Learning Convolutional Neural Networks (1)

Qiyang Hu
UCLA Office of Advanced Research Computing

Feb 11, 2022

In this talk

bit.ly/**LDL_repo**

Image Data

- Popular image dataset
- Image preprocessing

01

CNN Basics

- Origins & ideas
- CNN mechanism

02

CNN Variants

- Conv as feature extractor
- De-conv, 3d conv, …
- FCN and GCN

03

Demo

- CNN in pytorch
- Save/load pytorch model
- Setup of colab env

04

In this talk

Image Data

- Popular image dataset
- Image preprocessing

01

CNN Basics

- Origins & ideas
- CNN mechanism

02

CNN Variants

- Conv as feature extractor
- De-conv, 3d conv, …
- FCN and GCN

03

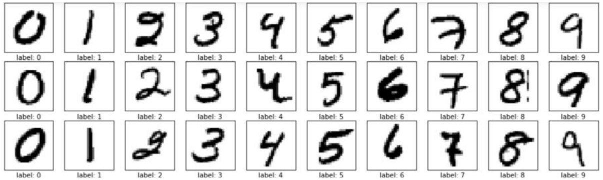
Demo

- CNN in pytorch
- Save/load pytorch model
- Setup of colab env

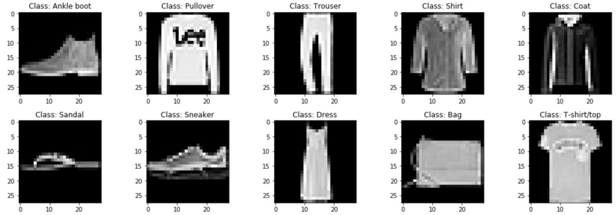
04

Popular Datasets for Computer Vision Tasks

- [MNIST dataset of handwritten digits](#)

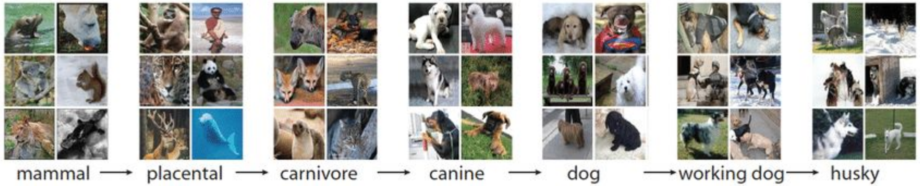


- [Fashion MNIST dataset](#)



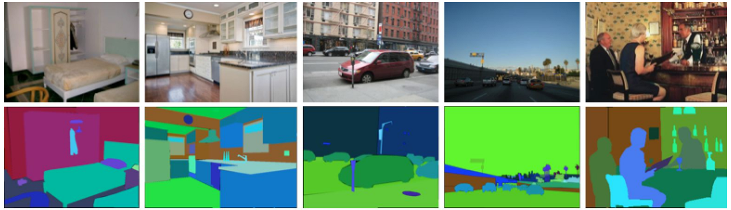
- [CIFAR-10](#)

- IMAGENET



- [Microsoft COCO](#)

- [ADE20K](#)



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 ⇒ 150x150x3 matrices

Digitalization for Color Images

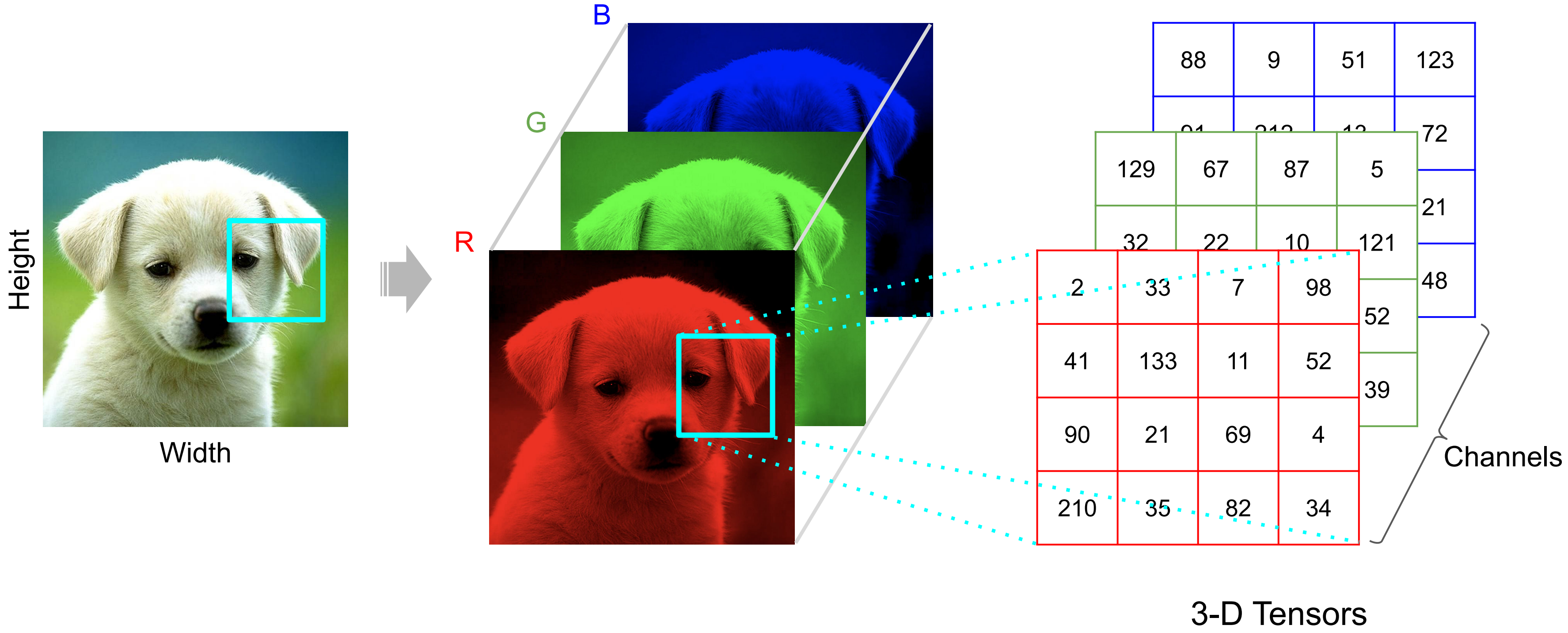


Image data conversion in PyTorch

- 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 × W × C) ⇒ (C × H × W)
    - Range [0, 255] ⇒ [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

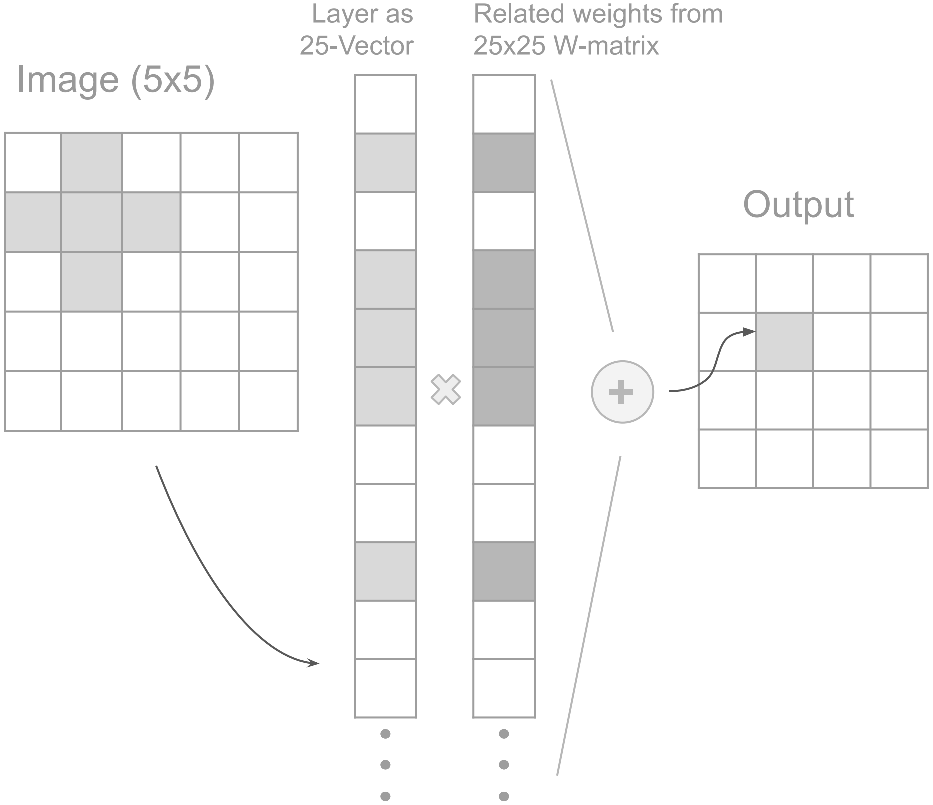
- Defining the dataset class
    - Subclassing torch.utils.data.Dataset
    - PyTorch dataset object requires 2 methods:
        - __len__()
        - __getitem__()
    - Wrapping conversions in __getitem__()

- Loading the dataset with torch.utils.data.DataLoader
    - Batching the data
    - Shuffling the data
    - Loading the data in parallel using multiprocessing workers

For Image Classification Tasks

- Naive steps:
  a. Matrices ⇒ Vectors
  b. Fully connected (FC) networks

- Limitations for FC models
  - 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-equivariant
    - Shifting 1 pixel ↛ Re-learn!



In this talk

Image Data

01

- Popular image dataset
- Image preprocessing

CNN Basics

02

- Origins & ideas
- CNN mechanism

CNN Variants

03

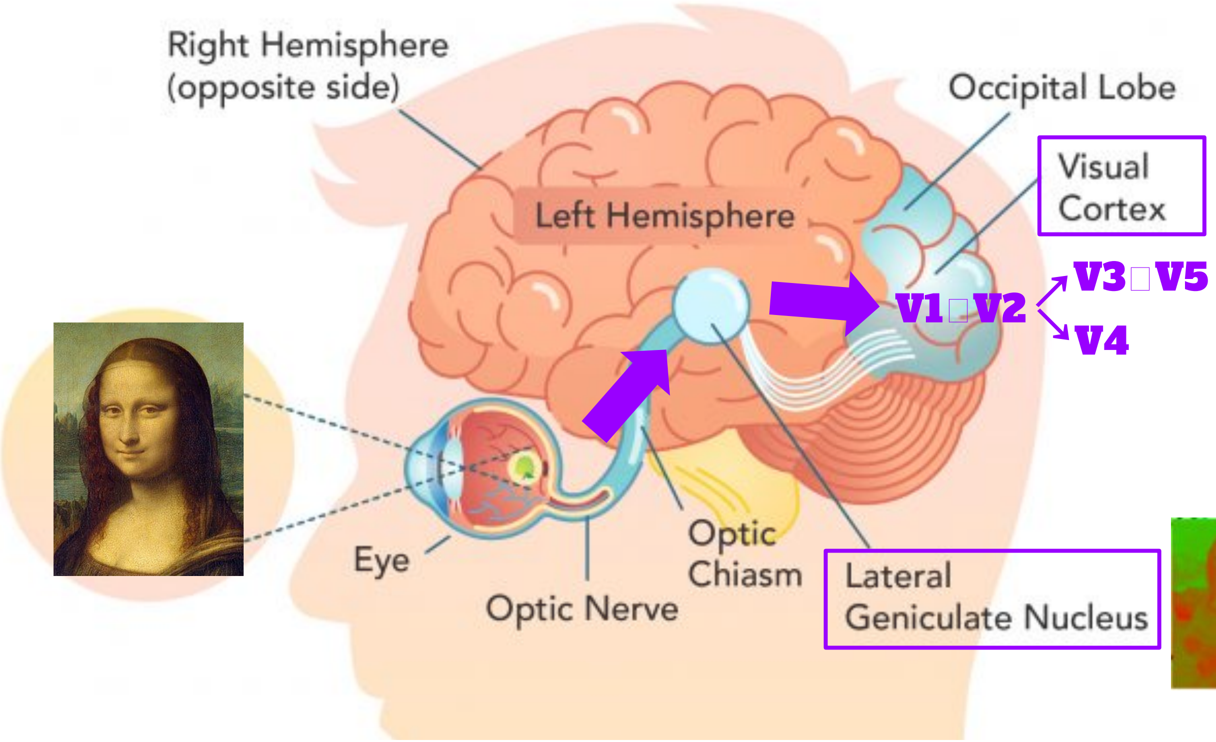
- Conv as feature extractor
- De-conv, 3d conv, …
- FCN and GCN

Demo

04

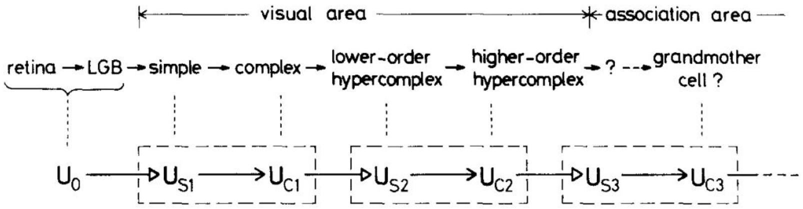
- CNN in pytorch
- Save/load pytorch model
- Setup of colab env

Inspiration from Cognitive Neuroscience



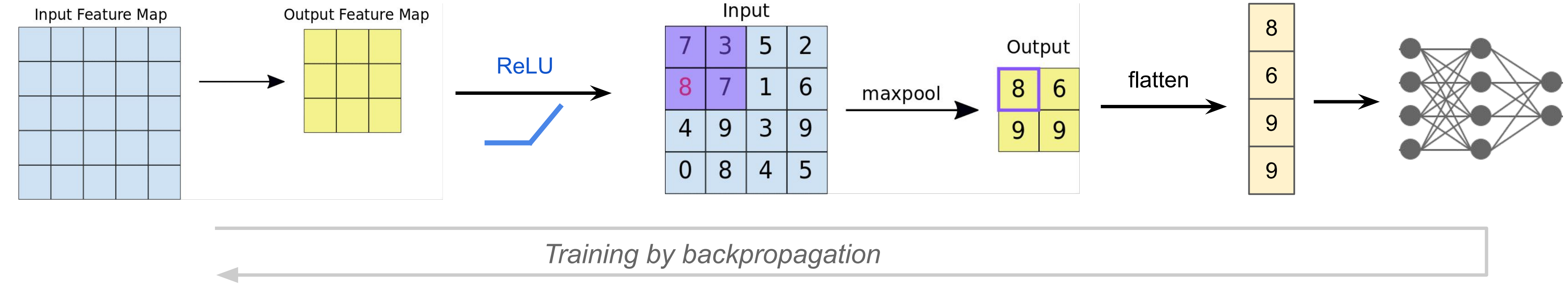
Convolutional Neural Networks (CNNs)

- Origins in computer vision
  - Neocognitron: K. Fukushima (1980)
    - convolutional layers, and downsampling layers
  - Modern CNN: Yann LeCun et al. (1989)
    - backpropagation

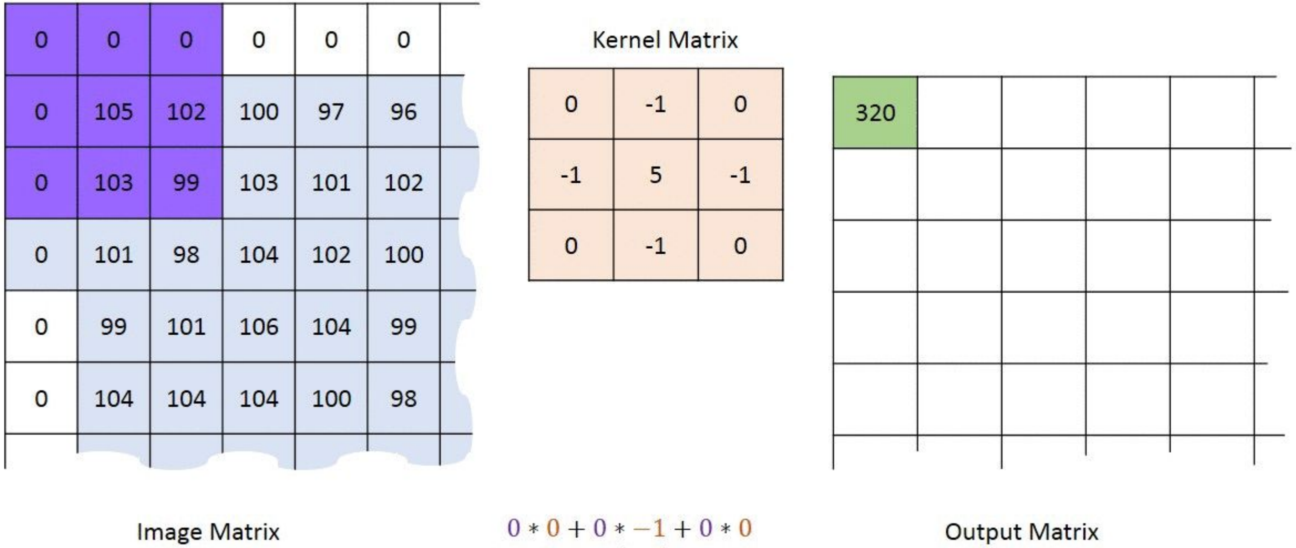


- Steps in CNNs:

Convolutions to extract as tiles | Activation | Poolings to downsample | Flattening | FCN



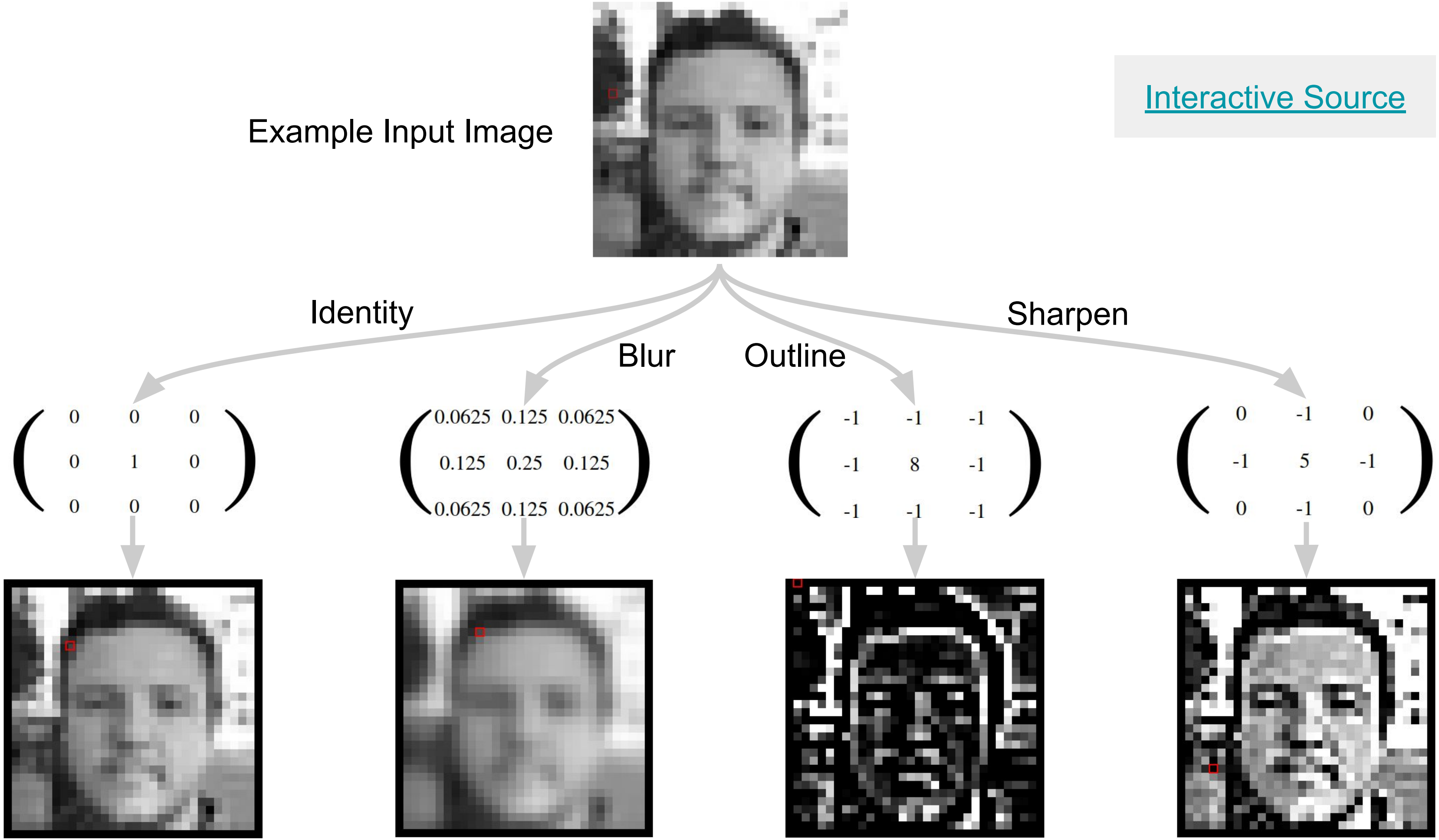
One Channel, One Filter



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

Convolution with horizontal and vertical strides = 1

[Interactive Source]



Multiple Channels

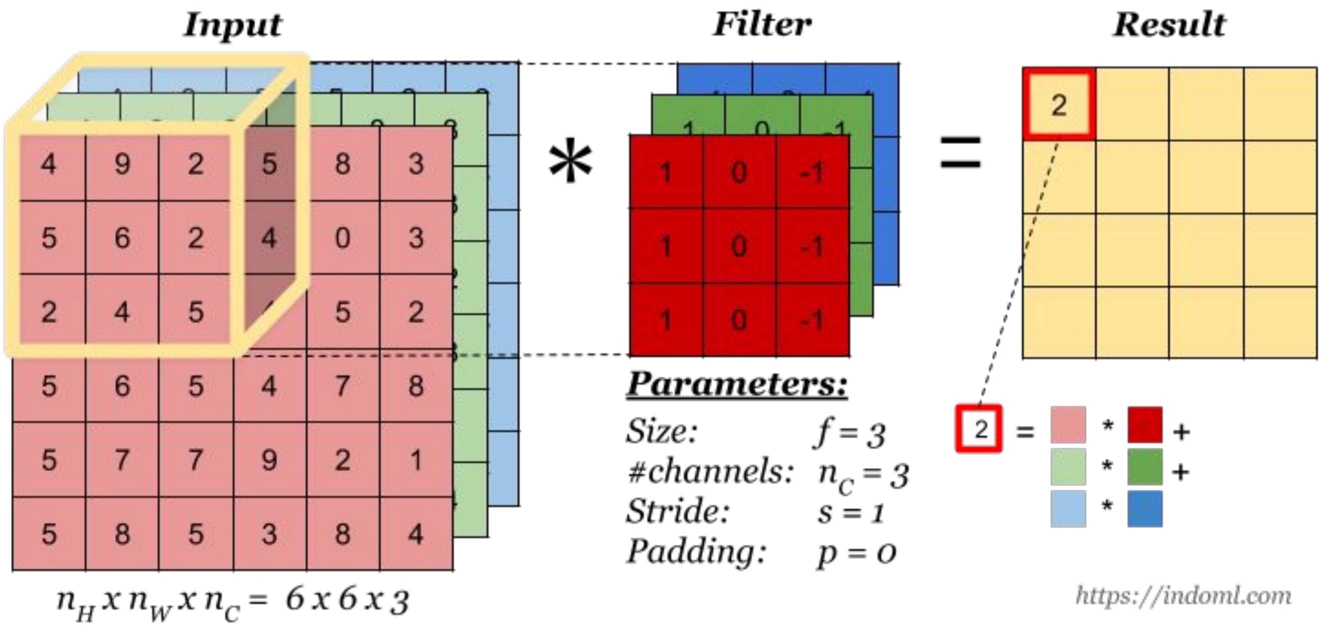
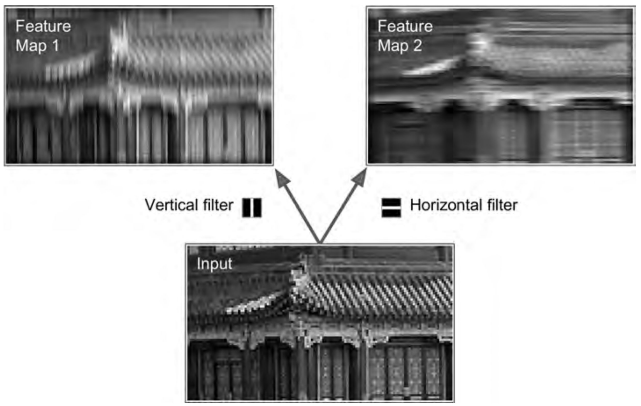
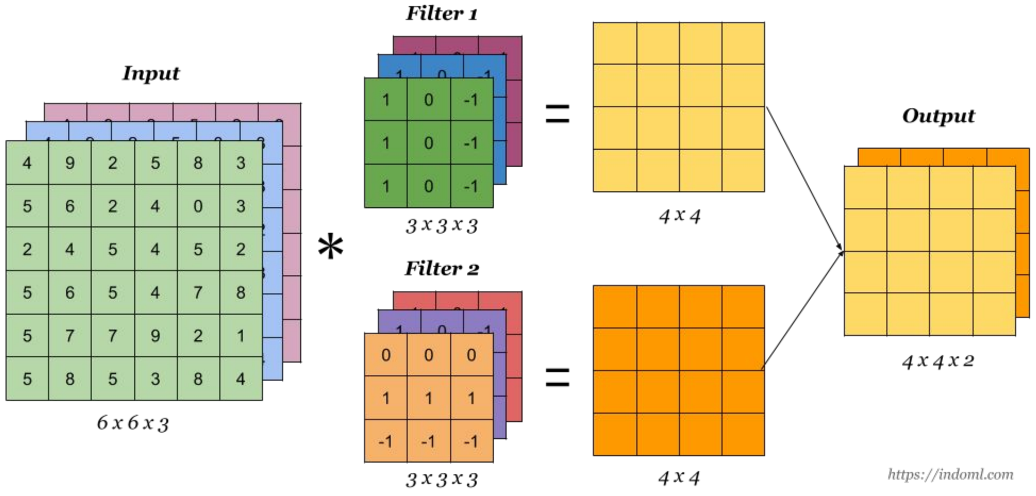


Figure Source

Stacking Multiple Filters (Feature Maps)





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

Figure [Source]

A Convolutional layer

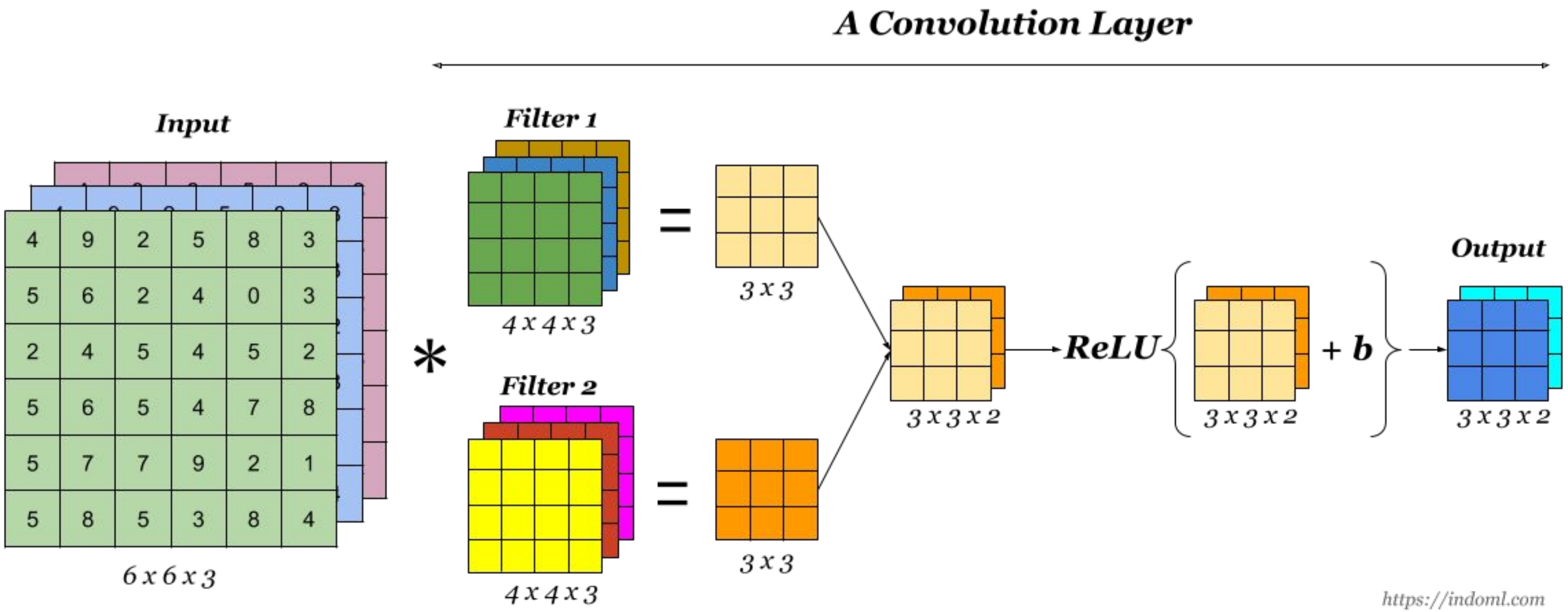


Figure Source

Pooling Layer

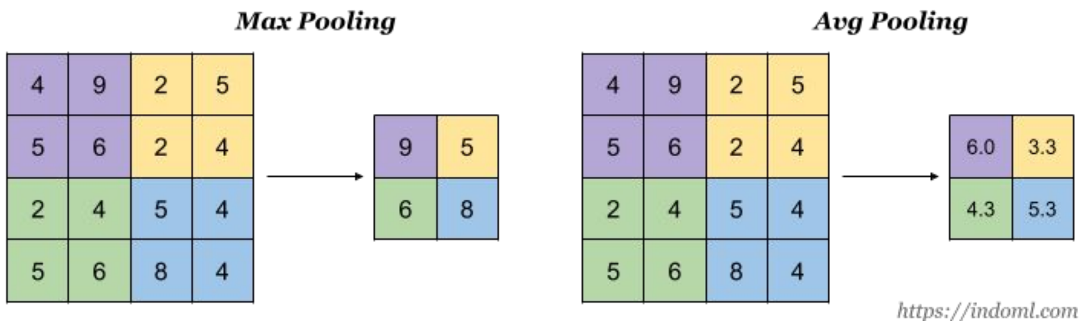
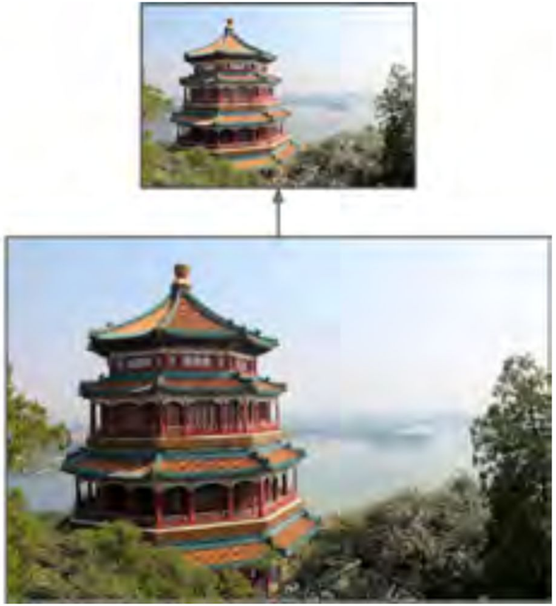


Figure Source



Figures from Aurélien Géron's 1st Ed. Book

- Assuming downsampling will not lose the major information.

Architecture of Convolutional Neural Networks

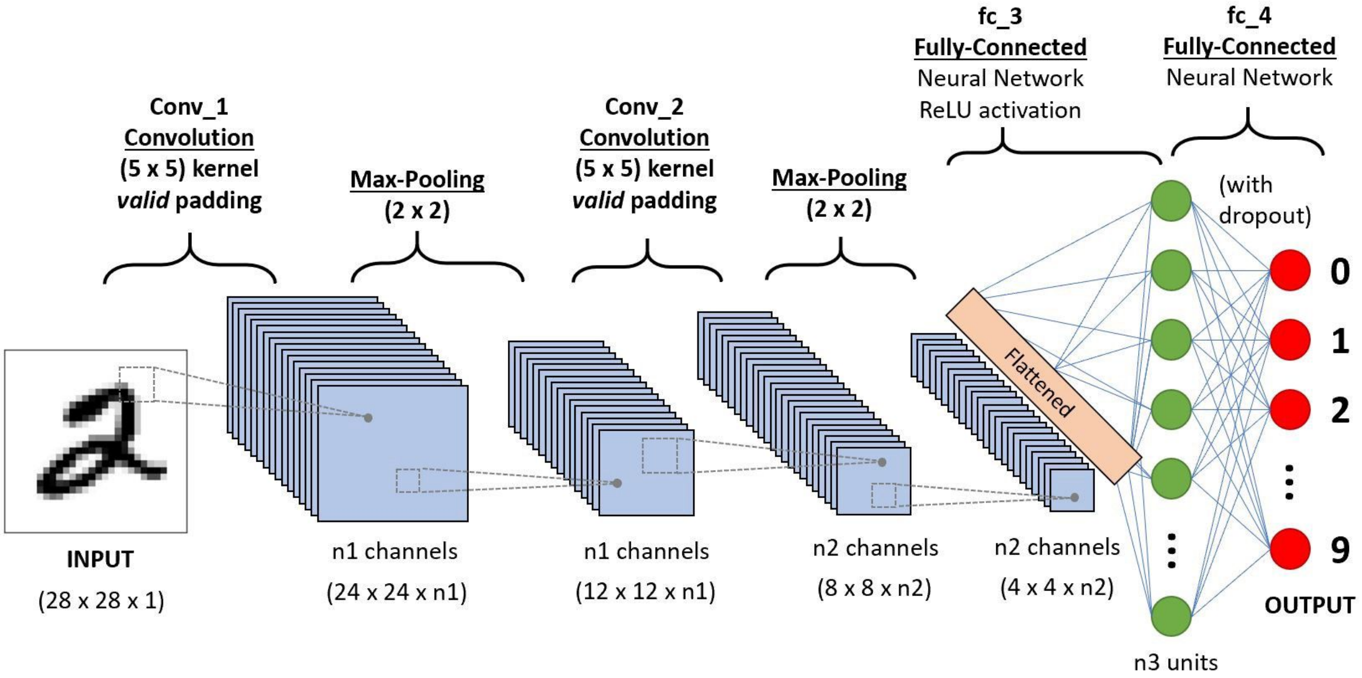


Figure Source

In this talk

Image Data

- Popular image dataset
- Image preprocessing

01

CNN Basics

- Origins & ideas
- CNN mechanism

02

CNN Variants

- Conv as feature extractor
- De-conv, 3d conv, …
- FCN and GCN

03

Demo

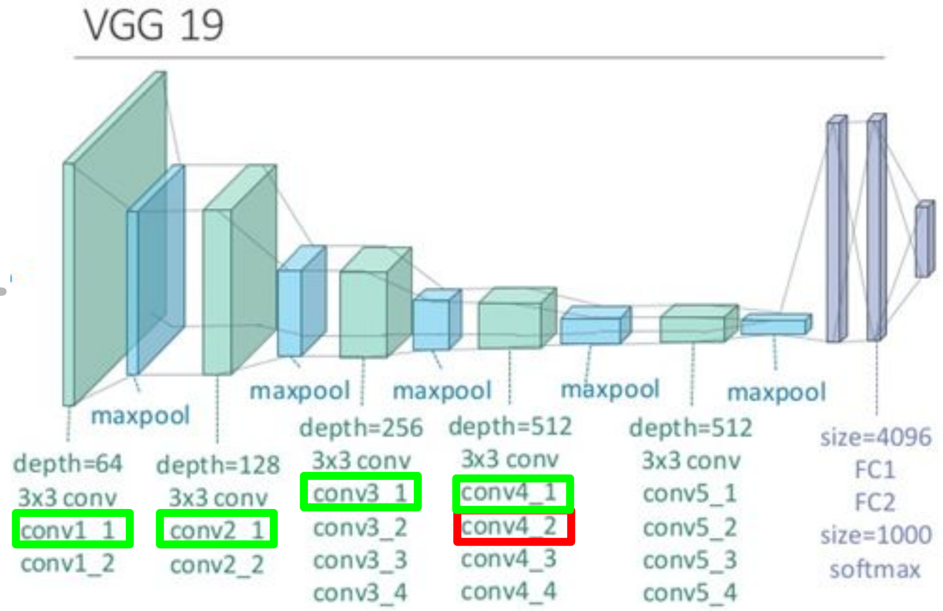
- CNN in pytorch
- Save/load pytorch model
- Setup of colab env

04

Convolutional layers as feature extractors

Content representation of a photograph

Style representation of the artwork



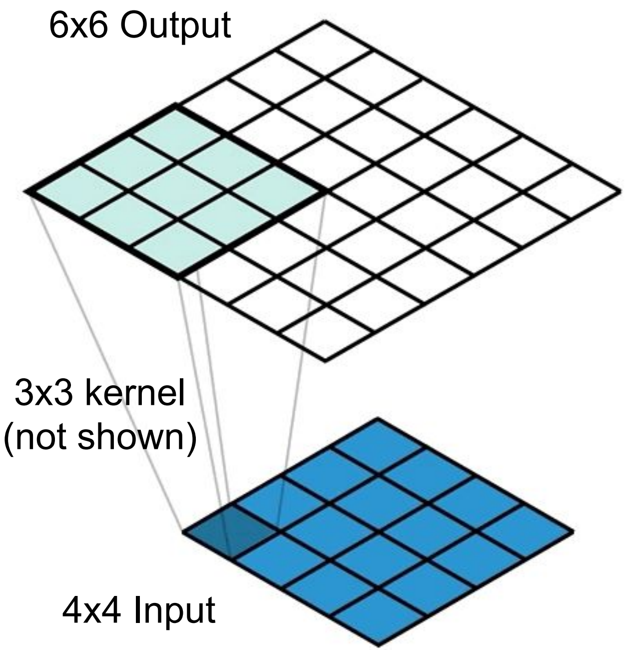
[Style Transfer Paper (2016)]

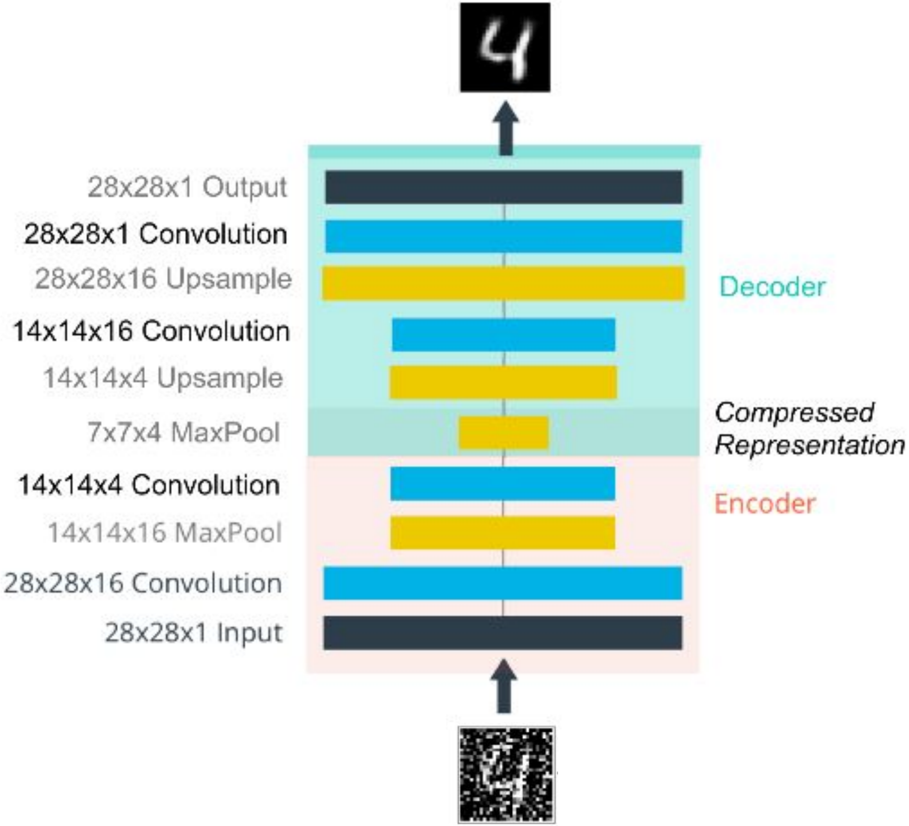
Style transferred art image

Try it by yourself using [Lucent]!

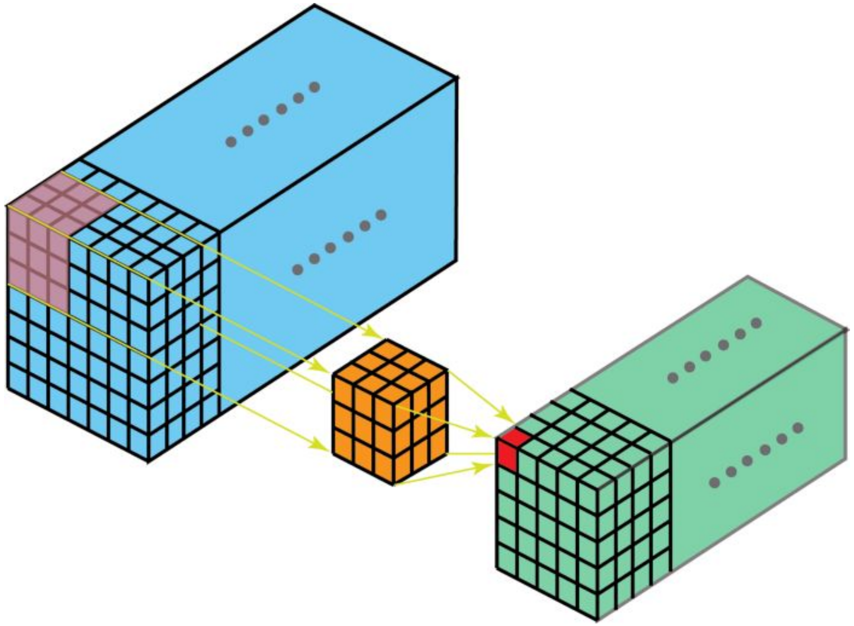
Qiyang Hu

Convolution and “Deconvolution”: Autoencoder

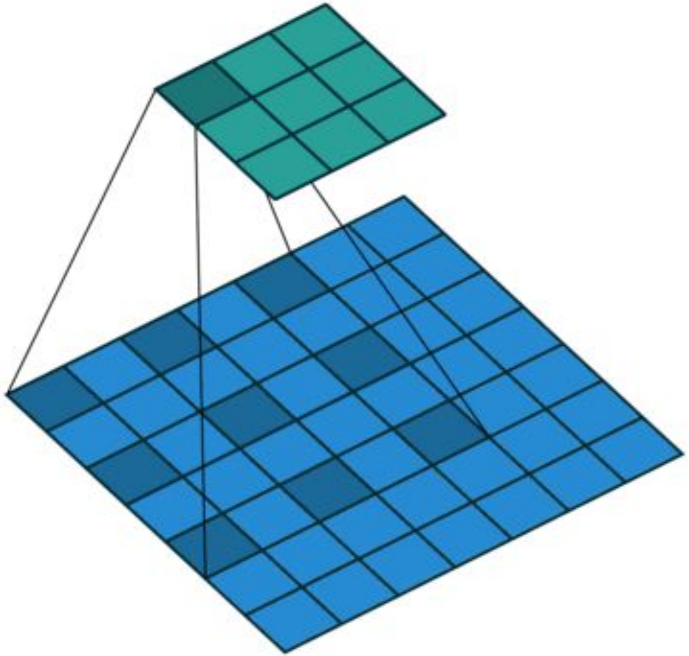




3D Convolutions

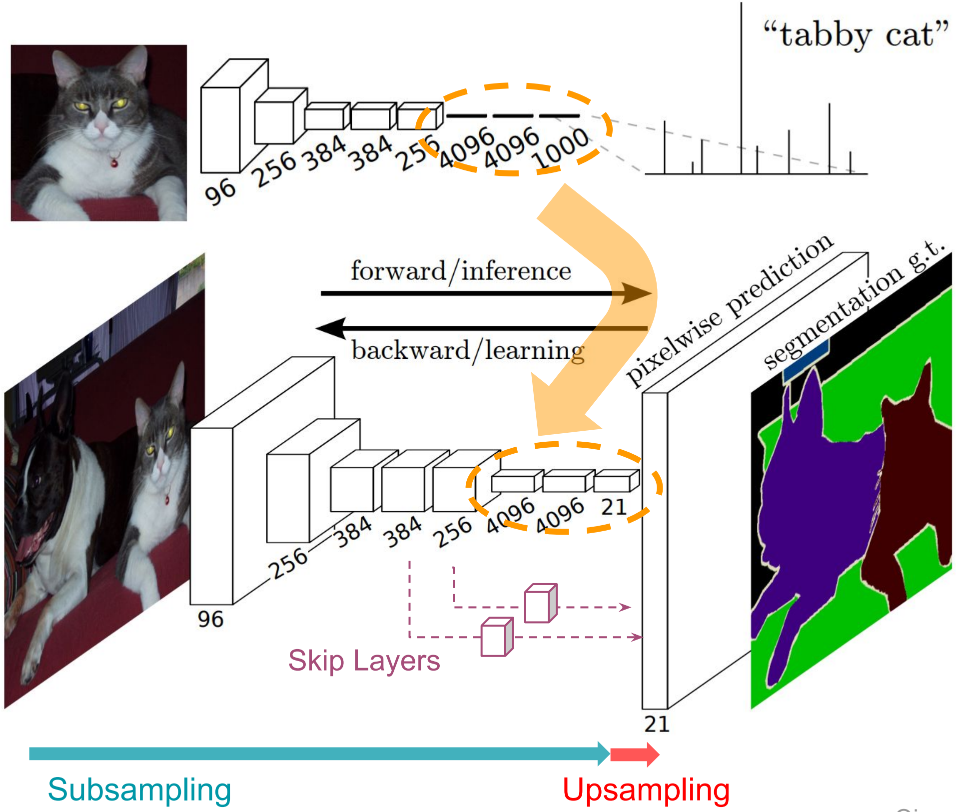


Dilated Convolutions



Fully Convolutional Neural Networks (FCNs)

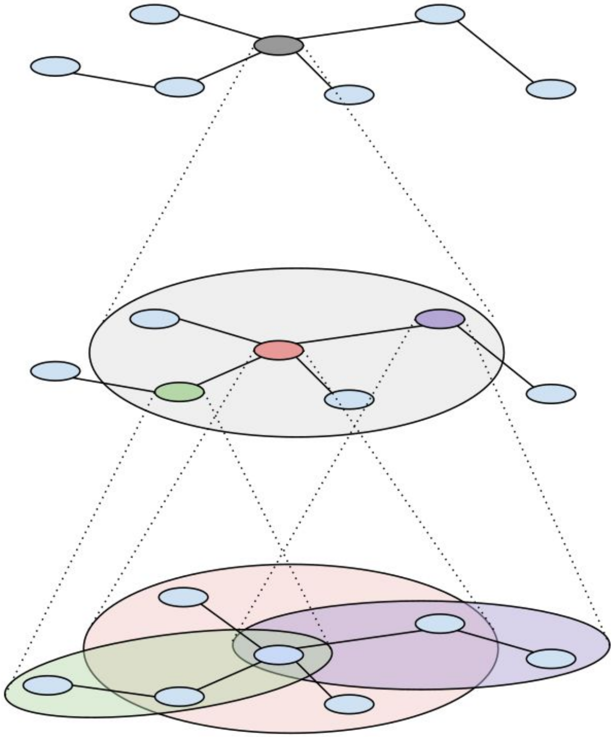
- From image classification to semantic segmentation
  - Per-pixel classifications
  - CNN’s fully connection layers:
    - throw away spatial coordinate
    - ~ applying an img-size kernel
- Ideas in [FCNs]
  - Convolutionalization
  - Upsampling by deconvolution
  - Skip layers
- Similar ideas and variants:
  - R-FCN, Mask R-FCN, SSD, …



Qiyang Hu

Graph Convolutional Neural Networks

- From images to graphs
  - Images: a special grid graph
    - Vertex: Pixel; Edges: indirectly connected to 4 neighbors
  - Graphs:
    - Embedding the info on both V + E
- Graph Neural Network:
  - Input: (X, A), Latents: (H, A)
  - Predictions over nodes, graph, edges
- Graph Convolutional Neural Network:
  - Update with a symmetric normalisation on Adj Matrix
  - Popularized by Kipf & Welling, ICLR 2017
- MPNNs and GATs



In this talk

Image Data
01
- Popular image dataset
- Image preprocessing

CNN Basics
02
- Origins & ideas
- CNN mechanism

CNN Variants
03
- Conv as feature extractor
- De-conv, 3d conv, …
- FCN and GCN

Demo
04
- CNN in pytorch
- Save/load pytorch model
- Setup of colab env

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)             (36,36,64)

        X = F.relu(self.conv3(X))          (34,34,128)
        X = F.max_pool2d(X, 2)             (17,17,128)

        X = F.relu(self.conv4(X))          (15,15,128)
        X = F.max_pool2d(X, 2)             (7,7,128)

        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
    - Kaggle.com → "Register"

2. Create Kaggle API token and download json file
    - Sign in → Your Profile → "Account" → "Create New API Token"

3. Join the competition → "Join Competition"
    - Dogs-vs-Cats Challenge

Colab Hands-on

bit.ly/**LDL_cnn1**

Questions to think about:

- How can we improve the performance of our CNNs model?
- Should we have to start from the scratch?
- Any guidelines to design a CNN model?
    - Kernel size? Channel number? Layer number?
- What’s the latest development of CNNs?

See you next Friday!

Survey

bit.ly/**survey_cnn1**