

Learning Deep Learning with PyTorch

(4) Convolutional Neural Networks

Qiyang Hu

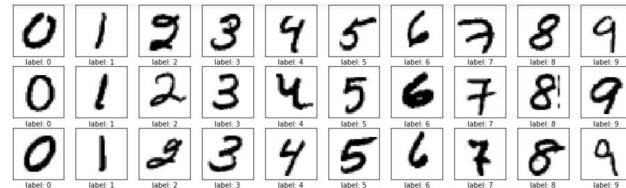
IDRE

Oct 27, 2020

“Hello-World” Problems in Image Classification

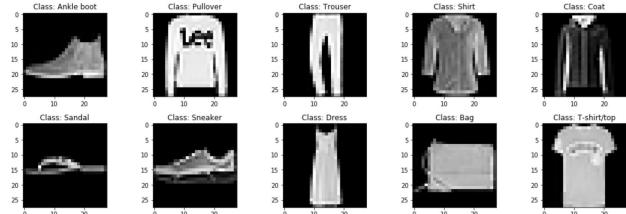
- MNIST dataset of handwritten digits

- 28x28 pixel grey images
- 10 classes
- training set: 60,000, test set: 10,000



- Fashion MNIST dataset

- 28x28 pixel grey images
- 10 classes
- training set: 60,000, test set: 10,000



- CIFAR-10

- 32x32 pixel color images
- 10 classes
- training set: 50,000, test set: 10,000



Dogs vs. Cats Kaggle Challenge

- Redux: Kernels Edition
 - Submission scored by the probability of dogs using log loss

$$L = -\frac{1}{n} \sum_{i=1}^n \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

- Dataset
 - Training set: 25,000 dogs and cats images
 - Testing set: 12,500 images
 - Images with different sizes
 - Neural network needs fixed sized input.
 - We will resize images to 150x150 pixels
 - Images are colored
 - Represented by Red-Green-Blue channels
 - One image \Rightarrow 150x150x3 matrices



Digitalization for Color Images

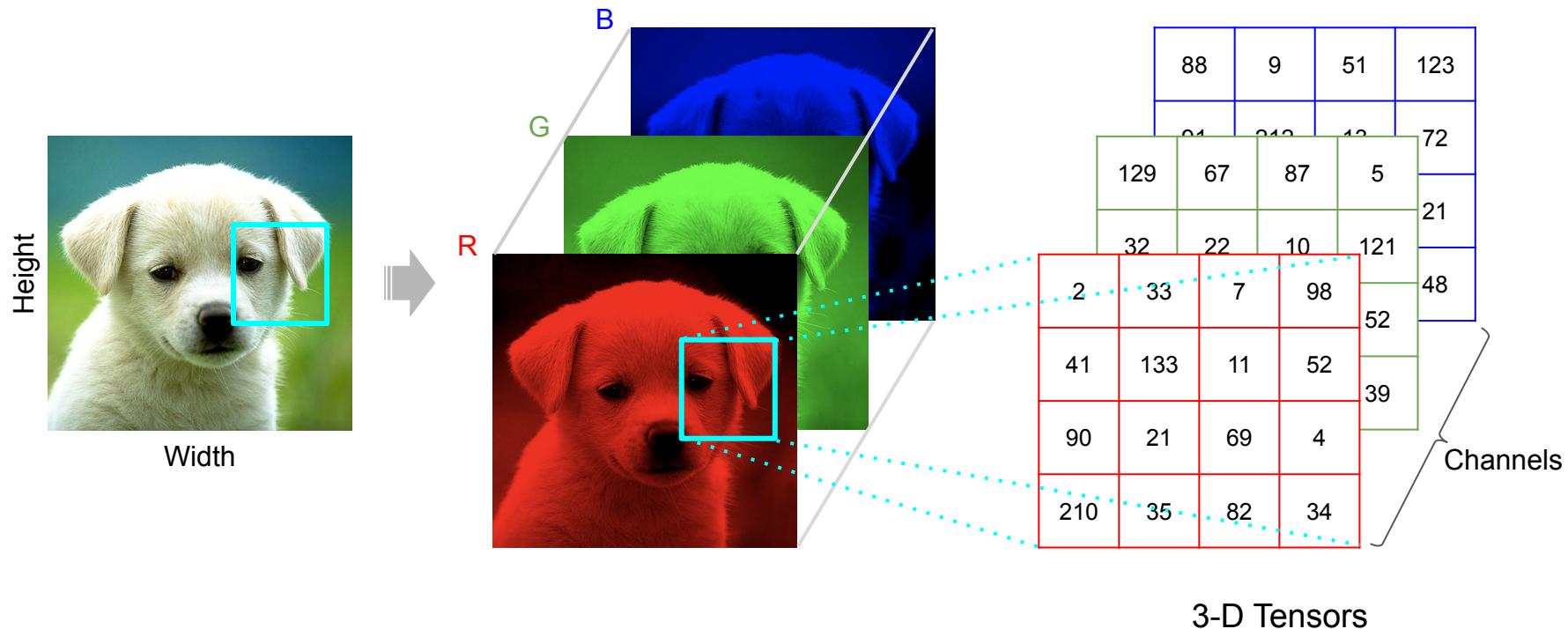


Image data conversion in PyTorch

- PIL to convert JPG to PIL Image
 - `pil.Image.open(path).convert('RGB')`
- Resize to the uniform sizes for all images
 - `torchvision.transforms.Resize((150, 150))`
- Convert to tensors:
 - `torchvision.transforms.ToTensor()`
 - Indexes $(H \times W \times C) \Rightarrow (C \times H \times W)$
 - Range $[0, 255] \Rightarrow [0.0, 1.0]$

Python Image Library (PIL)

- Pillow as newer versions
- Various image processing
- Per-pixel manipulations

Torchvision is a package for computer vision, containing:

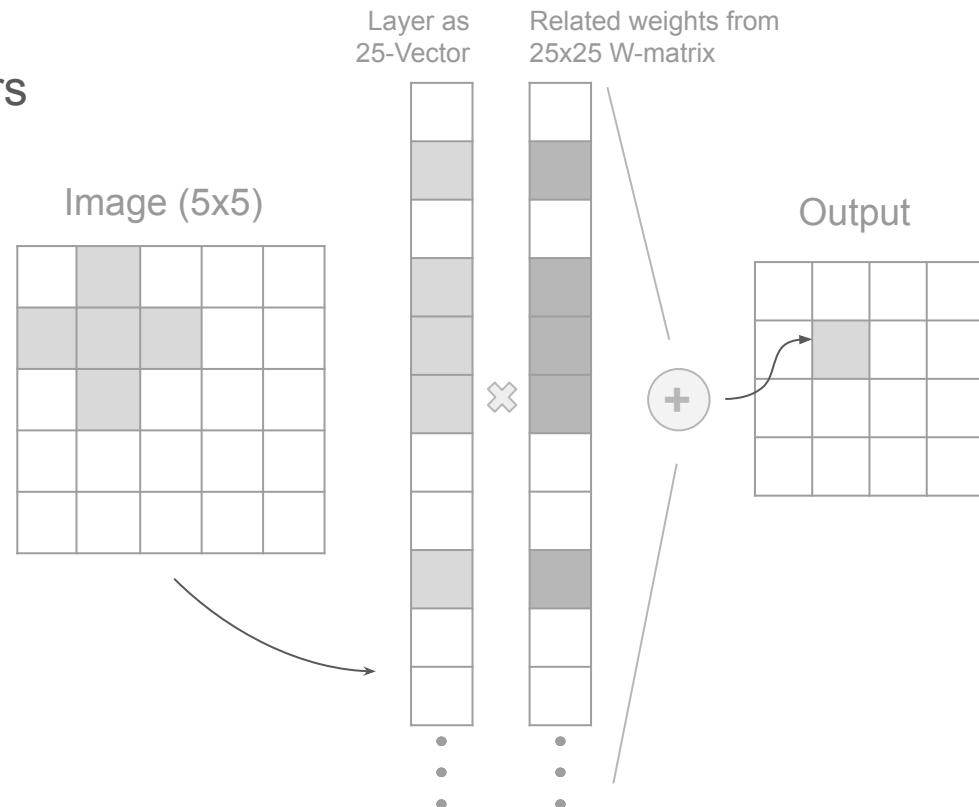
- Popular datasets
- Model architectures
- Image transformations

Datasets and Data loading

- Defining the dataset class
 - Subclassing `torch.utils.data.Dataset`
 - PyTorch dataset object requires 2 methods:
 - `__len__()`
 - `__getitem__()`
 - Wrapping conversions in `__getitem__()`
- Loading the dataset with `torch.utils.data.DataLoader`
 - Batching the data
 - Shuffling the data
 - Loading the data in parallel using multiprocessing workers

Limits of fully connected model

- Not scale well with pixel numbers
 - 1024x1024 RGB image
One 1024-feature hidden layer
 - → 3 billion parameters
→ 12 GB ram for 32-bit floats
→ Hard to fit in a GPU
- Not translation-invariant
 - Shifting 1 pixel
→ Re-learn!



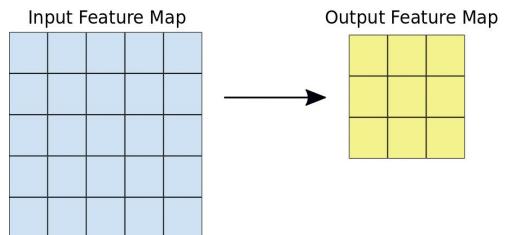
Convolutional Neural Networks (CNNs)

- A special network architecture

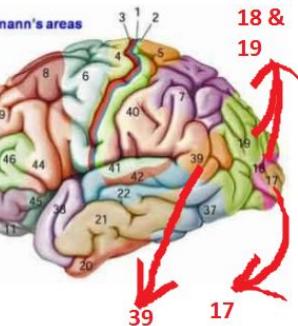
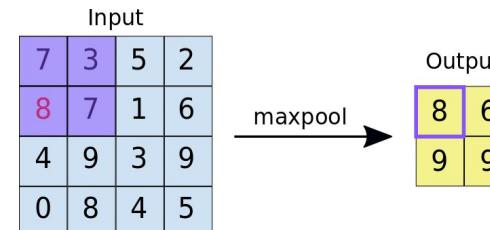
- Primary concepts inspired by the physiological visual pathways
- Visual signals processed by *hierarchical* functions with structures

- 3 steps in CNNs:

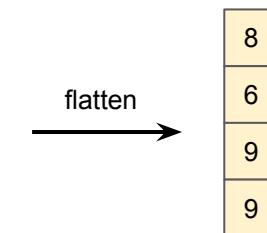
- Convolutions to extract as tiles



- Poolings to downsample



- Flattening



- Logic behind CNNs

- Sparse connectivity (characteristic features in smaller local regions)
- Parameter equivariance & sharing (features appear in different locations)
- Translation invariance (some sampling will not lose main information)

One Channel, One Filter

0	0	0	0	0	0
0	105	102	100	97	96
0	103	99	103	101	102
0	101	98	104	102	100
0	99	101	106	104	99
0	104	104	104	100	98

Image Matrix

0	-1	0
-1	5	-1
0	-1	0

Kernel Matrix

320					

Output Matrix

$$\begin{aligned} & 0 * 0 + 0 * -1 + 0 * 0 \\ & + 0 * -1 + 105 * 5 + 102 * -1 \\ & + 0 * 0 + 103 * -1 + 99 * 0 = 320 \end{aligned}$$

Convolution with horizontal and vertical strides = 1

[Interactive Source](#)

Example Input Image



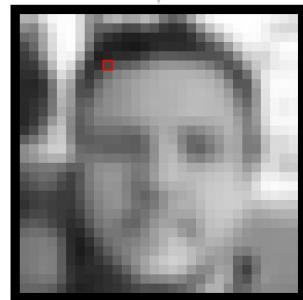
Identity

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$



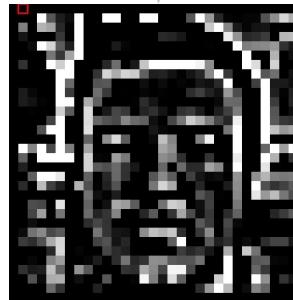
Blur

$$\begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$



Outline

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$



Sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$



Multiple Channels

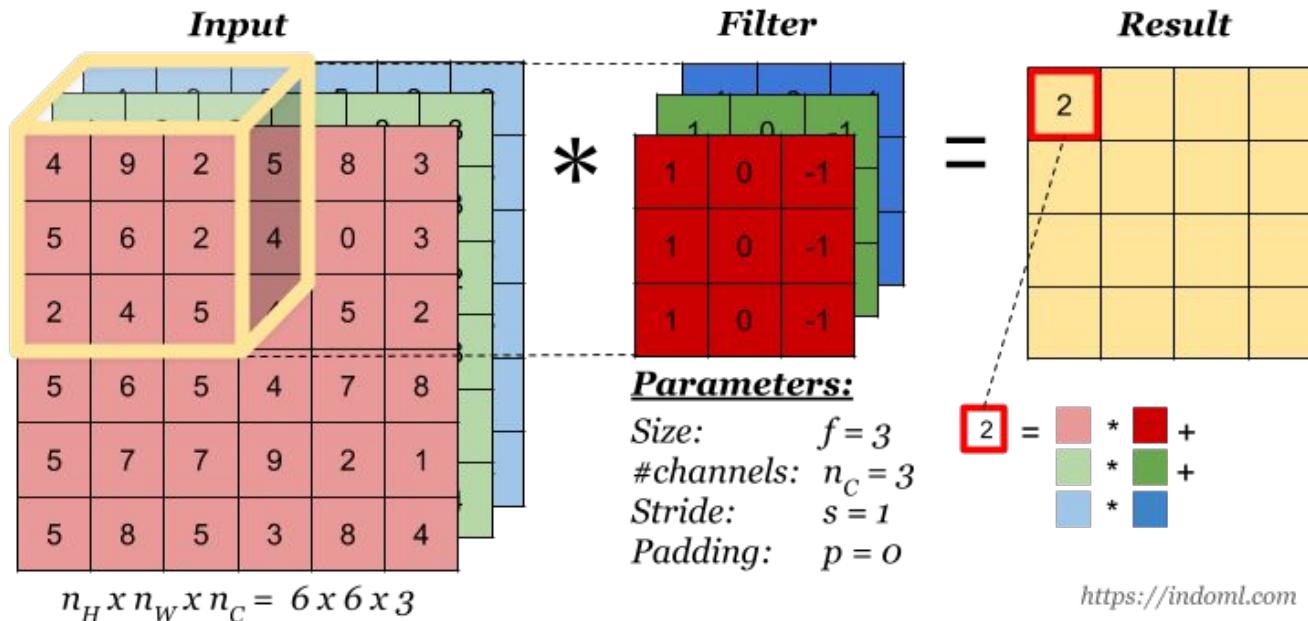
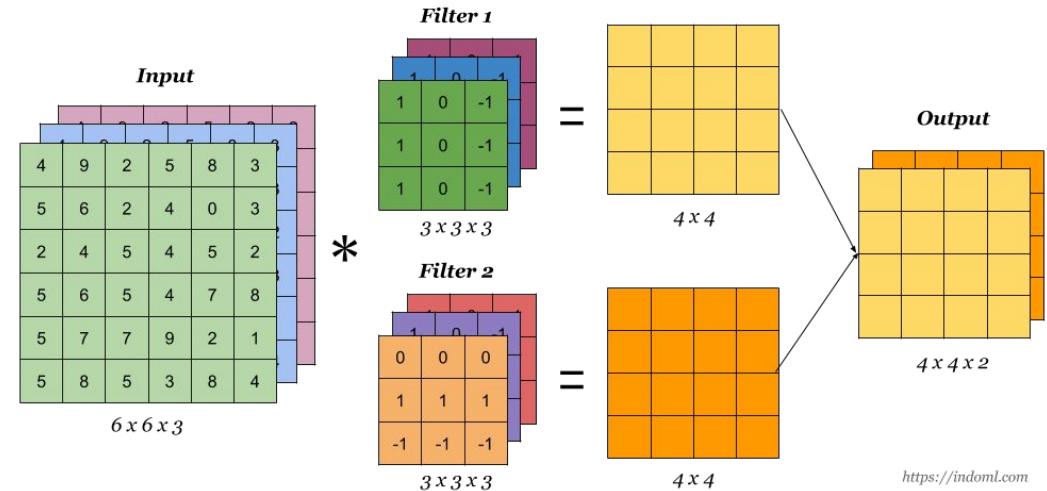
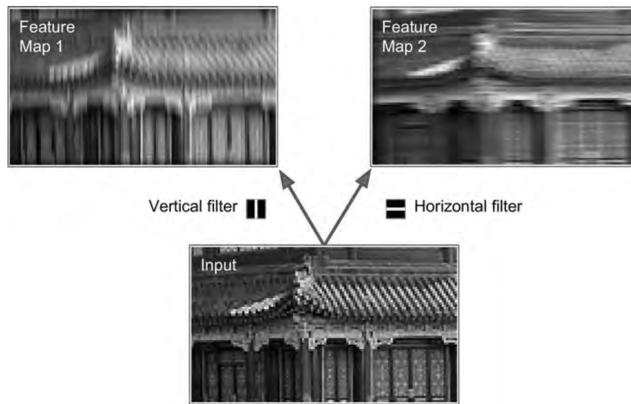


Figure [Source](#)

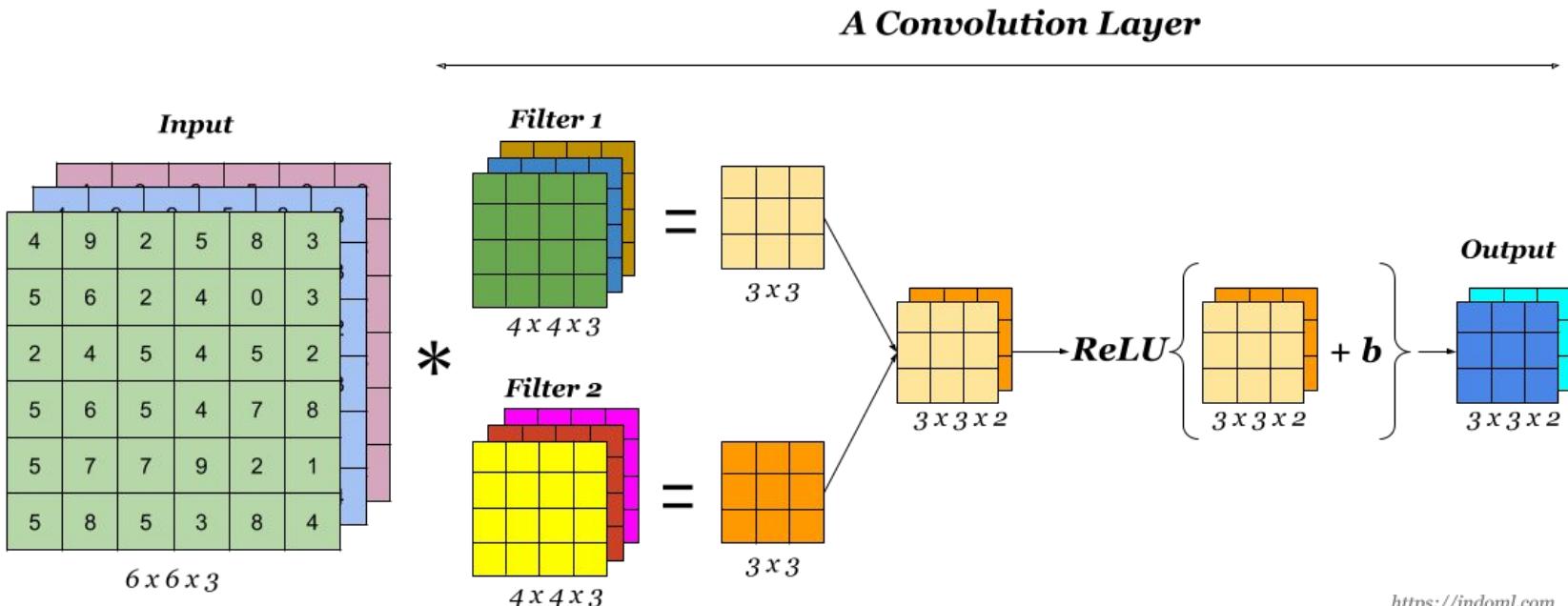
Stacking Multiple Filters (Feature Maps)



Figures from Aurélien Géron's [1st Ed. Book](#)

Figure [Source](#)

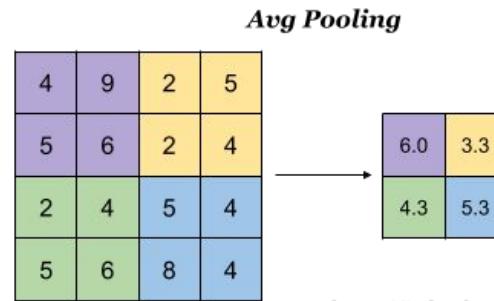
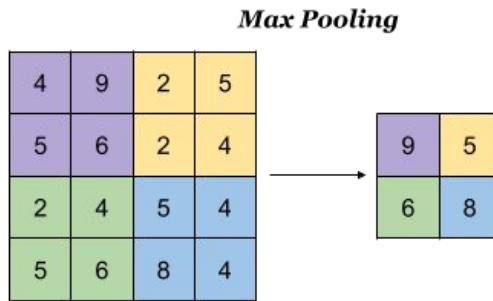
A Convolutional layer



<https://indoml.com>

Figure [Source](#)

Pooling Layer



<https://indoml.com>

Figure [Source](#)

- Assuming downsampling will not lose the major information.



Figures from Aurélien Géron's [1st Ed. Book](#)

Architecture of Convolutional Neural Networks

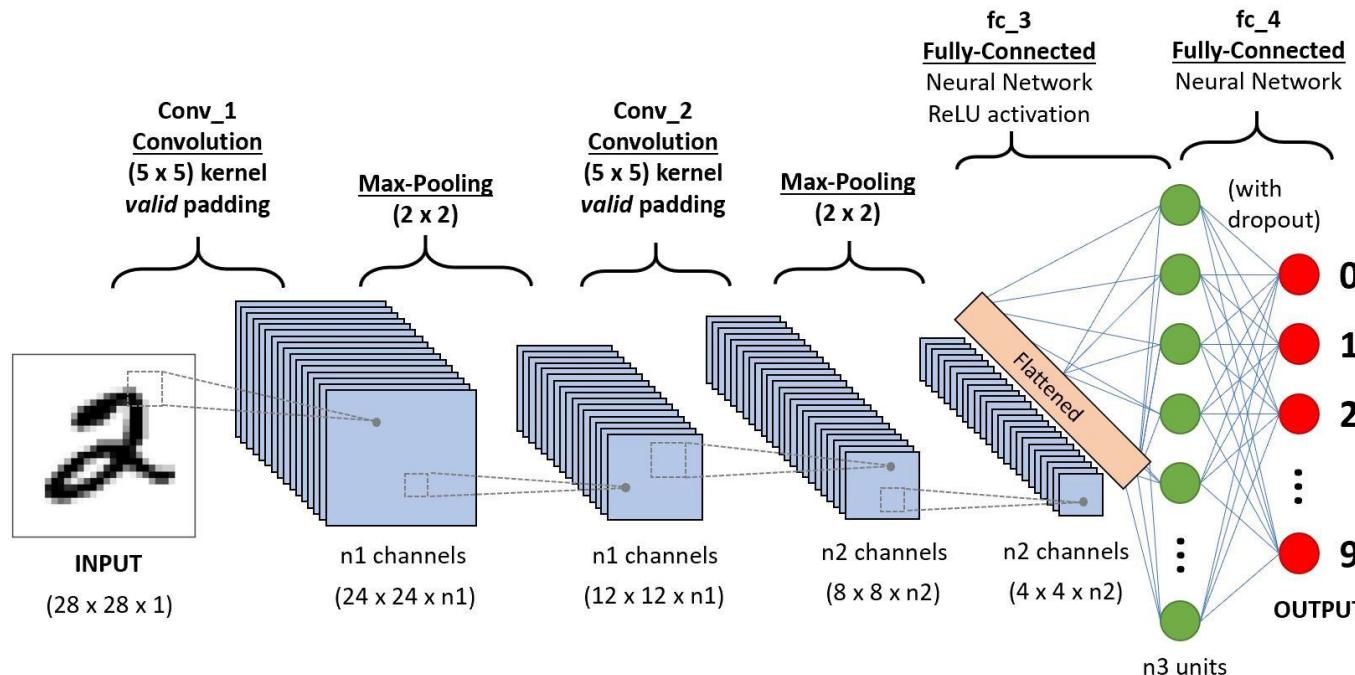


Figure [Source](#)

Convolutional layers as feature extractors

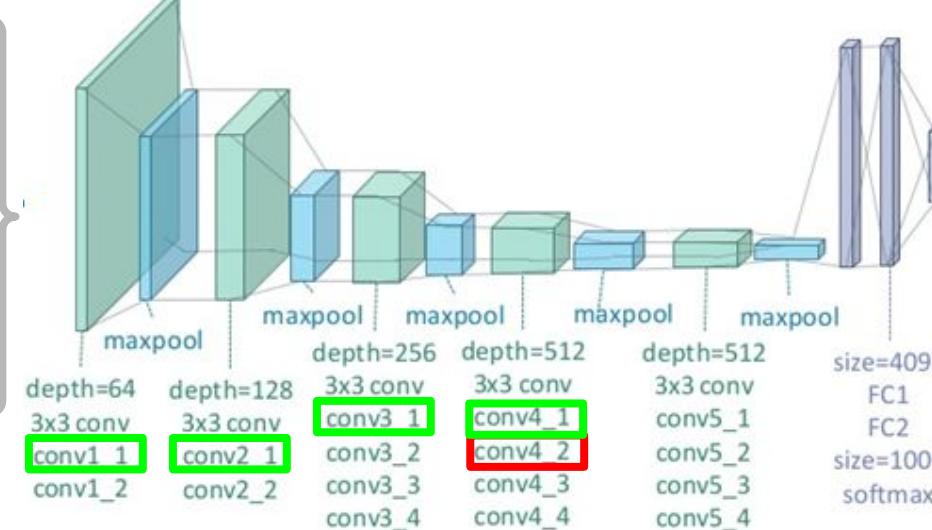


Content representation
of a photograph



Style representation
of the artwork

VGG 19

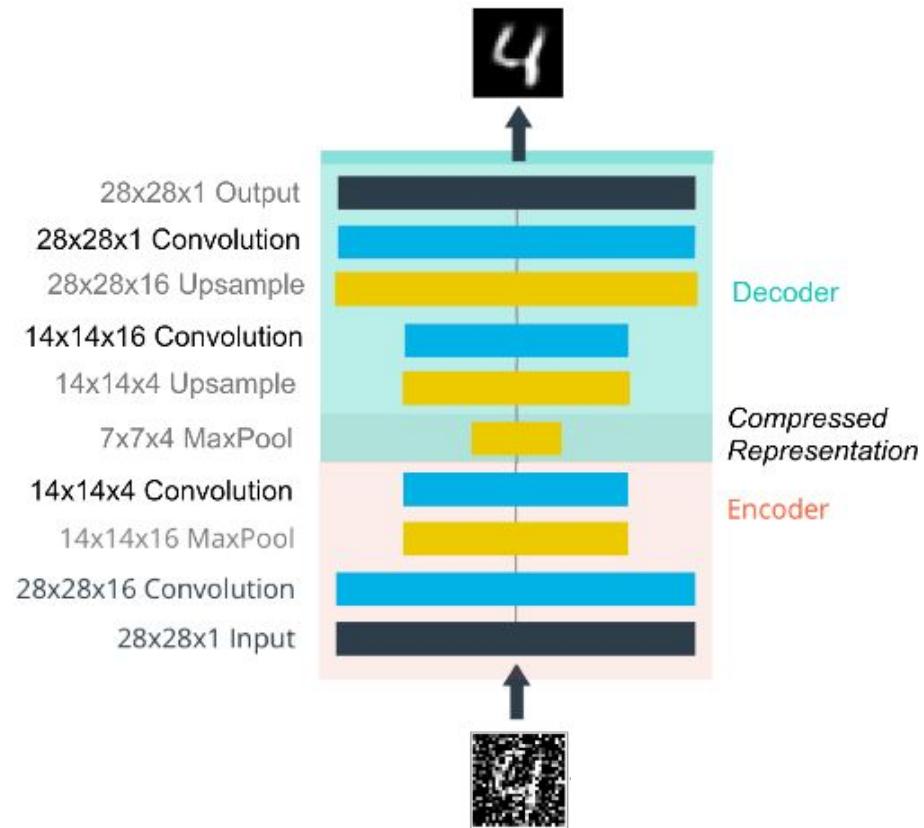
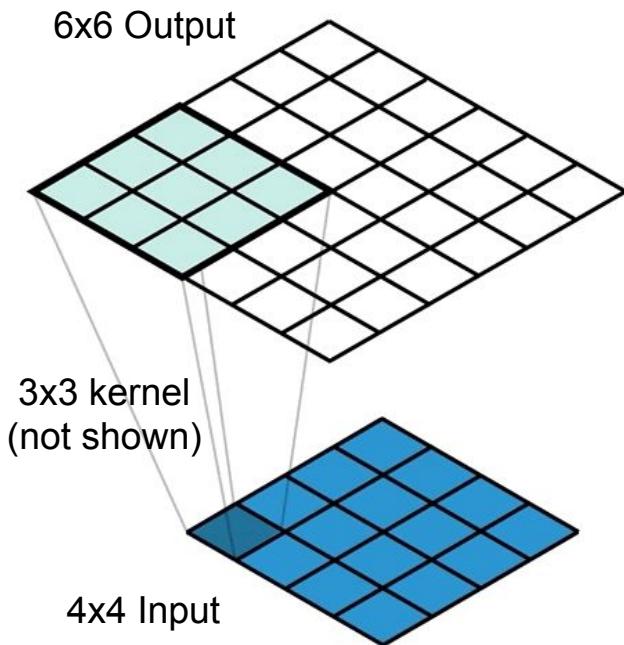


[Style Transfer Paper \(2016\)](#)



Style transferred
art image

Convolution and “Deconvolution”: Autoencoder



Construct CNN architecture for Dogs-vs.-Cats Problem

- 4 Convolution layers:

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

- Input size: (N, C_{in}, H, W)
- Output size: $(N, C_{out}, H_{out}, W_{out})$
- Activation function: `torch.nn.functional.relu(...)`

- MaxPooling layer:

`torch.nn.max_pool2d(...)`

- Kernel size: 2
- Default: stride=None, padding=0, dilation=1

- Flattened layer

- Manually flattening tensor by views

- Dense (linear) layer

`torch.nn.Linear(in_features, out_features)`

- Units: 512 and 2
- Activation: ‘relu’ and ‘softmax’

```
class CatAndDogNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 32, kernel_size=(3, 3))
        self.conv2 = nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size=(3, 3))
        self.conv3 = nn.Conv2d(in_channels = 64, out_channels = 128, kernel_size=(3, 3))
        self.conv4 = nn.Conv2d(in_channels = 128, out_channels = 128, kernel_size=(3, 3))
        self.fc1 = nn.Linear(in_features= 128 * 7 * 7, out_features=512)
        self.fc2 = nn.Linear(in_features=512, out_features=2)

    def forward(self, x):
        X = F.relu(self.conv1(X))          → (148,148,32)
        X = F.max_pool2d(X, 2)            → (74,74,32)

        X = F.relu(self.conv2(X))          → (72,72,64)
        X = F.max_pool2d(X, 2)

        X = F.relu(self.conv3(X))
        X = F.max_pool2d(X, 2)

        X = F.relu(self.conv4(X))          → (7,7,128)
        X = F.max_pool2d(X, 2)            → 6272

        X = X.view(-1, self.num_flat_features(X)) → 6272
        X = F.relu(self.fc1(X))
        X = self.fc2(X)
        return X

    def num_flat_features(self, x):
        size = x.size()[1:] # all dimensions except the batch dimension
        num_features = 1
        for s in size:        # Get the products
            num_features *= s
        return num_features
```

Save and Load the model in PyTorch

- Need to save the trained model
 - Colab's active session time is limited.
 - Models can be re-used at user's end (e.g. browser with tf.js or phone with tf.lite)
- PyTorch 3 core functions:
 - `torch.save`: saves a serialized object to disk
 - `torch.load`: deserializes pickled object files to memory
 - `torch.nn.Module.load_state_dict`: loads parameters using a deserialized state_dict
- Recommended usage (for inference):
 - `torch.save(model.state_dict(), PATH)`
 - `model.load_state_dict(torch.load(PATH))`
 - `model.eval()`
- Saving & loading a checkpoint for resuming training ([link](#))

Before running the colab demo in this workshop

1. Register a Kaggle account
 - o Kaggle.com → “Register”
2. Create Kaggle API token and download json file
 - o Sign in → Your Profile → “Account” → “Create New API Token”
3. Join the competition → “Join Competition”
 - o [Dogs-vs-Cats Challenge](#)

Colab Hands-on

bit.ly/LDL_02

Don't forget to

- Github Repo:
 - <https://github.com/huqy/idre-learning-deep-learning-pytorch>
- Slack workspace:
 - bit.ly/join-LDL
- Contact me
 - huqy@idre.ucla.edu
 - Direct message in Slack