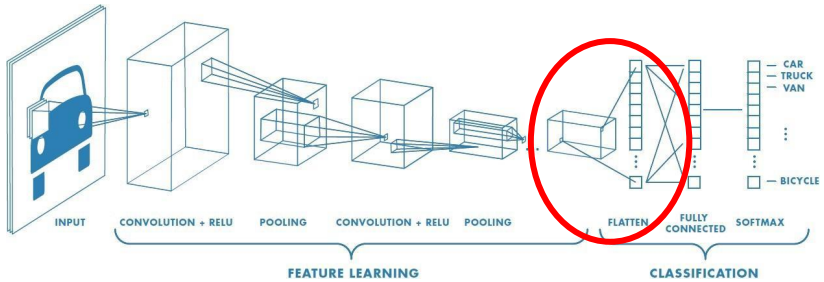






Using global pooling to reduce parameters

```
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),

    GlobalPooling(),   # This replaces the Flatten layer

    nn.Linear(64, 120),
    nn.ReLU(),
    nn.Linear(120, 84),
    nn.ReLU(),
    nn.Linear(84, 10)
)
```



Summary

Pooling is a way to compress information

- Pooling allows us to do lossy compression while retaining important visual features
- Convolutions with stride > 1 achieve a similar effect at the cost of a higher number of parameters

Global pooling is an alternative to flattening activation maps

- We can create feature vectors (aka embeddings) using the global mean and/or max pooling operations

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Resources

- Github Repository
- YouTube playlist
- Discord channel
  #practical-computer-vision-workshops