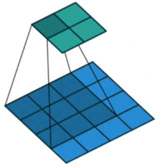
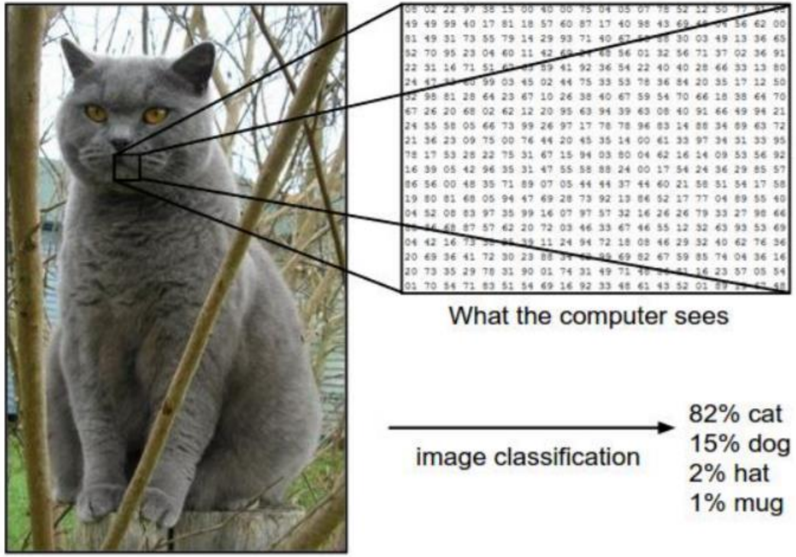
Convolutional Neural Networks

Devashish Khatwani

May 2018

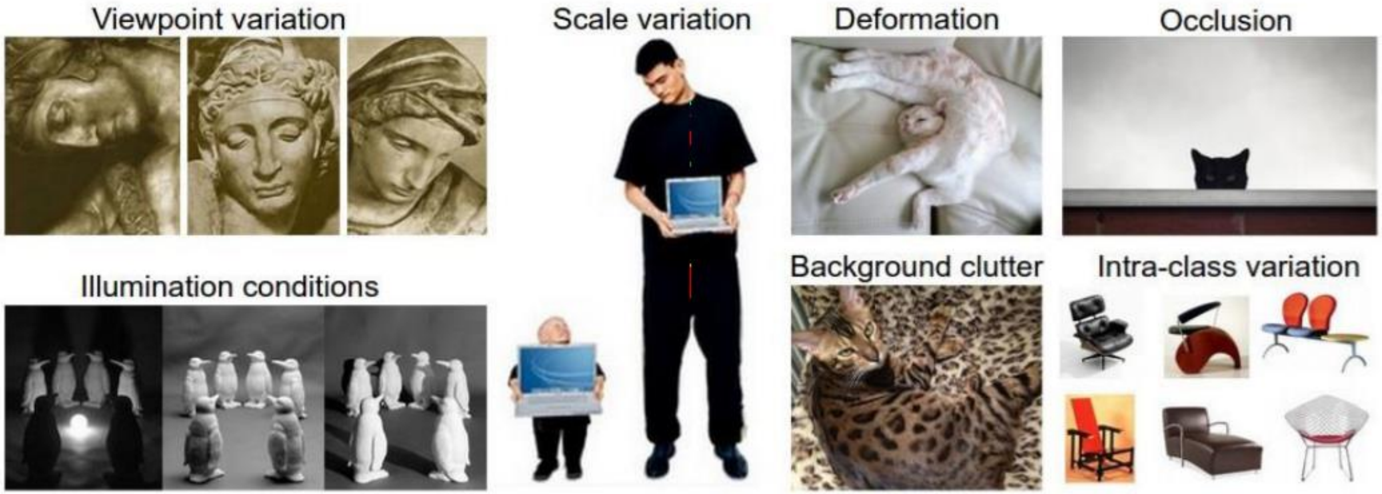


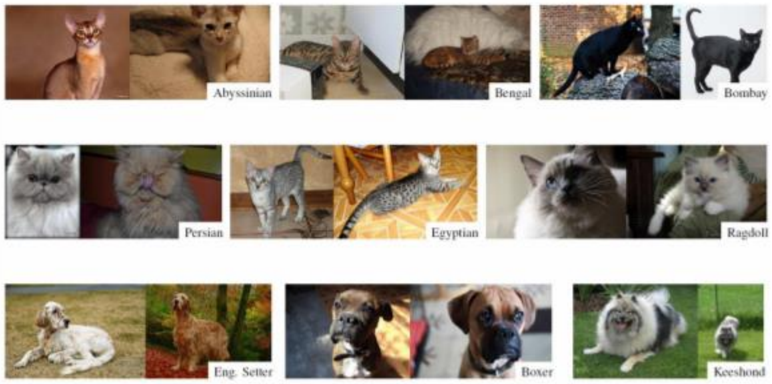
Images are numbers!



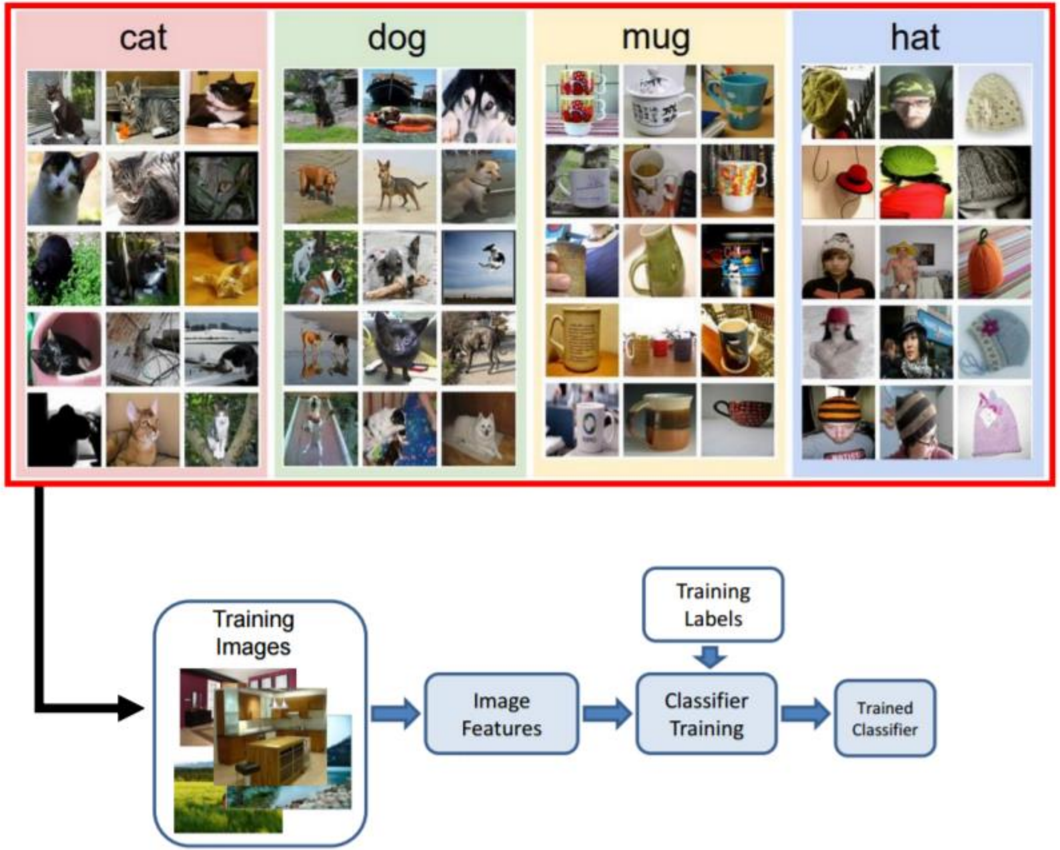
So it should be easy to classify images?

Computer Vision is hard!



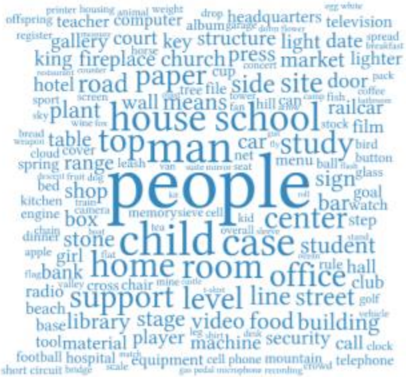


Computer Vision Machine Learning pipeline



Some famous Computer Vision datasets

MNIST: handwritten digits



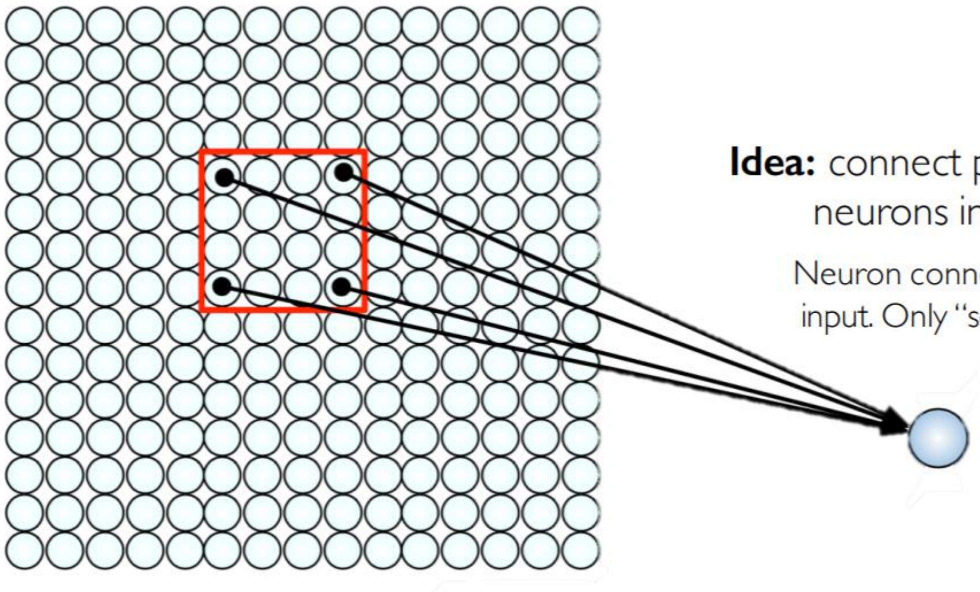
ImageNet: WordNet hierarchy

CIFAR-10(0): tiny images

Places: natural scenes

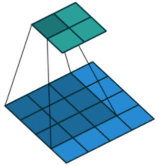
Use Spatial structure of data in an image

Input: 2D image.
Array of pixel values

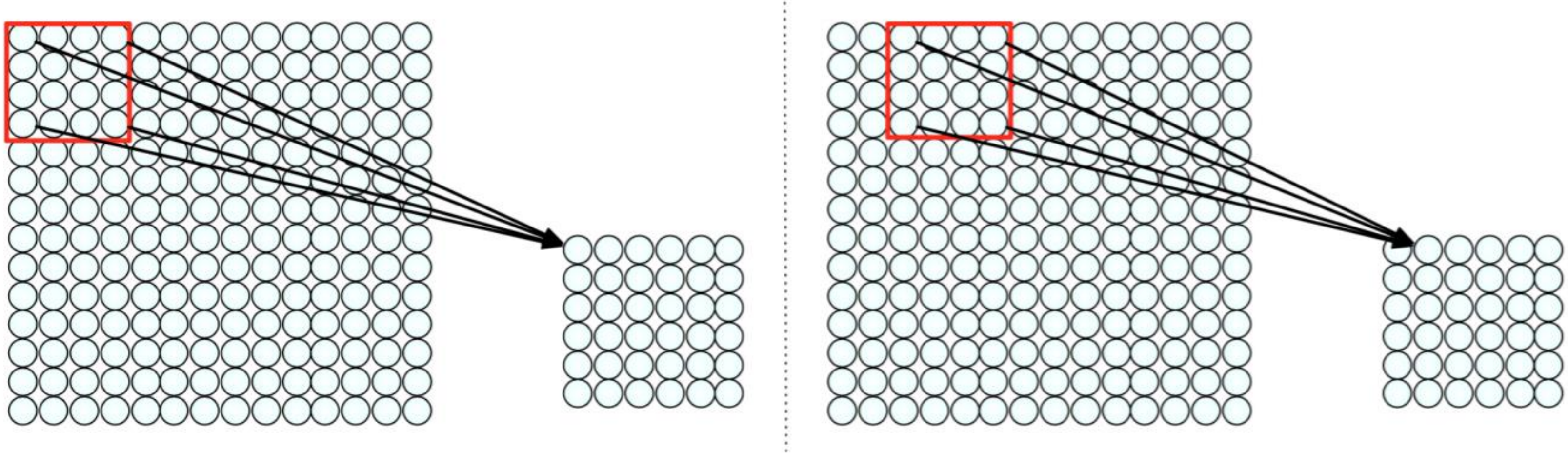


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

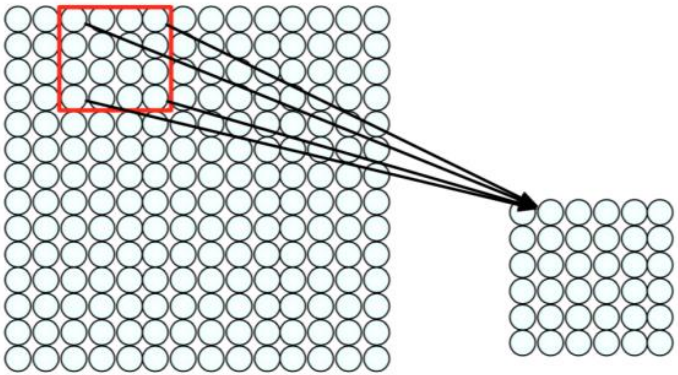


Use Spatial structure of data in an image



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Use Spatial structure of data in an image



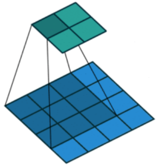
- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter



X or X?

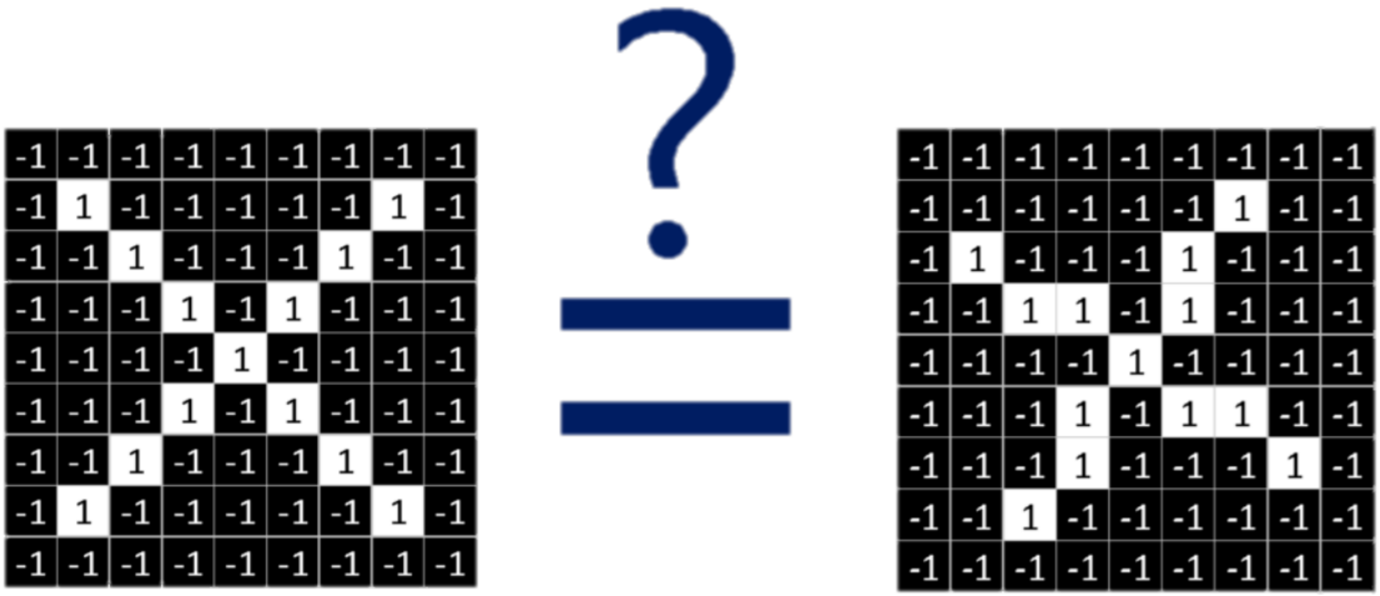
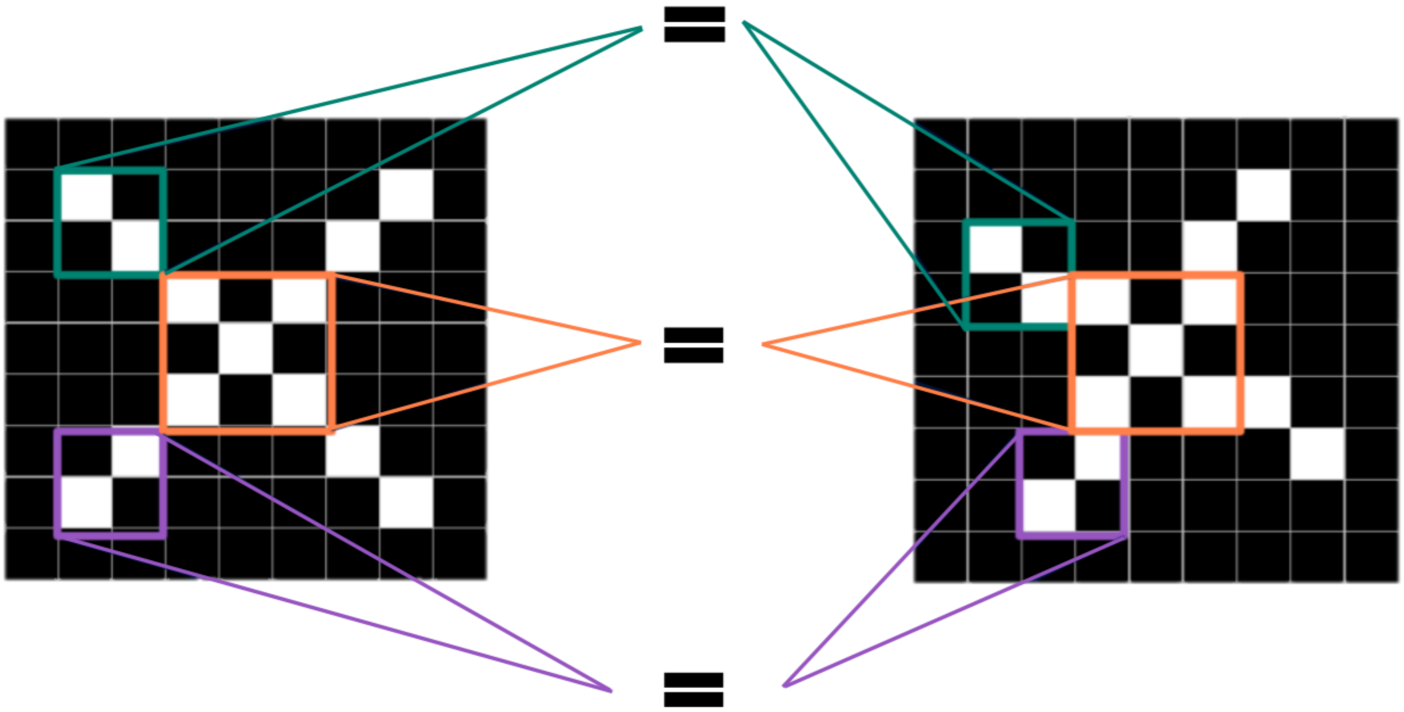
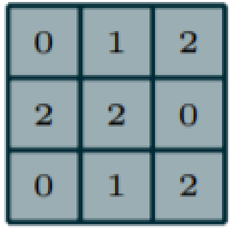


Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

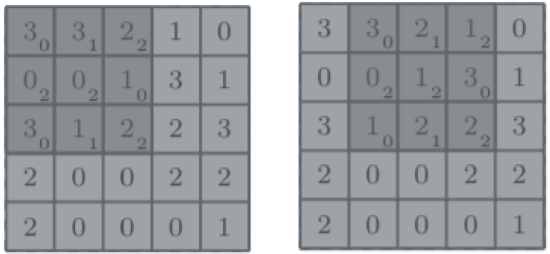
Features of X



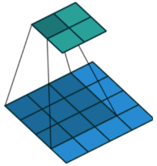
Convolution Operation



Kernel or Filter (3X3)

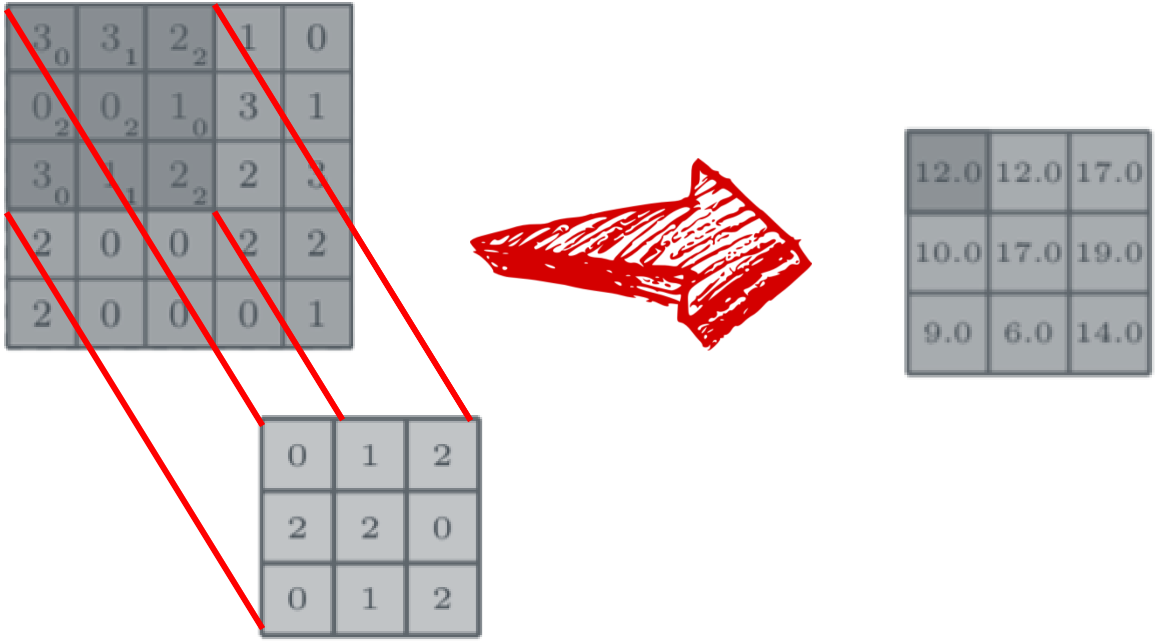


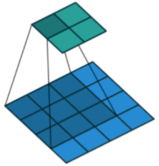
Stride of One



Padding of One

Convolution Operation Contd.





Relationship between input and output dimensions

Input Dimension

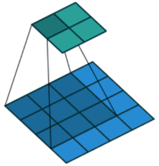
Padding

Filter size

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1.$$

Output Dimension

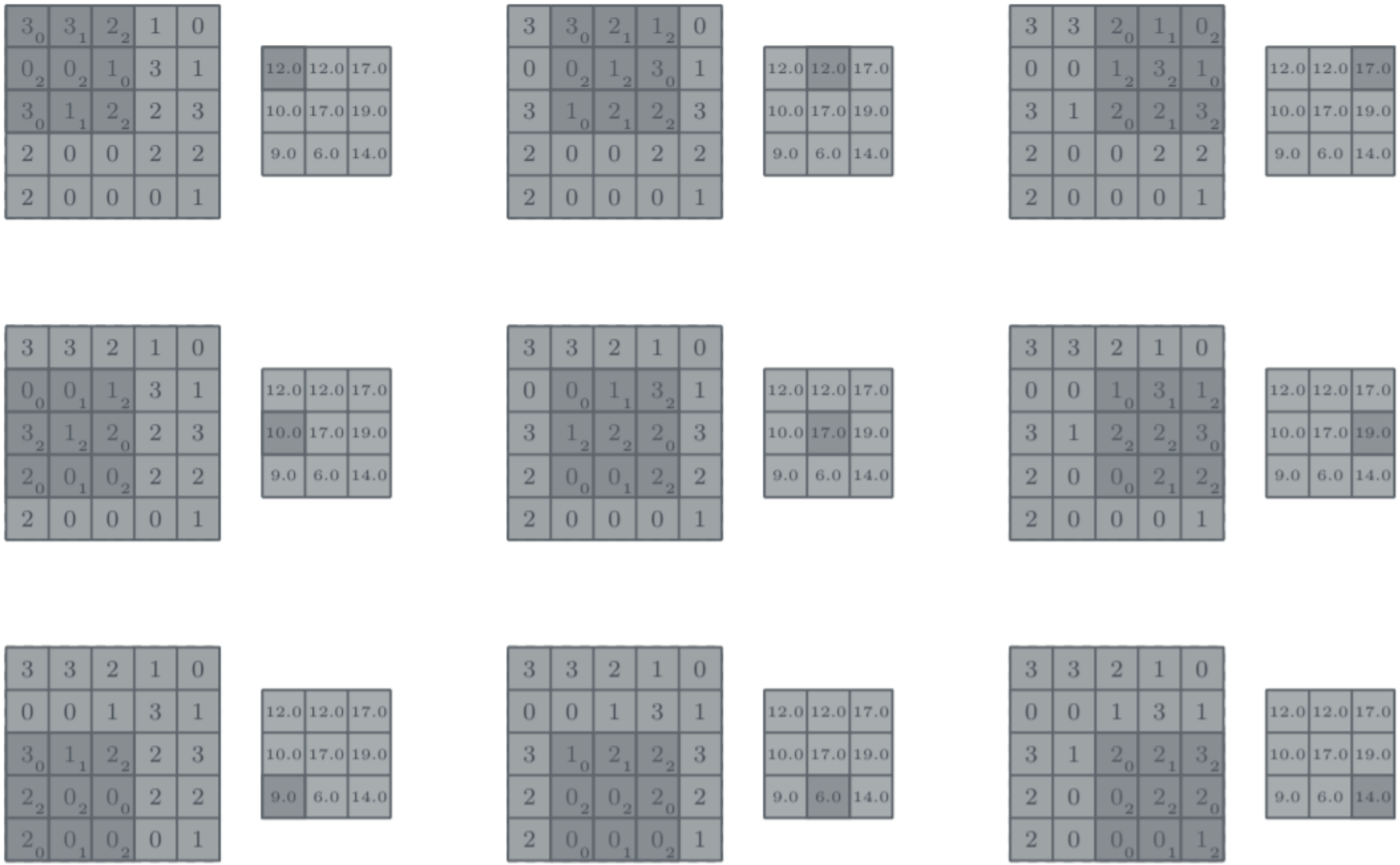
Stride

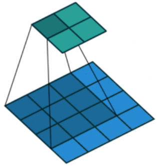


Convolution Operation Contd.

Filter?
Stride?
Padding?

Filter = 3x3
Stride = 1
Padding = 0

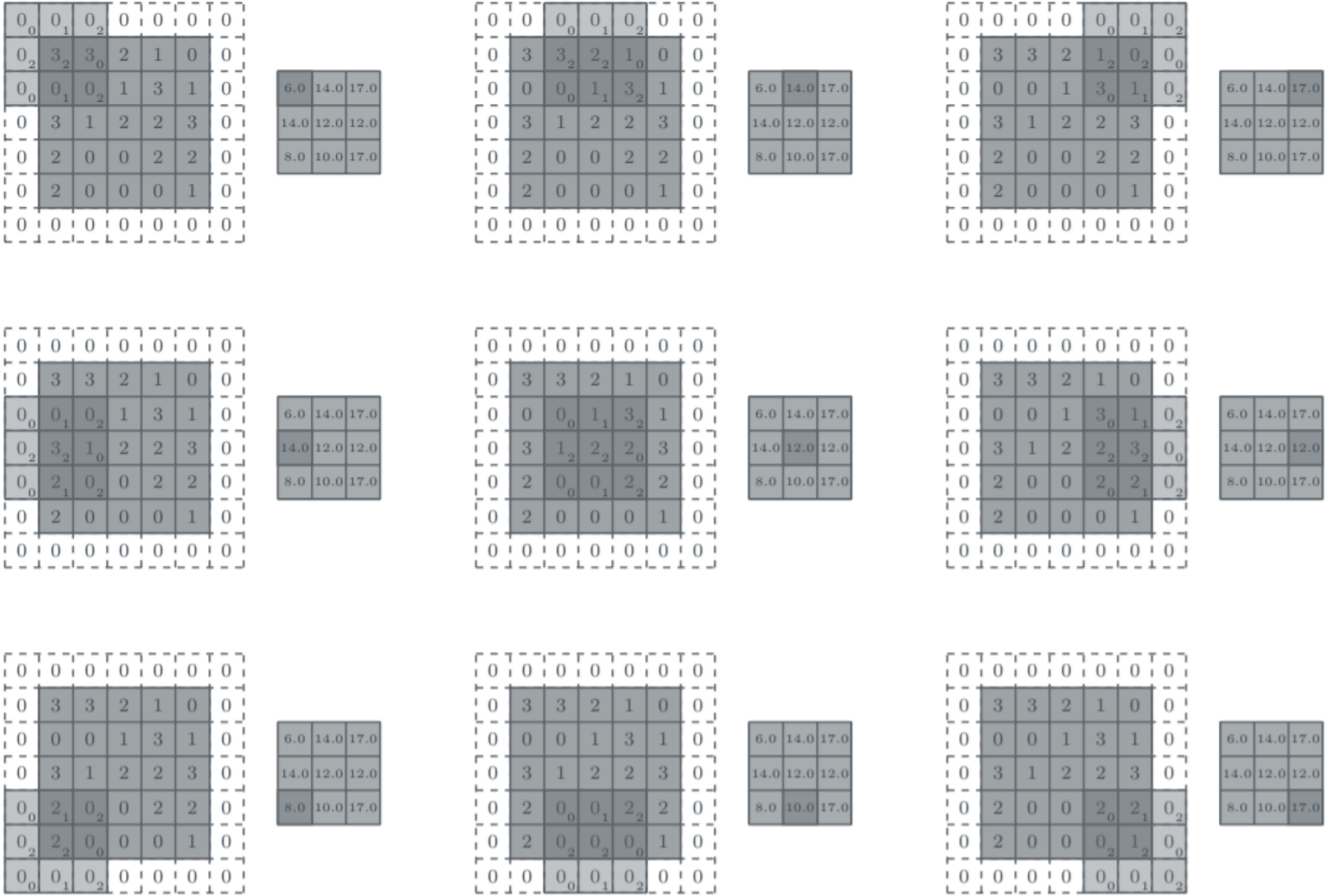


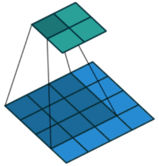


Convolution Operation Contd.

Filter?
Stride?
Padding?

Filter = 3x3
Stride = 1
Padding = 1

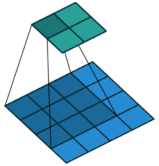


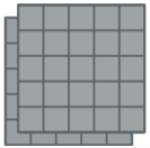


Relationship between Filter Channels, Input Channels and Output Channels

Filter = 3x3
Stride = 1
Padding = 0
Input Channels = 2
Filter Channels = 3
Output Channels = 3

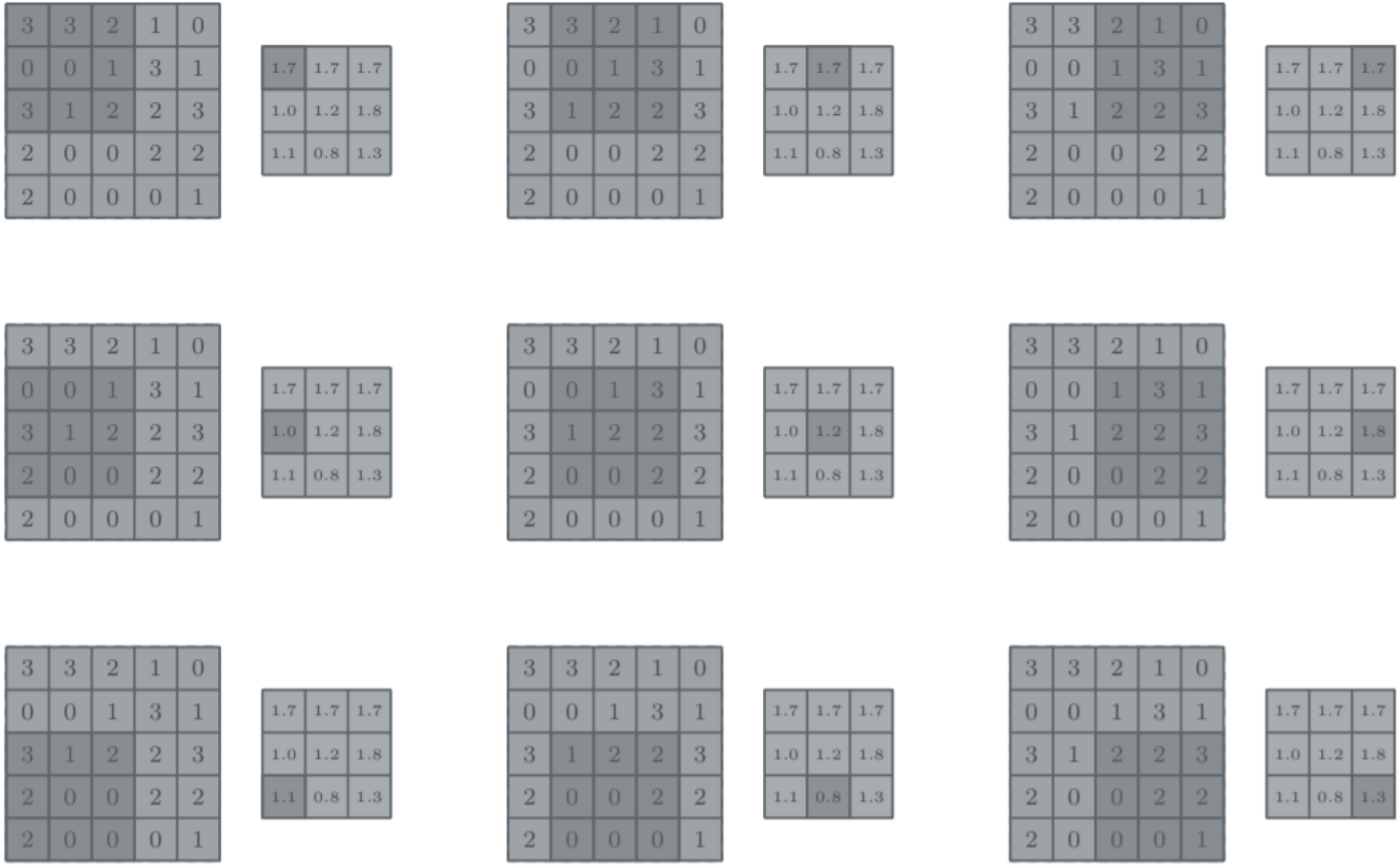
Filter 1 Filter 2 Filter 3

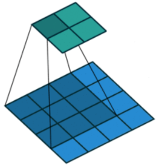




Pooling Operation

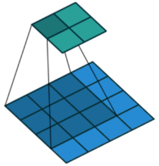
Average Pooling





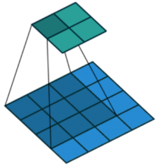
Pooling Operation Contd.

Max Pooling

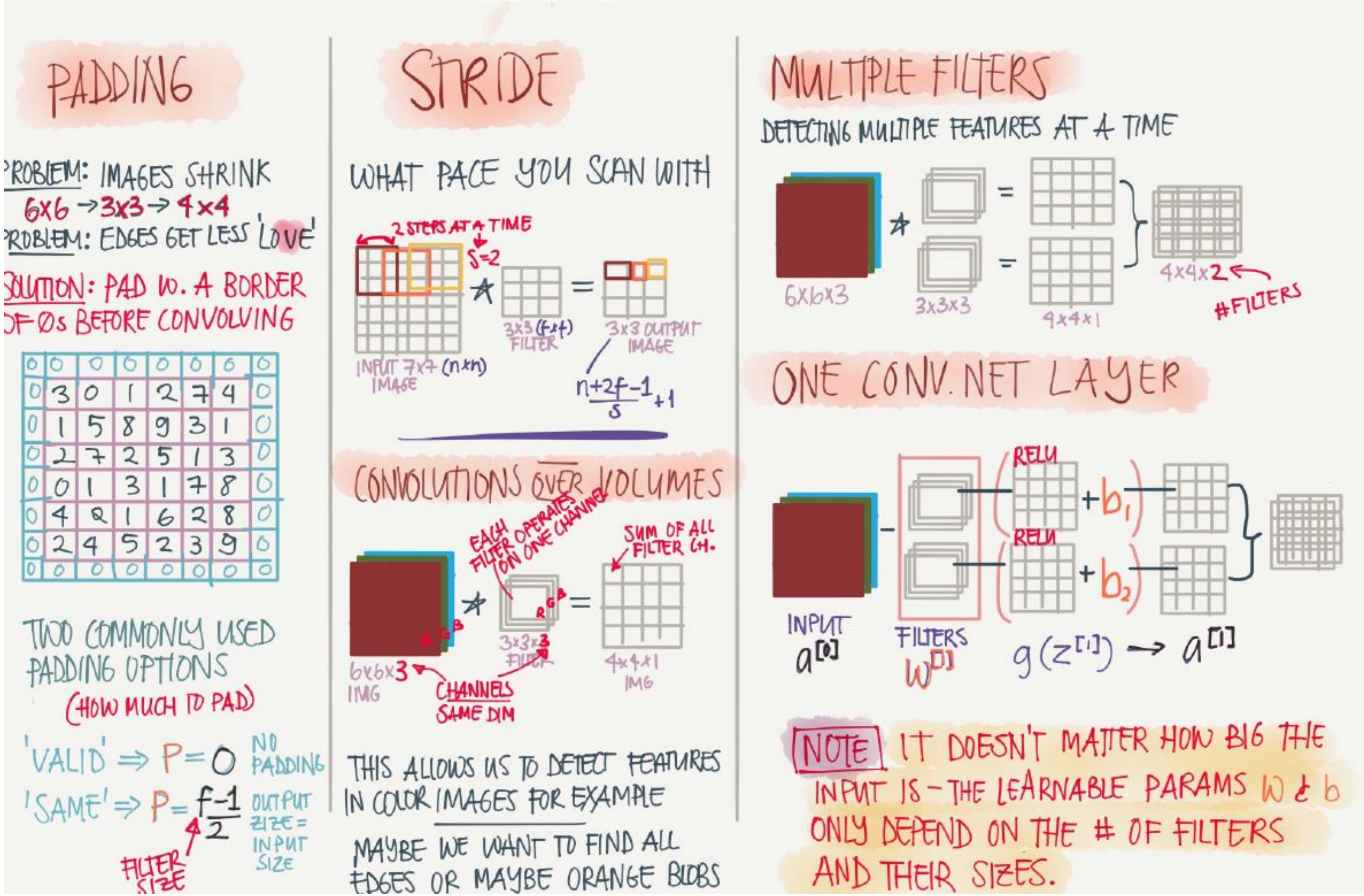


Pooling Operation Contd.

Max
Pooling

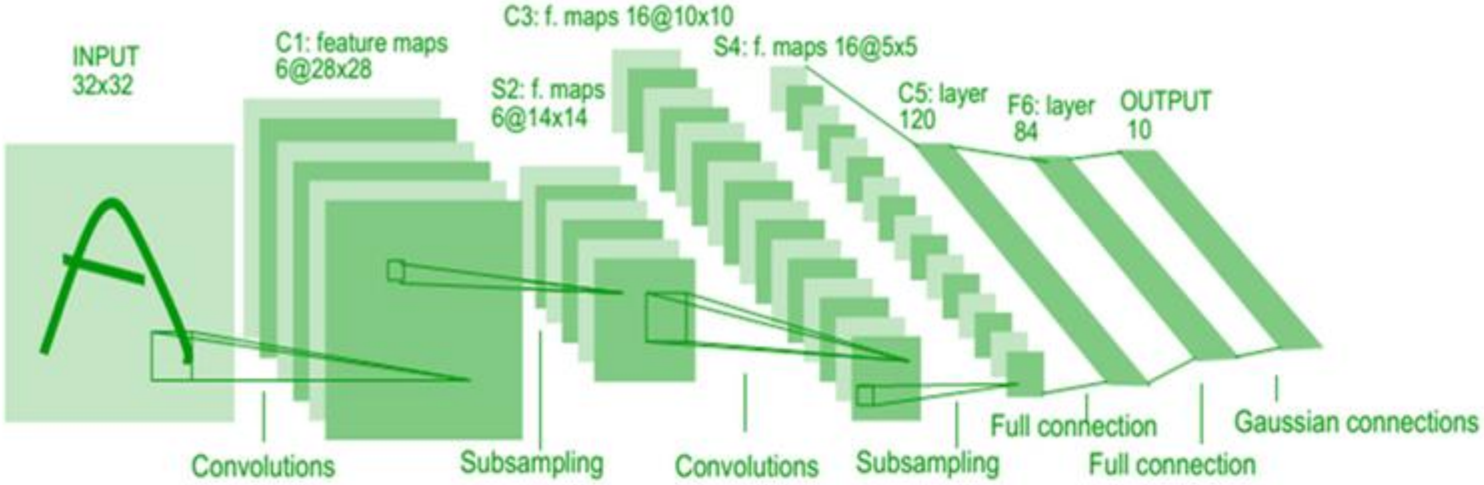


Putting it all together

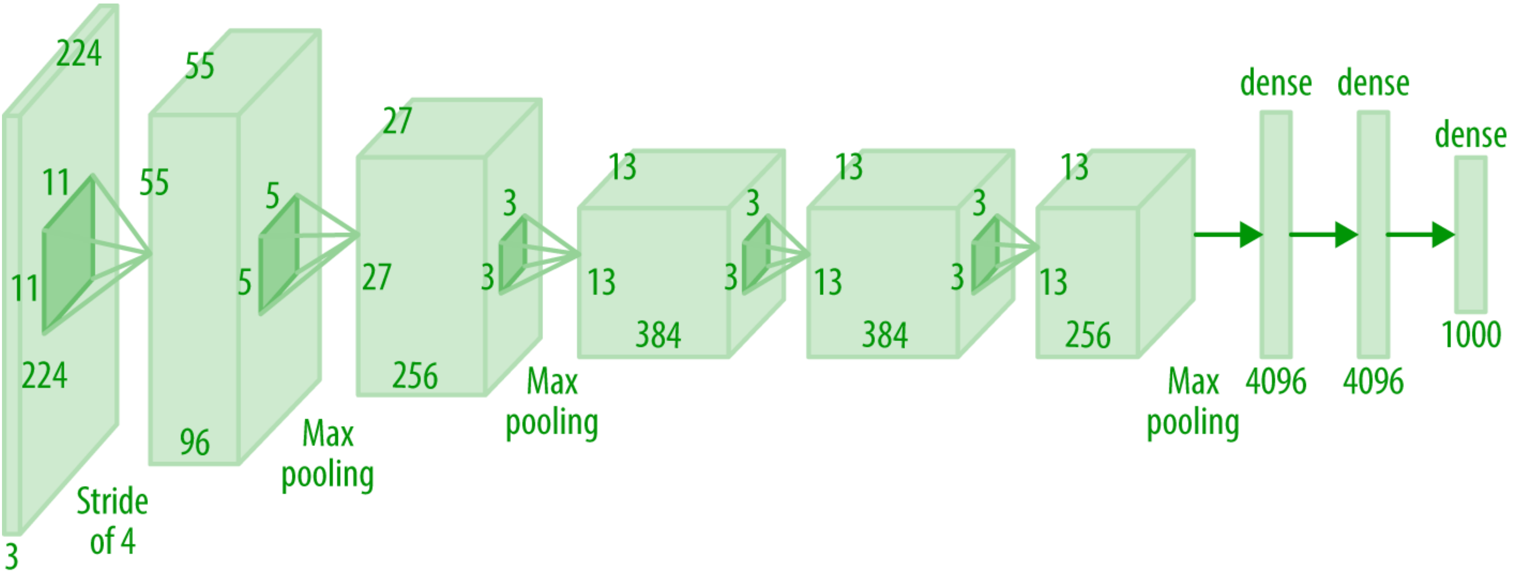


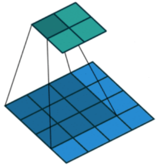
MNIST Classifier using Pytorch

Commonly used CNN Architectures LeNet

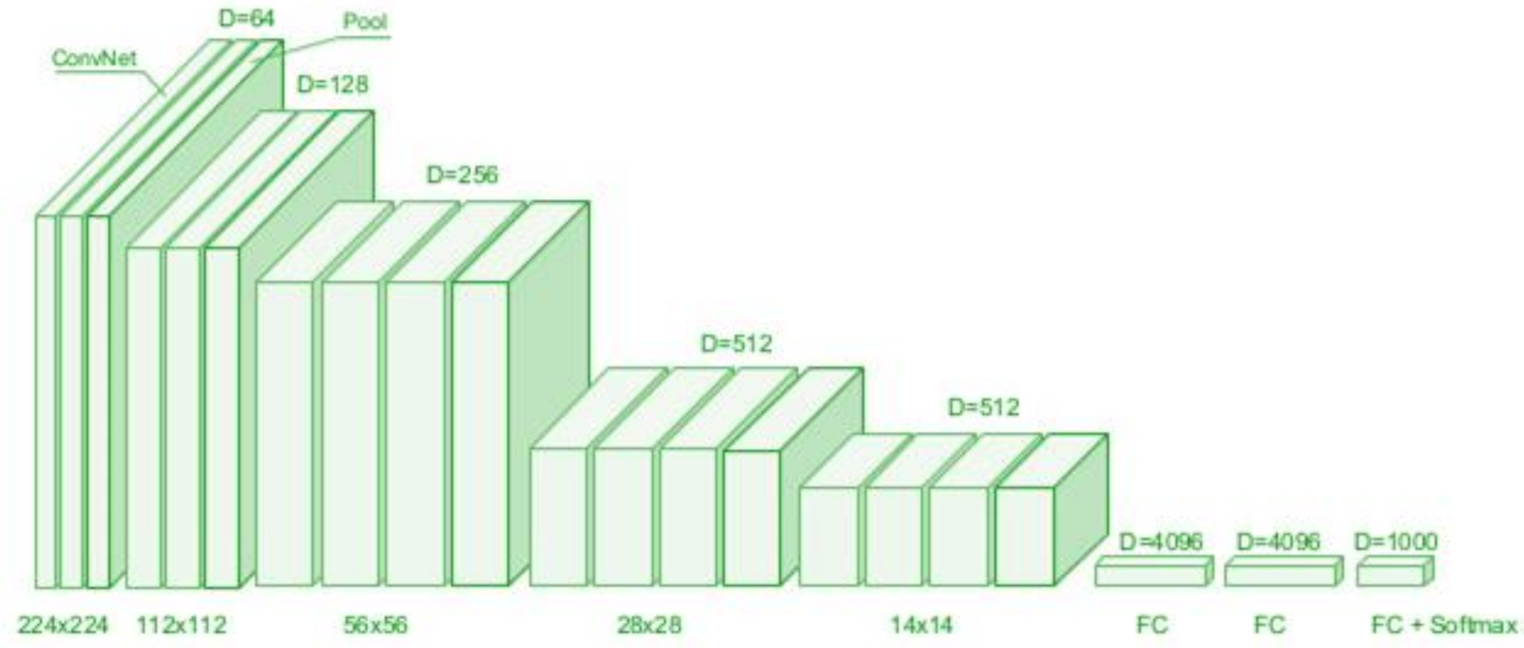


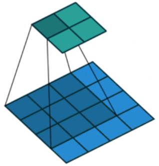
Commonly used CNN Architectures AlexNet





Commonly used CNN Architectures VGG16





What comes after VGG16?

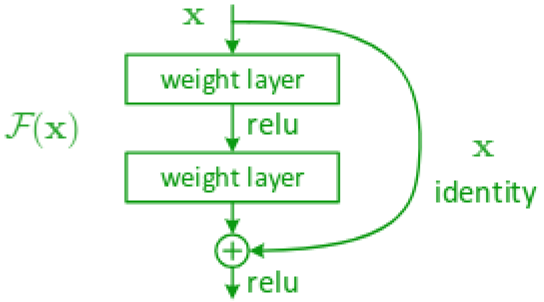
Just adding more layer doesn't work!

- Vanishing Gradient
- Solvers are not able to get to an identity mapping

Need a better way of propagating information within the network

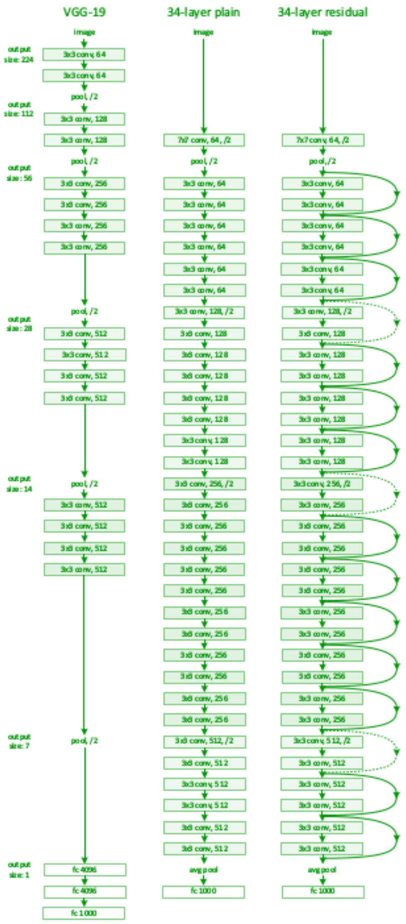
- Residual connections
- Highway networks
- Dense connections

Commonly used CNN Architectures Resnet50

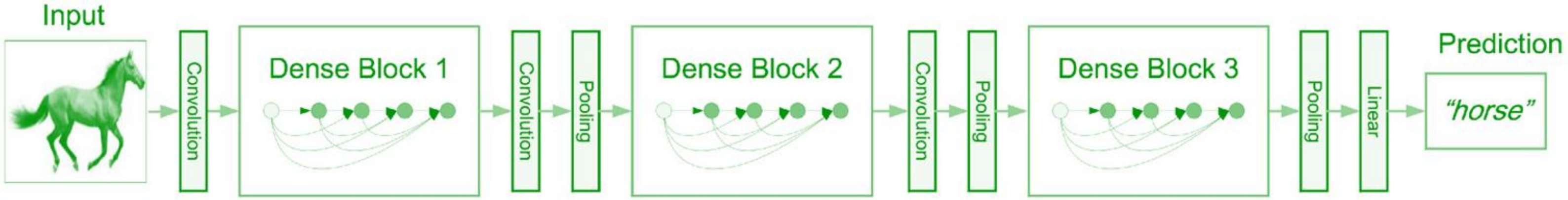


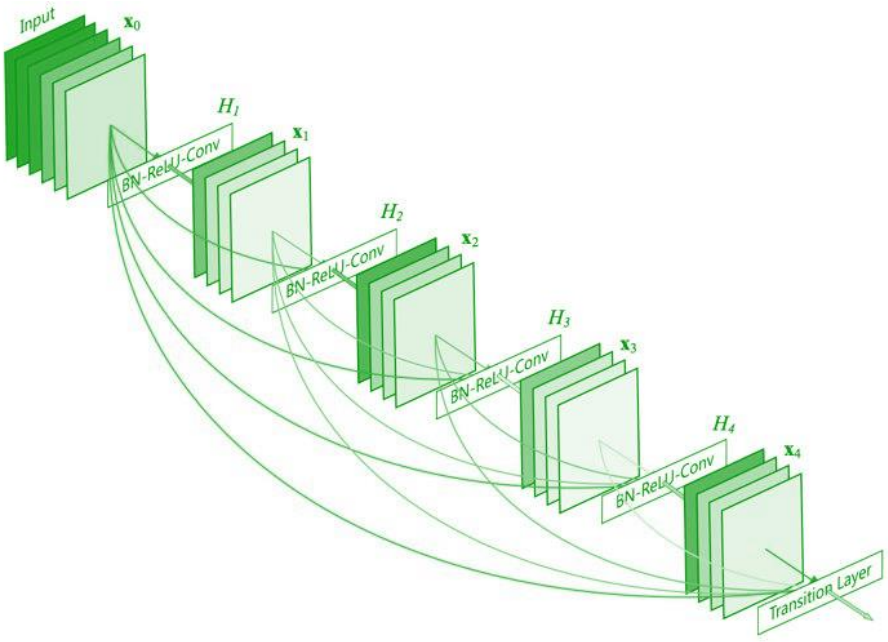
Residual learning: a building block.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

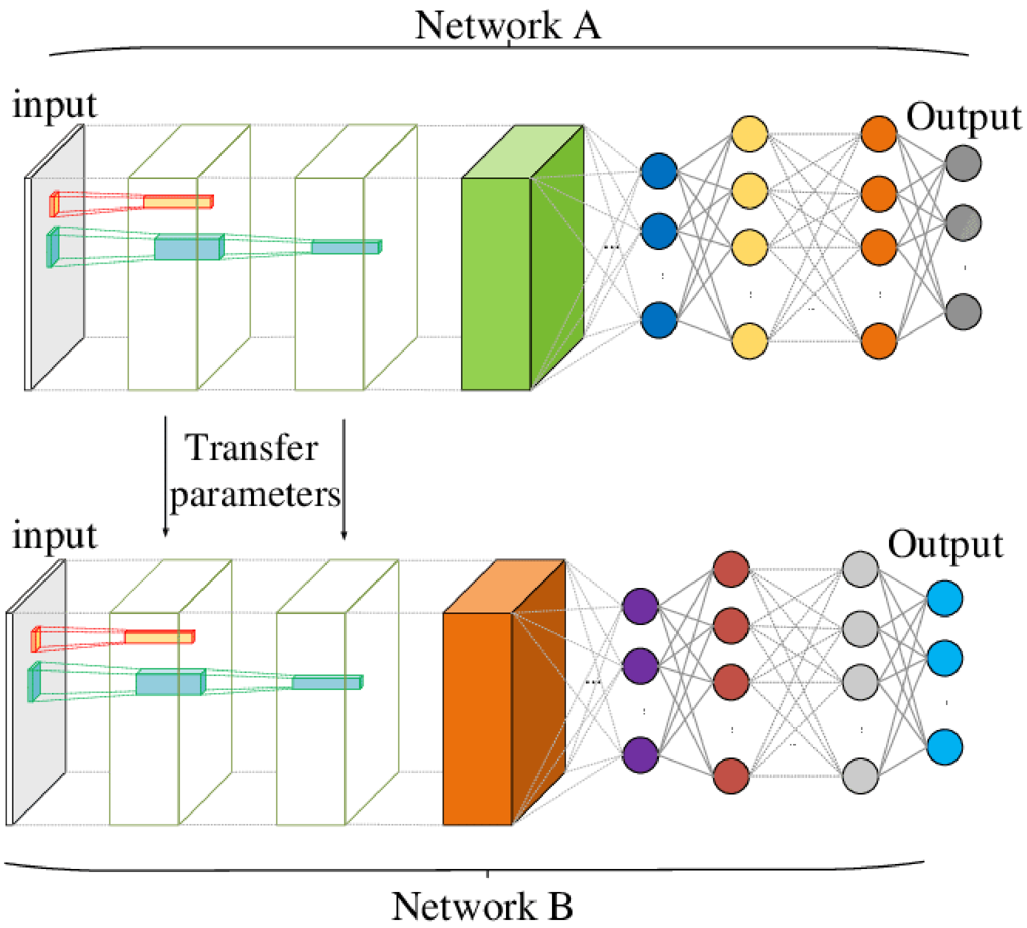


Commonly used CNN Architectures Densenet





Transfer Learning



Transfer Learning with Pytorch and Resnet18

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

plt.ion()   # interactive mode
```

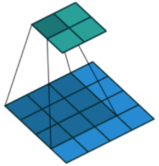
Transfer Learning with Pytorch and Resnet18

Transforms are common image transforms. They can be chained together using `Compose`

```
# Data augmentation and normalization for training
# Just normalization for validation
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

Data loader. Combines a dataset and a sampler, and provides single- or multi-process iterators over the dataset.

```
data_dir='c:\\Users\\khatwd2\\Documents\\Python Scripts2\\data\\hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes
device = 'cpu'
```



Transfer Learning with Pytorch and Resnet18

Define a function to train the model

Load the data into the model for training

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                scheduler.step()
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs
                labels = labels

                # zero the parameter gradients
                optimizer.zero_grad()
```

Transfer Learning with Pytorch and Resnet18

Train step

```
# forward
# track history if only in train
with torch.set_grad_enabled(phase == 'train'):
    outputs = model(inputs)
    '''
    Returns the maximum value of each row of the input tensor in the given dim
    The second return value is the index location of each maximum value found
    '''
    _, preds = torch.max(outputs, 1)
    loss = criterion(outputs, labels)

    # backward + optimize only if in training phase
    if phase == 'train':
        loss.backward()
        optimizer.step()
```