Learning Deep Learning with PyTorch
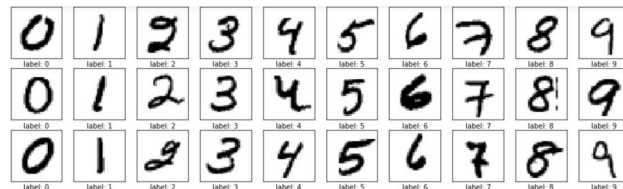
(4) Convolutional Neural Networks

Qiyang Hu
IDRE

"Hello-World" Image Classification Problems
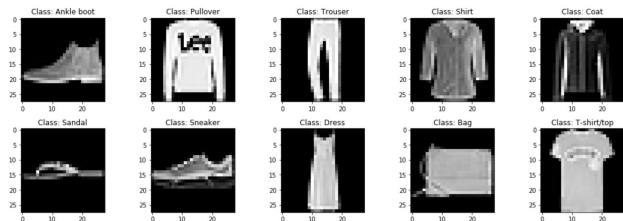
- **MNIST dataset of handwritten digits**
  - 28x28 pixel grey images
  - 10 classes
  - training set: 60,000, test set: 10,000

- **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

Qiyang Hu

Dogs vs. Cats Kaggle Challenge

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

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

- 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

Qiyang Hu

Digitalization for Color Images

Height

Width

B

G

R

88	9	51	123
91	213	13	72
129	67	87	5

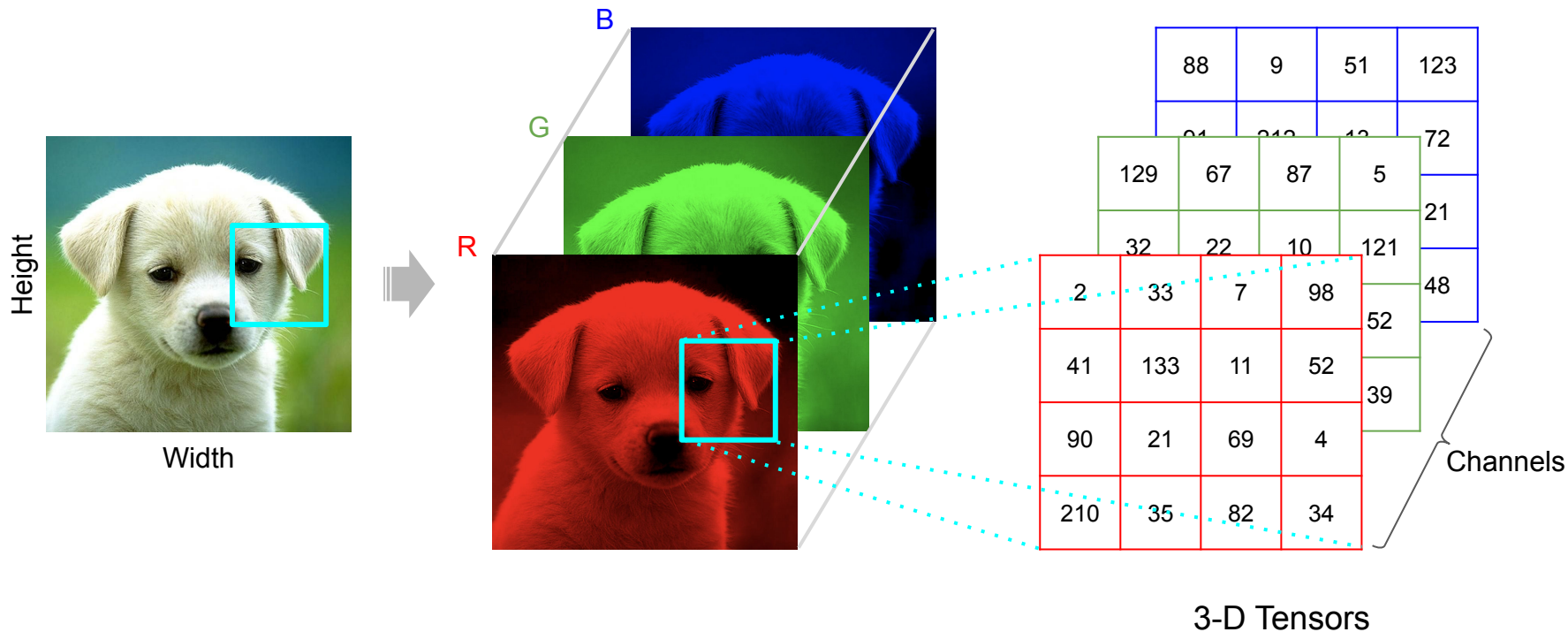| | | | 21 |
32	22	10	121
2	33	7	98	48
				52
41	133	11	52
			39
90	21	69	4
210	35	82	34

Channels

3-D Tensors

Qiyang Hu

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

Qiyang Hu

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
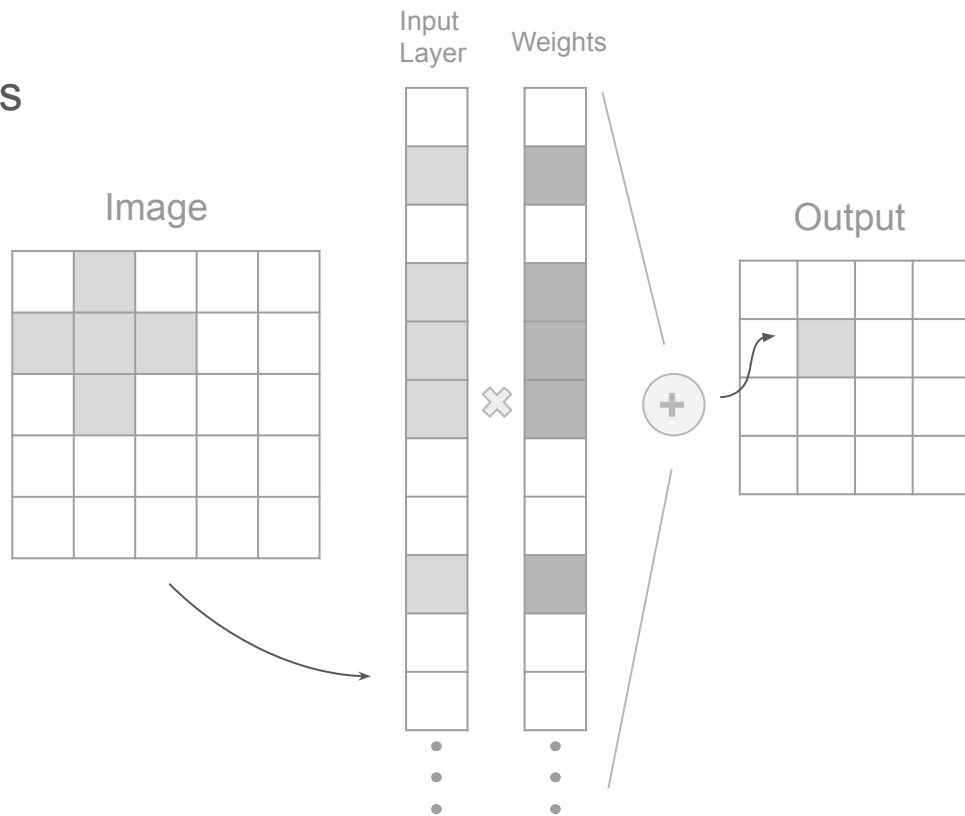 - Loading the data in parallel using multiprocessing workers

Qiyang Hu

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
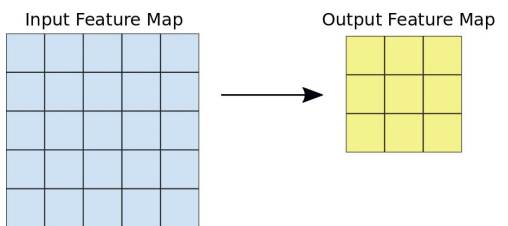   → Hard to fit in a GPU

- Not translation-invariant
  - Shifting 1 pixel
    → Re-learn!

Input
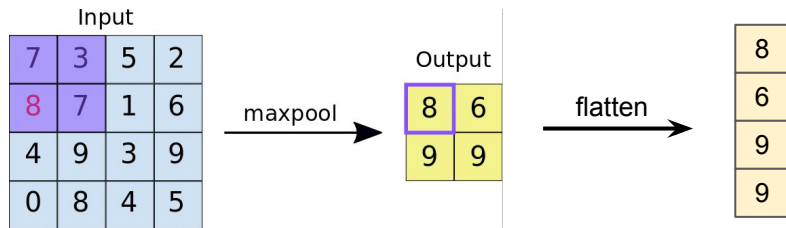Layer

Weights

Image

Output

Qiyang Hu

Convolutional Neural Networks (CNNs)

- A special network architecture

- 3 processes 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

Convolution with horizontal and
vertical strides = 1, with 'same' padding

Qiyang Hu

Multiple Channels

Figure Source

Qiyang Hu

Multiple Filters

A Convolutional layer

Figure [Source](https://indoml.com)

Qiyang Hu

Pooling Layer

Figure Source

https://indoml.com

- Assuming downsampling will not lose the major information.
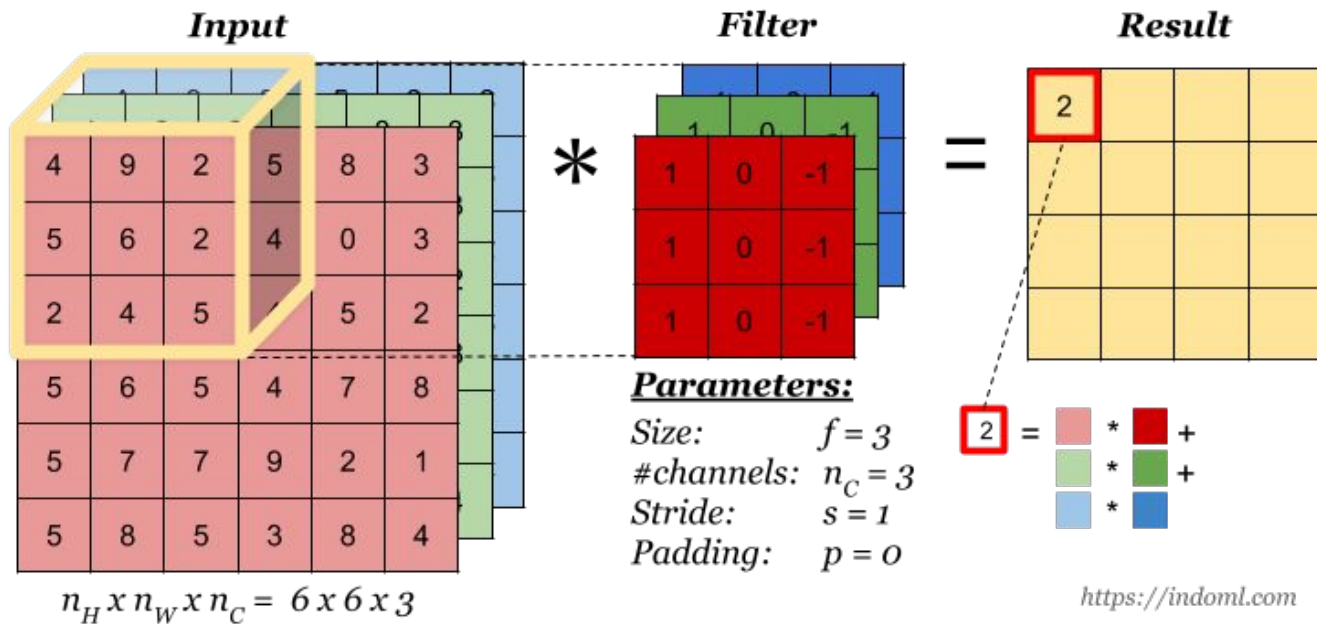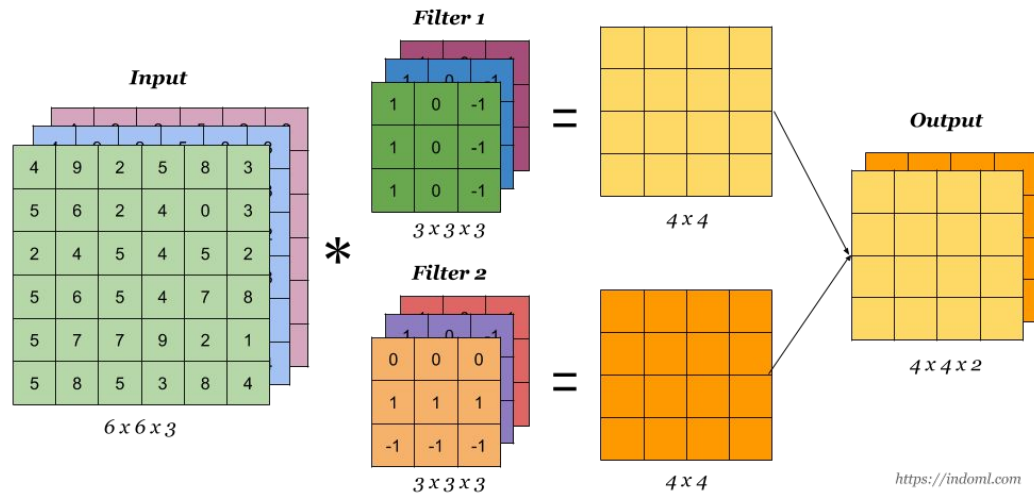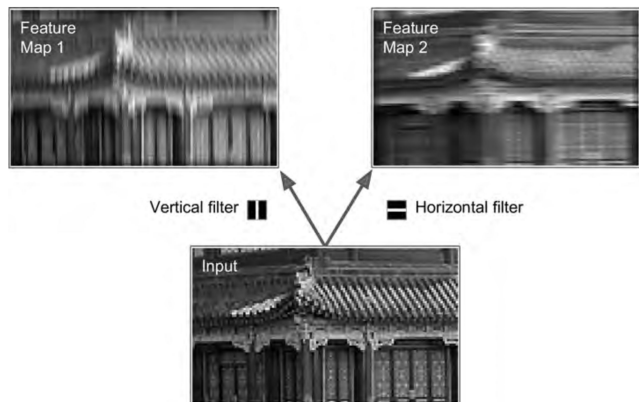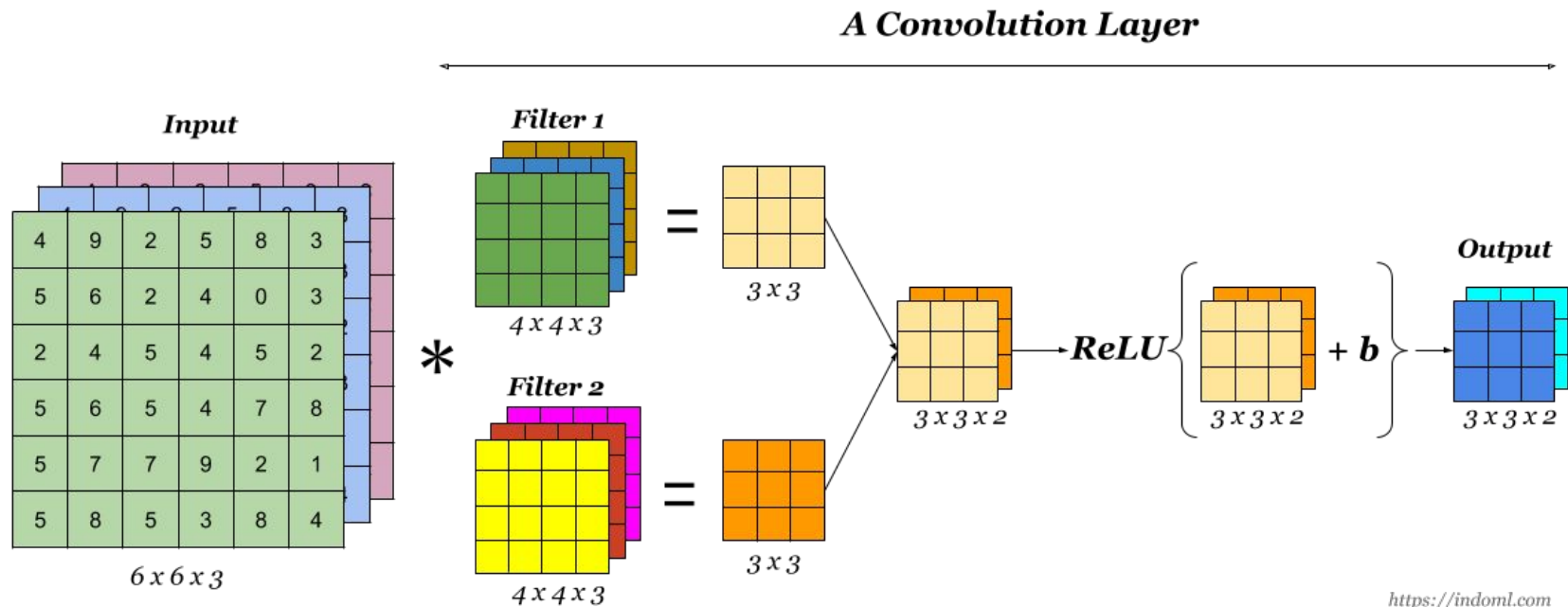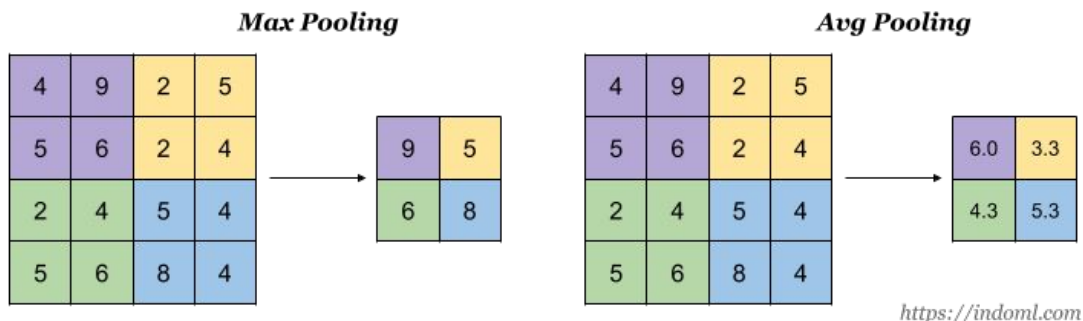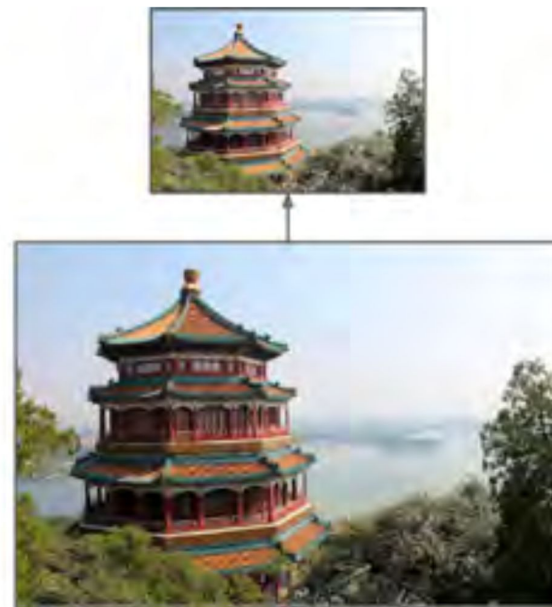
Figures from Aurélien Géron's 1st Ed. Book

Qiyang Hu

Architecture of Convolutional Neural Networks

Figure Source

Qiyang Hu
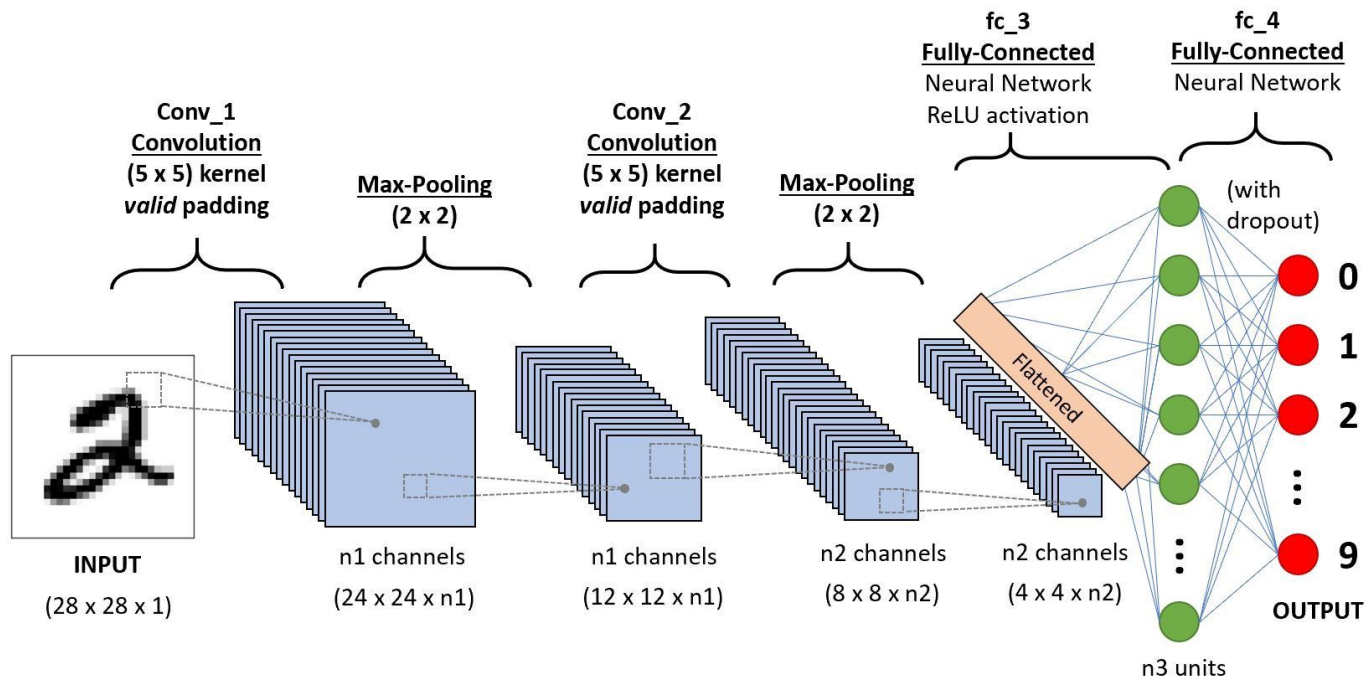
Construct CNN architecture for Dogs-vs.-Cats Problem

- 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'

```python
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))
        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's 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](#))

Qiyang Hu

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 → "My Account" → "Create New API Token"

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

Qiyang Hu

Colab Hands-on

bit.ly/LDL_02

Qiyang Hu

Don't forget to

- Sign in your info to the class
  - To get the email notifications

- Contact me for questions or discussions
  - huqy@idre.ucla.edu
  - Office: Math Sci #3330
  - Phone: 310-825-2011