Introduction to Modern AI
Week 5: Computer Vision

Gavin Hartnett

PRGS, Winter Quarter 2022

Topics we will cover this week

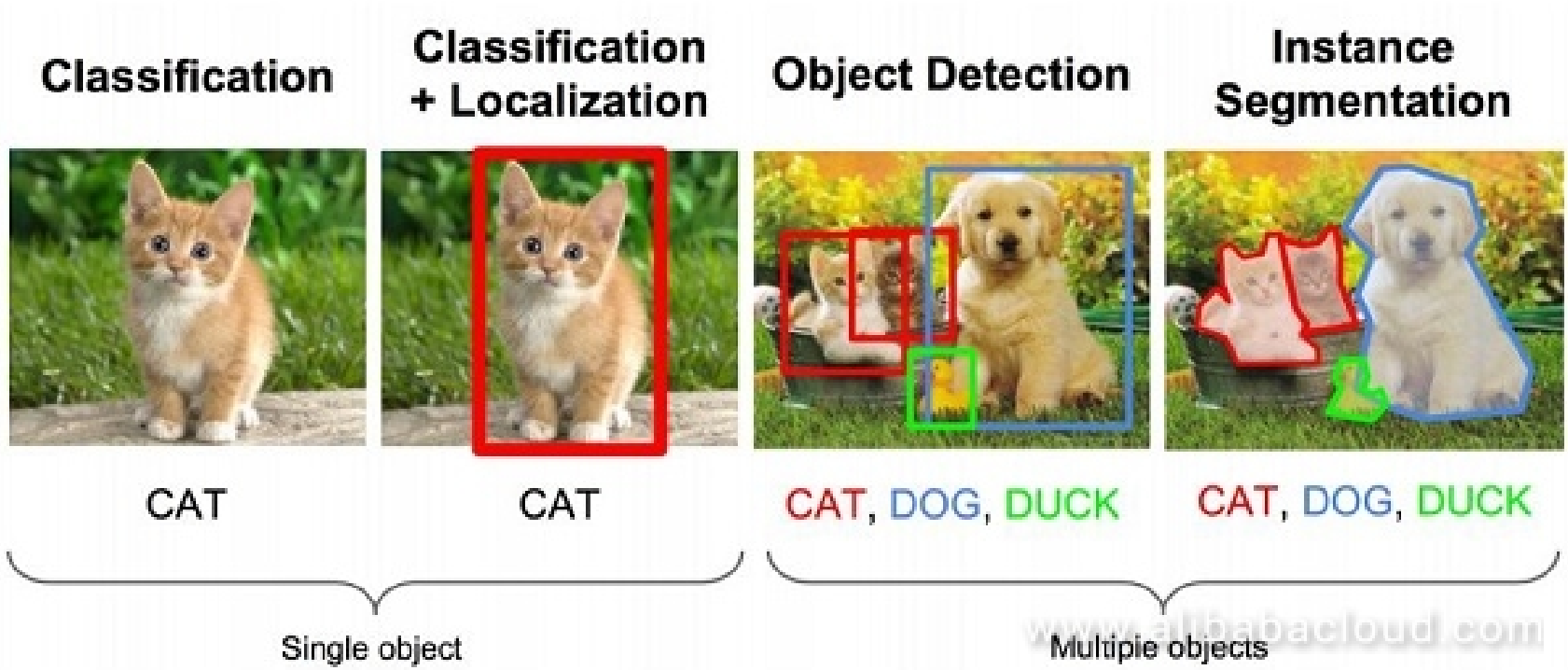
- Digital representation of images
- Important datasets
- Convolutional layers and Convolutional Neural Networks (CNNs)
- Advanced computer vision models

Computer Vision

- Computer Vision (CV) is the subfield of ML concerned with analyzing image data (including videos)
- The first dramatic successes in using deep neural networks to analyze data were made in CV
- Popular and highly successful CV algorithms exist for all 3 ML "pillars", supervised/unsupervised/reinforcement learning

Computer Vision: Supervised Learning

- Common SL problems involve identifying and locating objects of interest
- Field benefits from huge proliferation of images due to the internet and smart-phones



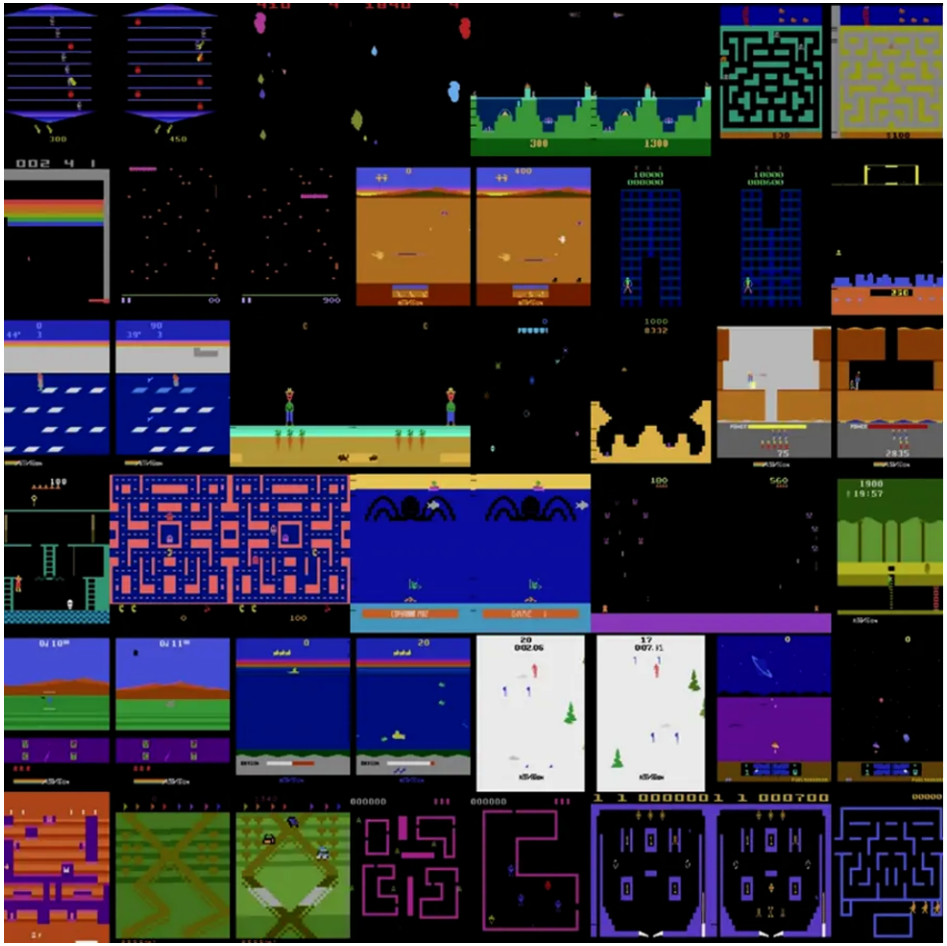
Source: https://www.alibabacloud.com/blog/from-r-cnn-to-faster-r-cnn-the-evolution-of-object-detection-technology_593829

Computer Vision: Unsupervised Learning

Computer Vision: Unsupervised Learning

- Common UL problems include
- Generating fake images
- Modifying existing images
- Training on large unlabeled datasets (pre-training)

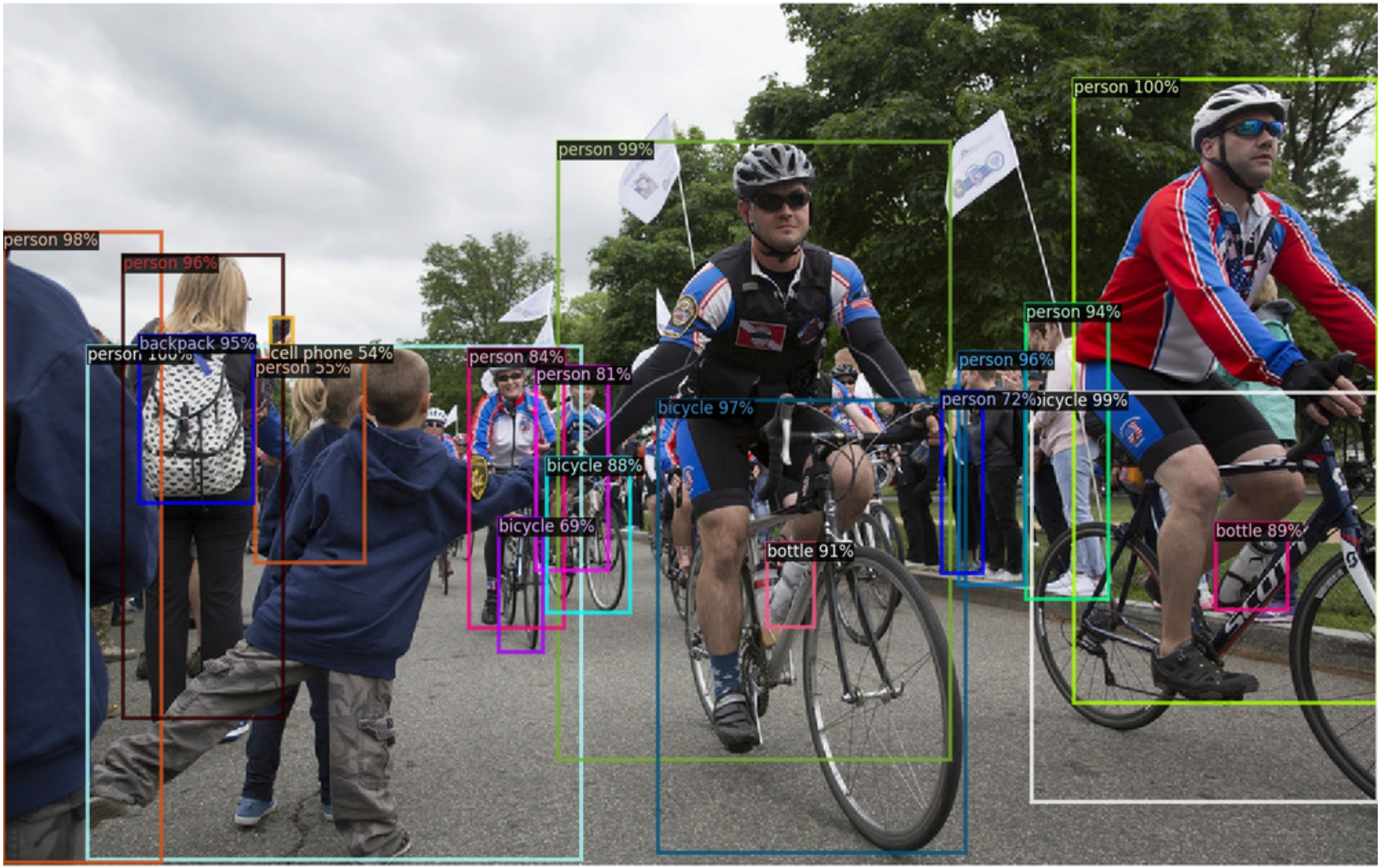
Computer Vision: Reinforcement Learning



Source: https://www.popularmechanics.com/culture/gaming/a32006038/deepmind-ai-atari-agent57/

Digital Images and Datasets

Digital Images

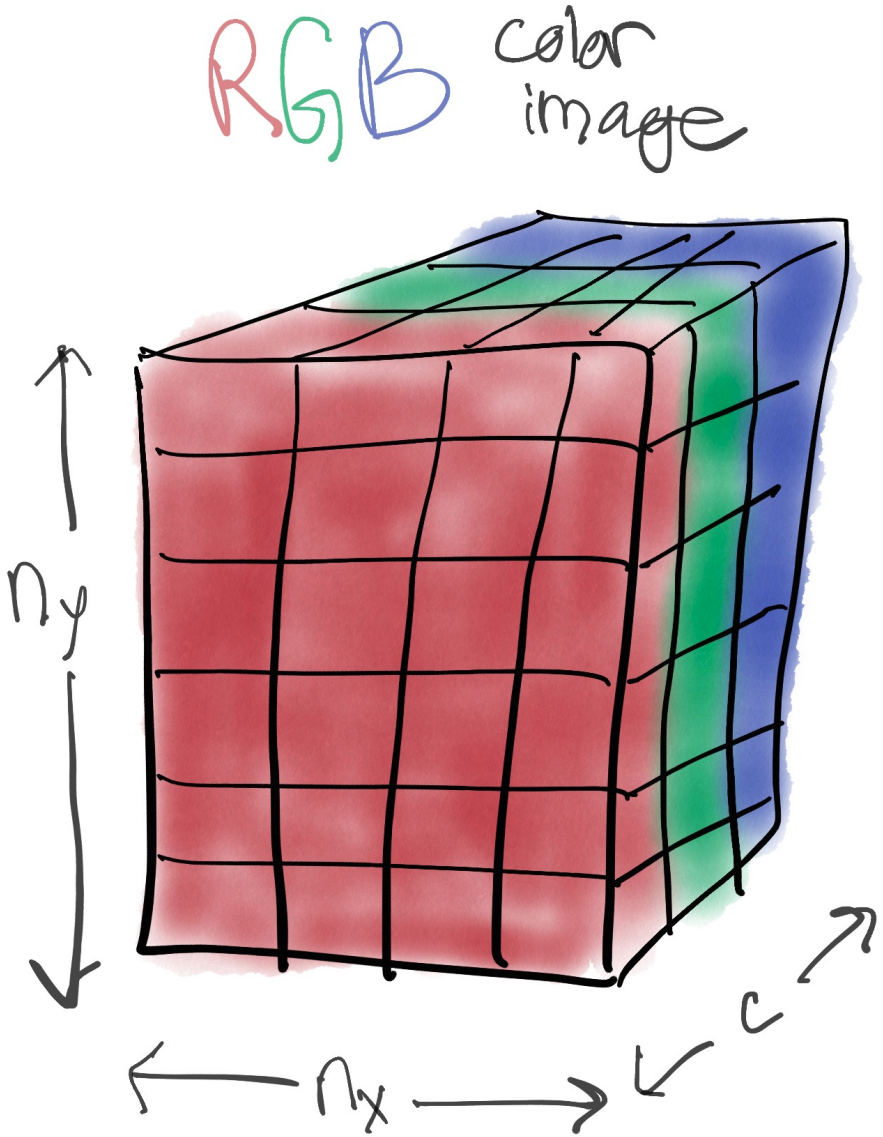


Source: https://github.com/facebookresearch/detectron2

Digital Images

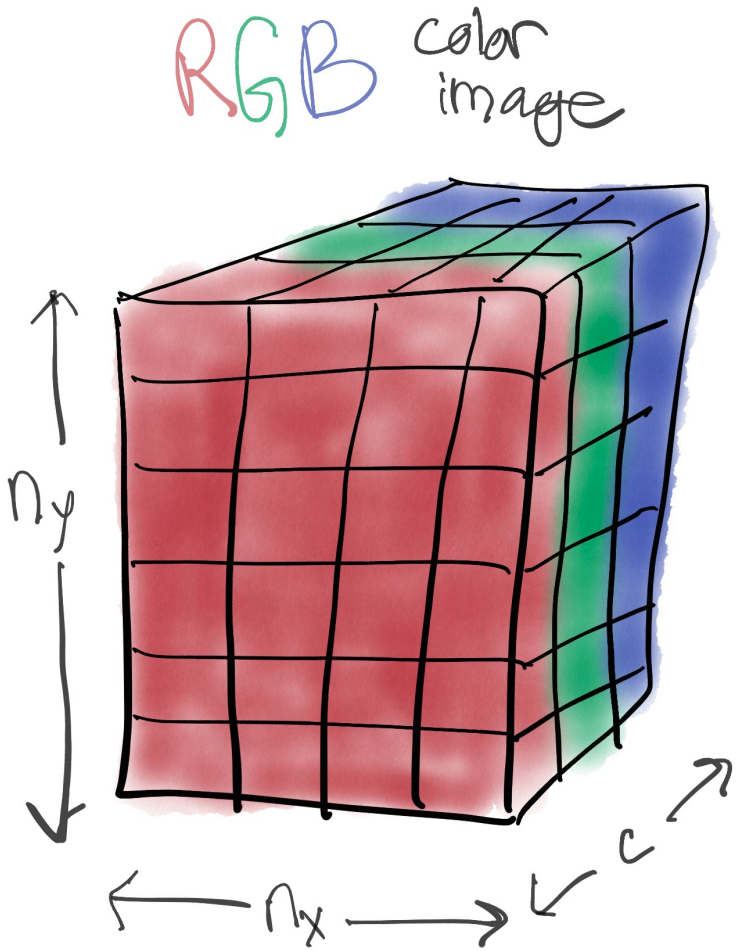
- In order to apply ML to image and video data, need to understand how images are digitally represented
- Black and white images: $n \times n$ matrices
- Color images: $n \times n \times c$ *tensors*, where c is the number of channels or colors
  - Most of the time c is 3 (Red, Green, Blue)
  - Different people use different ordering conventions
- Each pixel is assigned a numerical value for each color
- These values are quantized, range determined by *color depth*
- For example: 8 bits per pixel (bpp): each pixel ranges from 0 to 255, for a total of $2^8 = 256$ possible values
- From the perspective of PyTorch, images are just numerical tensors

Digital Images



Putting it all together - CNN

- Deep learning heavily exploits parallelism
- Images are processed in mini-batches
- input x becomes a tensor with 4 dimensions: (N_B, n, n, c)



Datasets

- Progress in ML has been driven by the availability of large, high-quality datasets
- This facilitated a social practice of having different teams compete on a standardized set of tasks, year after year
- Many new ML papers and tutorials demonstrate their approach on these datasets

MNIST

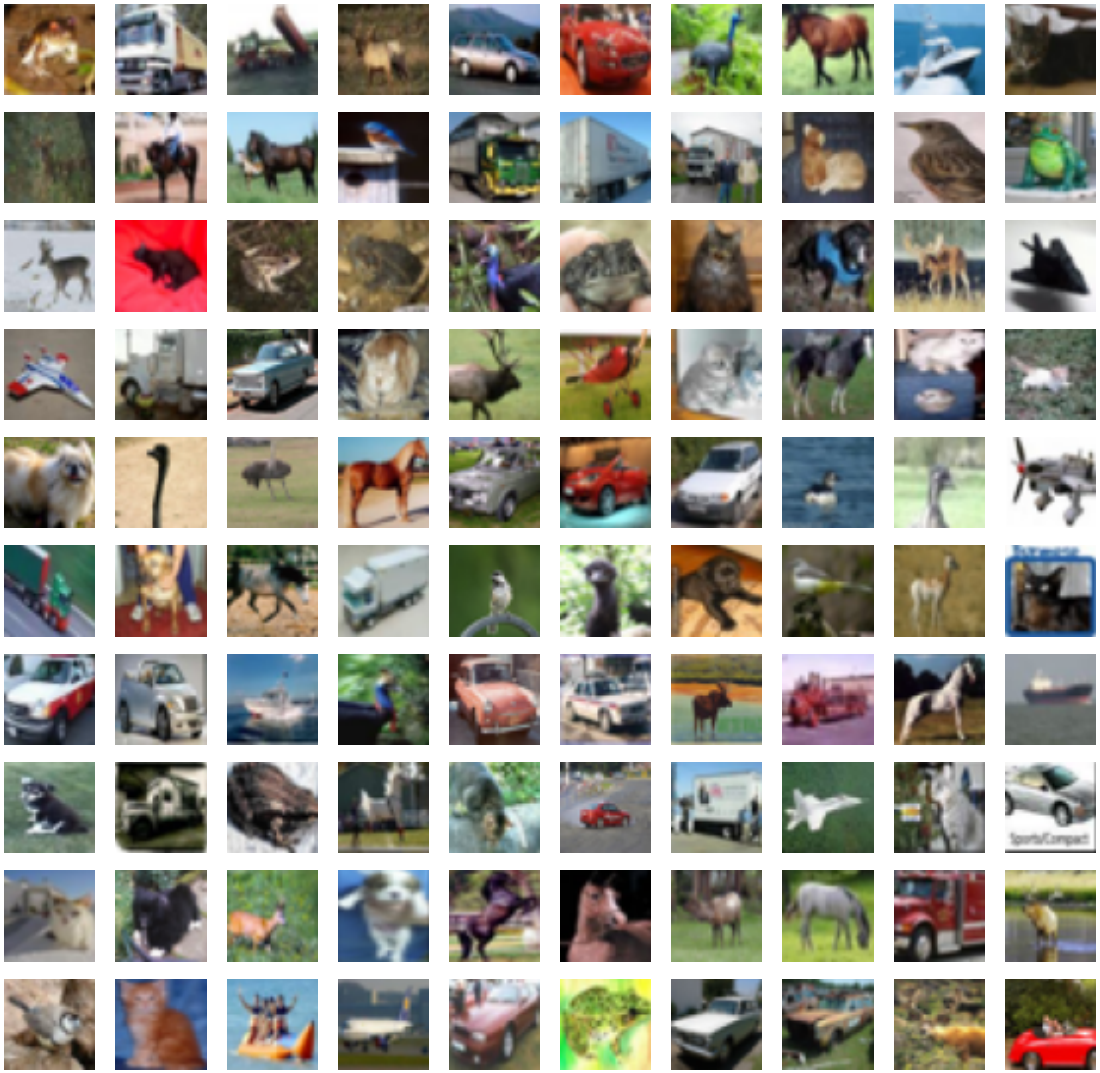
- MNIST: Modified National Institute of Standards and Technology
- Database of handwritten digits (0-9)
- black and white ($c = 1$)
- Each image is 28x28 resolution
- Each 8 bbp, meaning each pixel can take on 256 values
- Size: 60k train, 10k test

MNIST

CIFAR-10, CIFAR-100

- CIFAR: Canadian Institute For Advanced Research
- 32x32 resolution color images
- Size: 50k train, 10k test
- CIFAR-10: 10 classes (6k images/class)
- CIFAR-100: 100 classes (600 images/class)

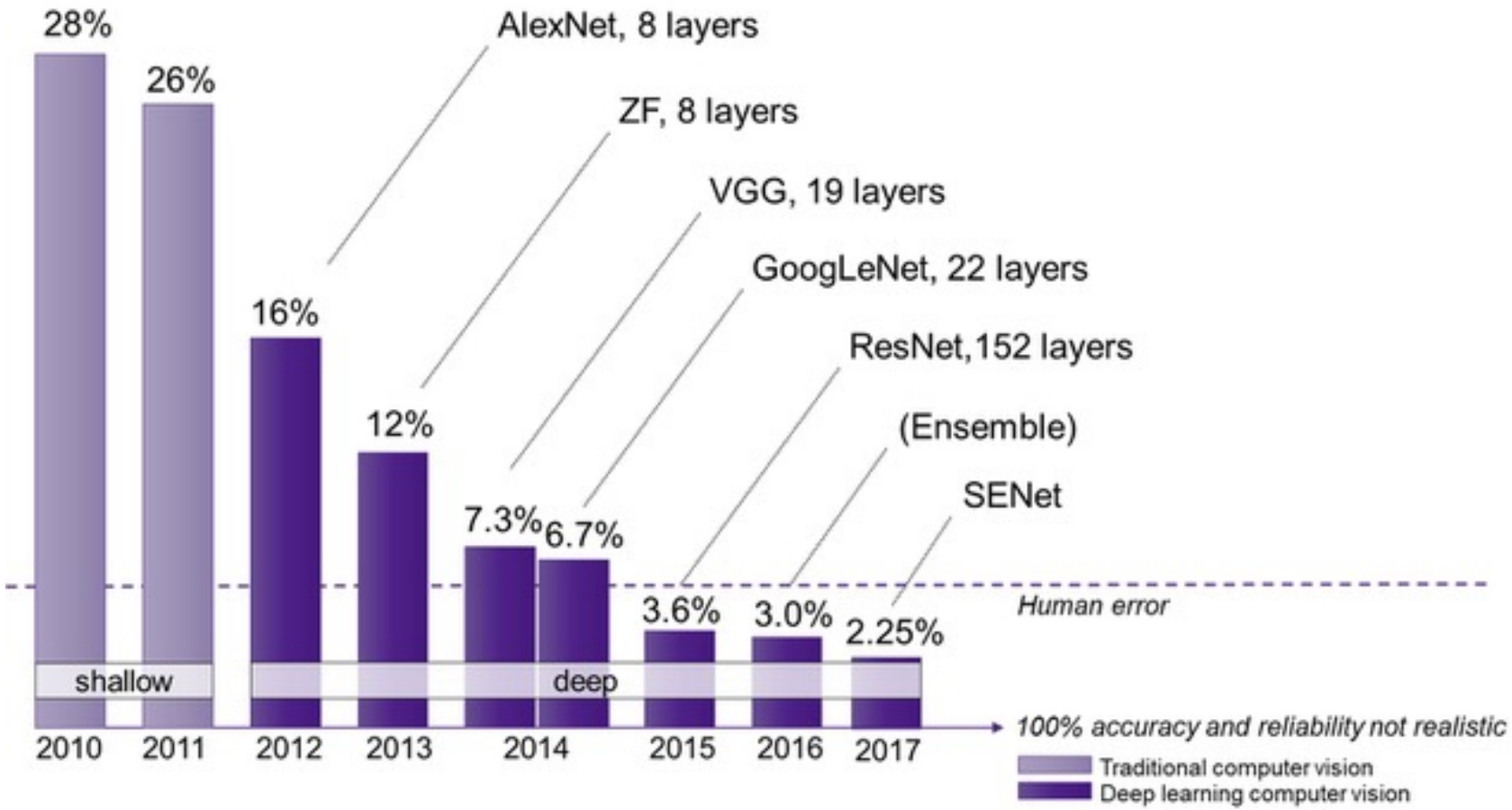
CIFAR-10, CIFAR-100



ImageNet

- Created by Fei Fei Li (and collaborators), now Professor at Stanford
- 14 million labeled images
- 20,000 categories
- Annual competition: ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Blowout performance by AlexNet in 2012 competition is often credited with initiating the current deep learning era
- Fun aside: the 3 authors who built Alex net formed a company called DNN research, evolved into Google Brain
- One of the co-authors went on to co-found OpenAI
- Another is Geoff Hinton, Turing award winner

ImageNet



Source: https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Convolutional Layers

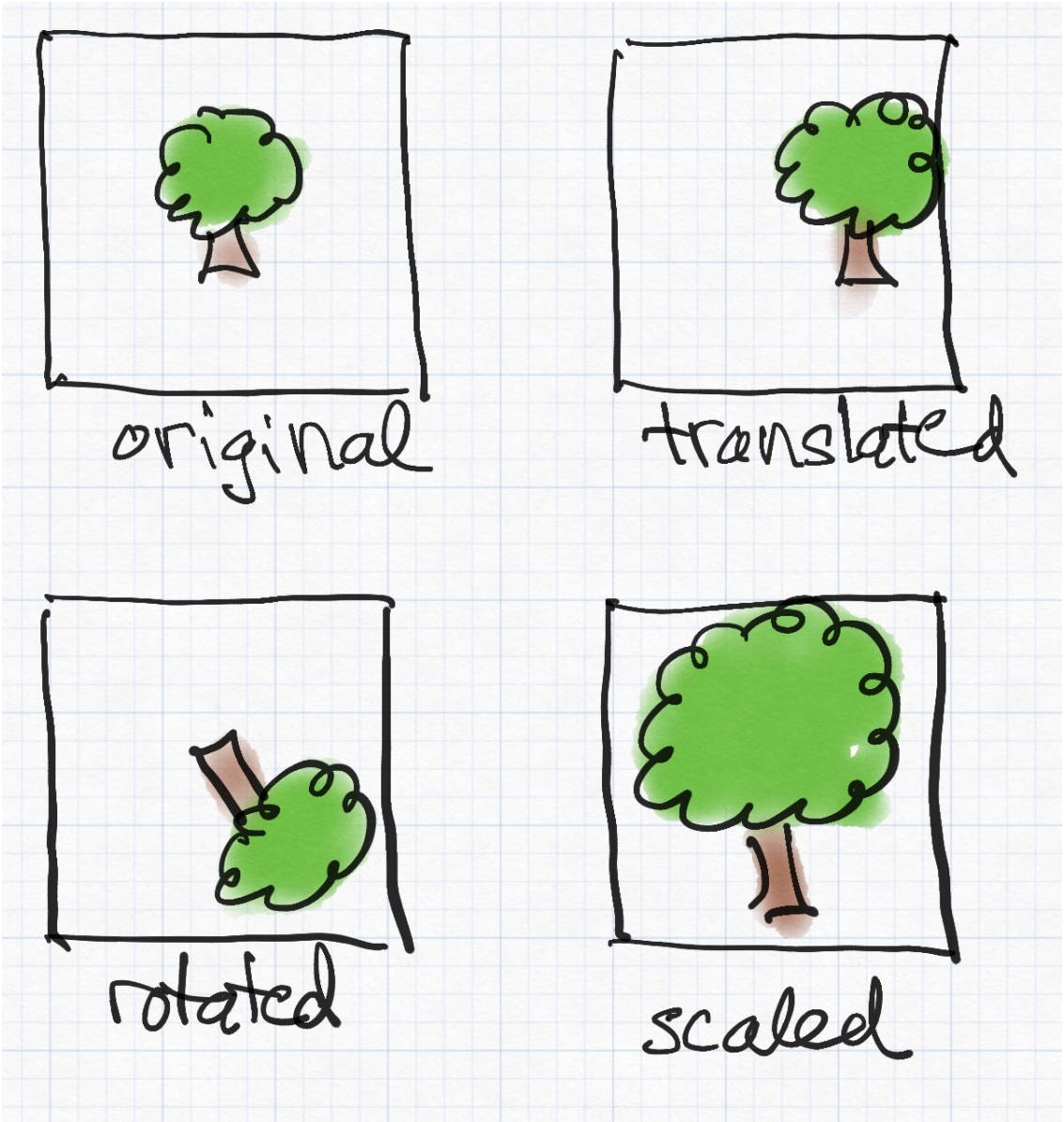
Useful References

- Ch. 9 of Goodfellow, Bengio, Courville's "Deep Learning"
- Lecture Notes from Stanford CS231

Motivation

- Regular (affine) layers exhibit an all-to-all connectivity
- For image data, this can lead to a large number of parameters
  - Not efficient
  - Can be prone to overfitting
- Example: CIFAR-10
  - image size $32 \times 32 \times 3 = 3072$
  - each hidden layer neuron needs $3072 + 1$ params
  - Even worse if we consider high-res images
- Not well-suited to hierarchical way information is organized in images
- Does not incorporate any of the symmetries of the problem (especially translations)
- Not how our eyes work (receptive field)

Some Image Transformations



Convolutions

- Convolution is a general mathematical operation
- There are both discrete and continuous convolutions
  - Continuous

$$(f \star g)(t) = \int_{-\infty}^{\infty} \mathrm{d}t' \, \underbrace{f(t')}_{\text{input}} \underbrace{g(t - t')}_{\text{kernel}}$$

  - Discrete

$$(f \star g)(i) = \sum_{m=-\infty}^{\infty} \underbrace{f(m)}_{\text{input}} \underbrace{g(i - m)}_{\text{kernel}}$$

- Higher dimensional generalizations also exist, e.g.,

$$\sum_{m,n=-\infty}^{\infty} f(m, n) g(i - m, j - n)$$

Convolutions

- Simple example: moving average

$$g_{\mathrm{ME}}(t) = \begin{cases} \frac{1}{T} & 0 < t < T \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\mathrm{MA}}(t) = \int_{-\infty}^{\infty} \mathrm{d}t' f(t') g_{\mathrm{ME}}(t - t') = \frac{1}{T} \int_{t-T}^{t} \mathrm{d}t' f(t')$$

Convolutions

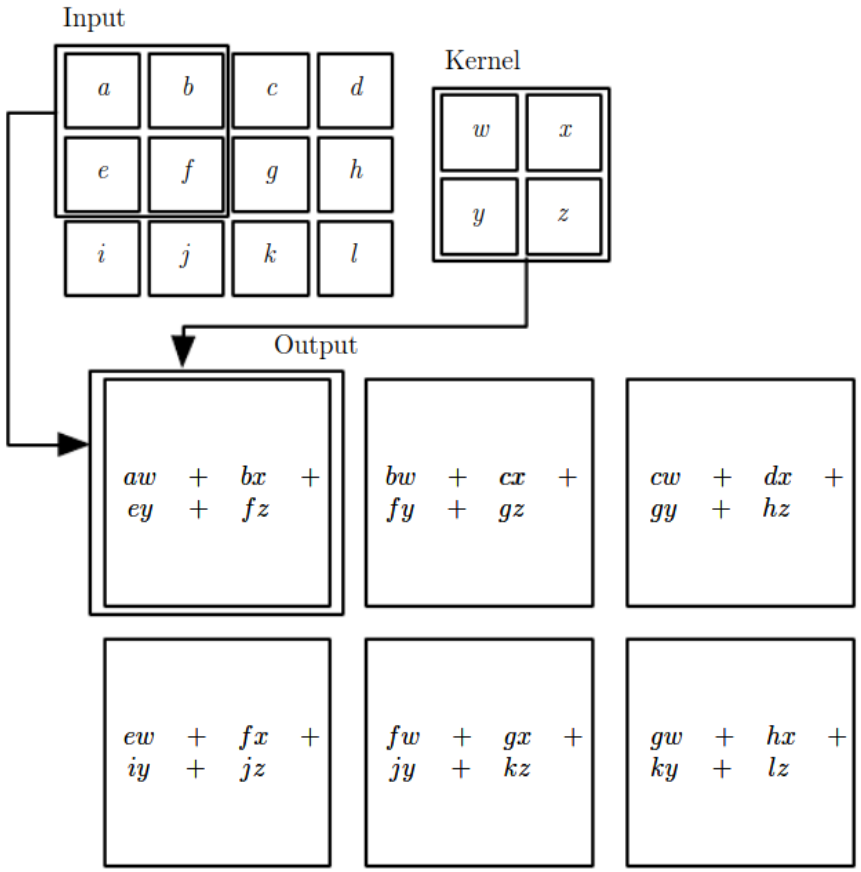


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Source: Goodfellow et al, *Deep Learning*

Convolutions

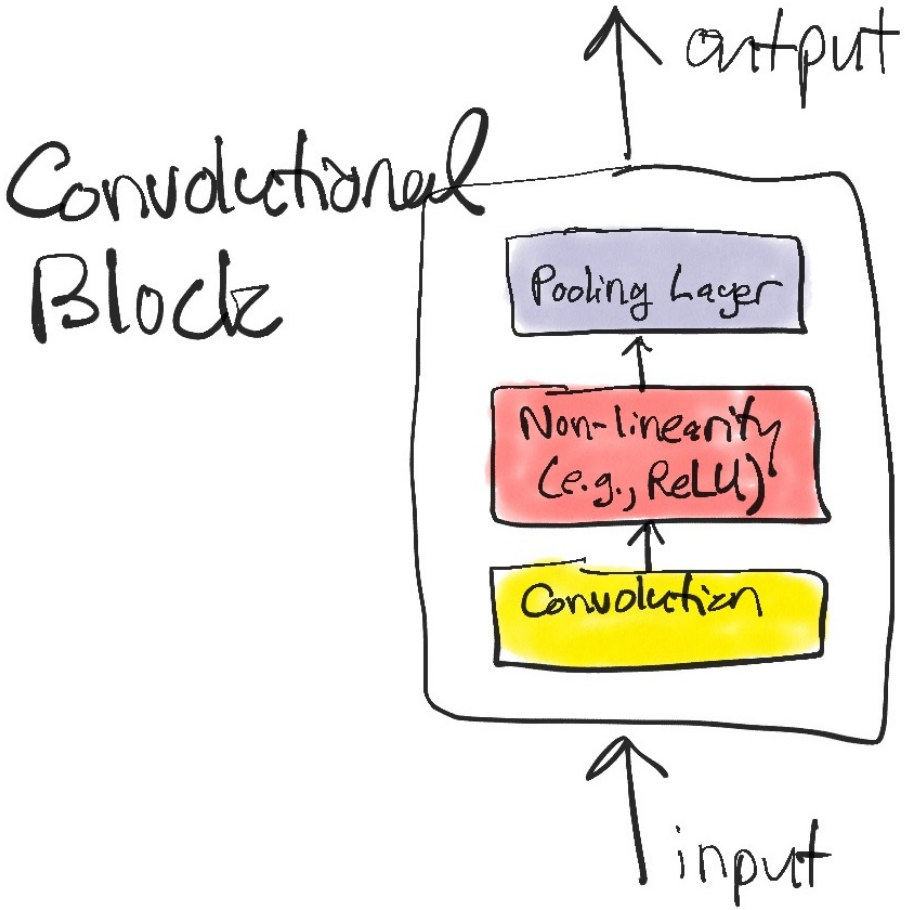
- Convolutions are not invariant to translations
- They are actually *equivariant*:

$$f(\text{transformation}(x)) = \text{transformation}(f(x))$$

- "translated input $\Rightarrow$ translated output"
- Not equivariant to other important transformations (rotations/scalings/changes in brightness, etc)
- Note that convolution operation changes the dimensionality
- Introduce a separate convolution for each color channel

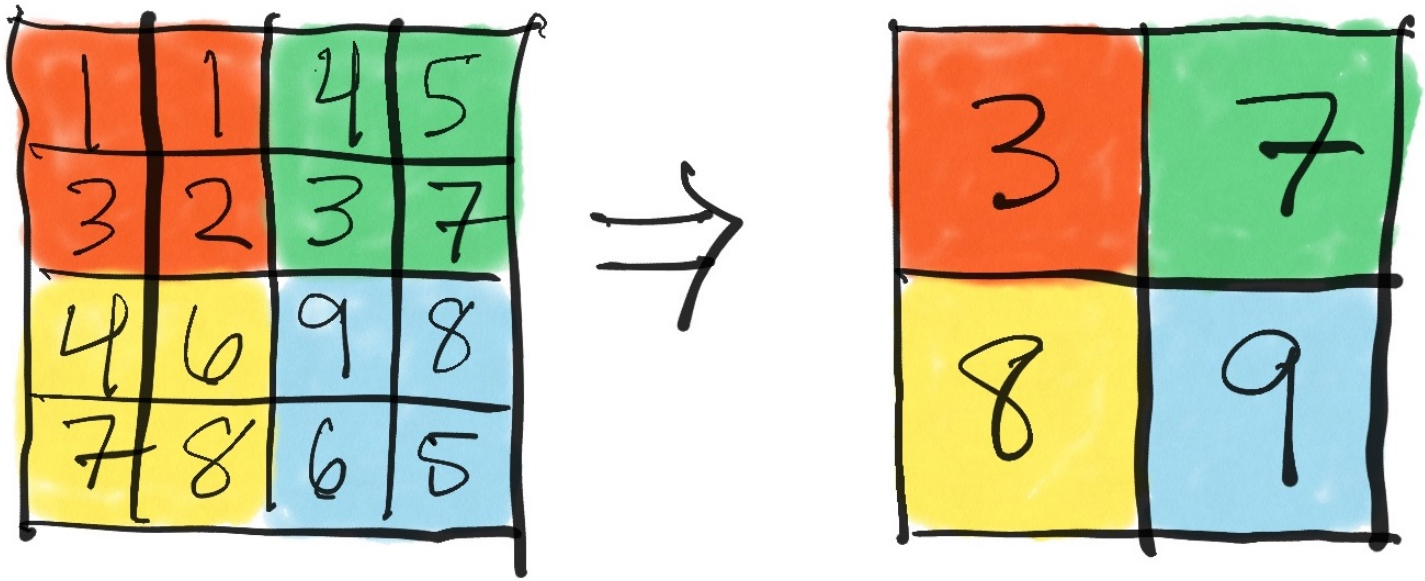
Convolutional Layers/Blocks

- Convolutional layers consist of 3 operations that are almost always grouped together
  - Convolution
  - Non-linear activation function ("detector stage")
  - Pooling function
- Detector stage acts as usual



Pooling

- Pooling replaces output with summary statistic of neighboring pixels
- Goal of pooling is to enforce **approximate** *invariance* to translations
- Different variants of pooling, common one is *max pooling*
- By pooling together separately parametrized convolutions, can in theory learn other invariances



Pooling

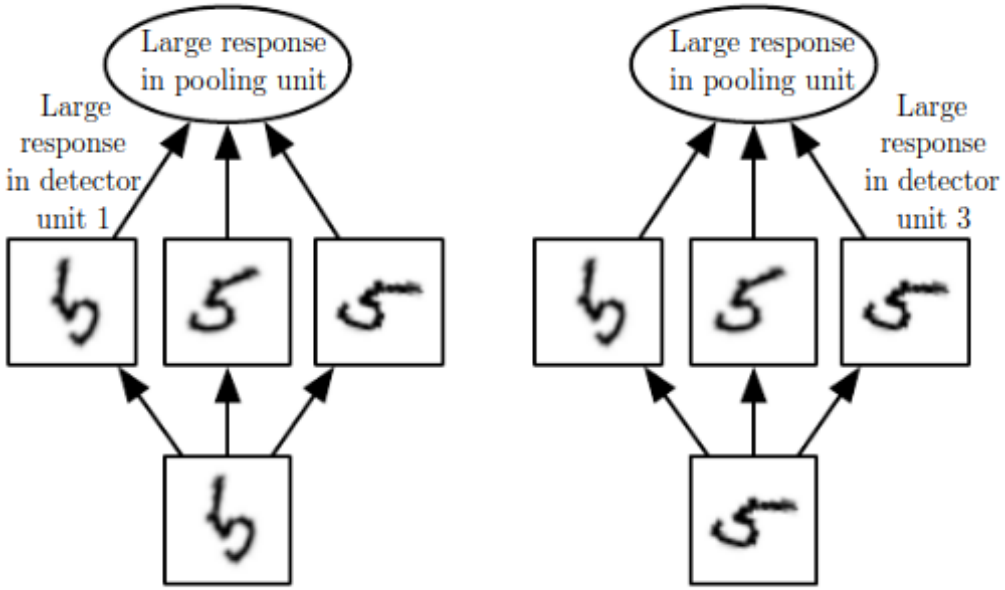
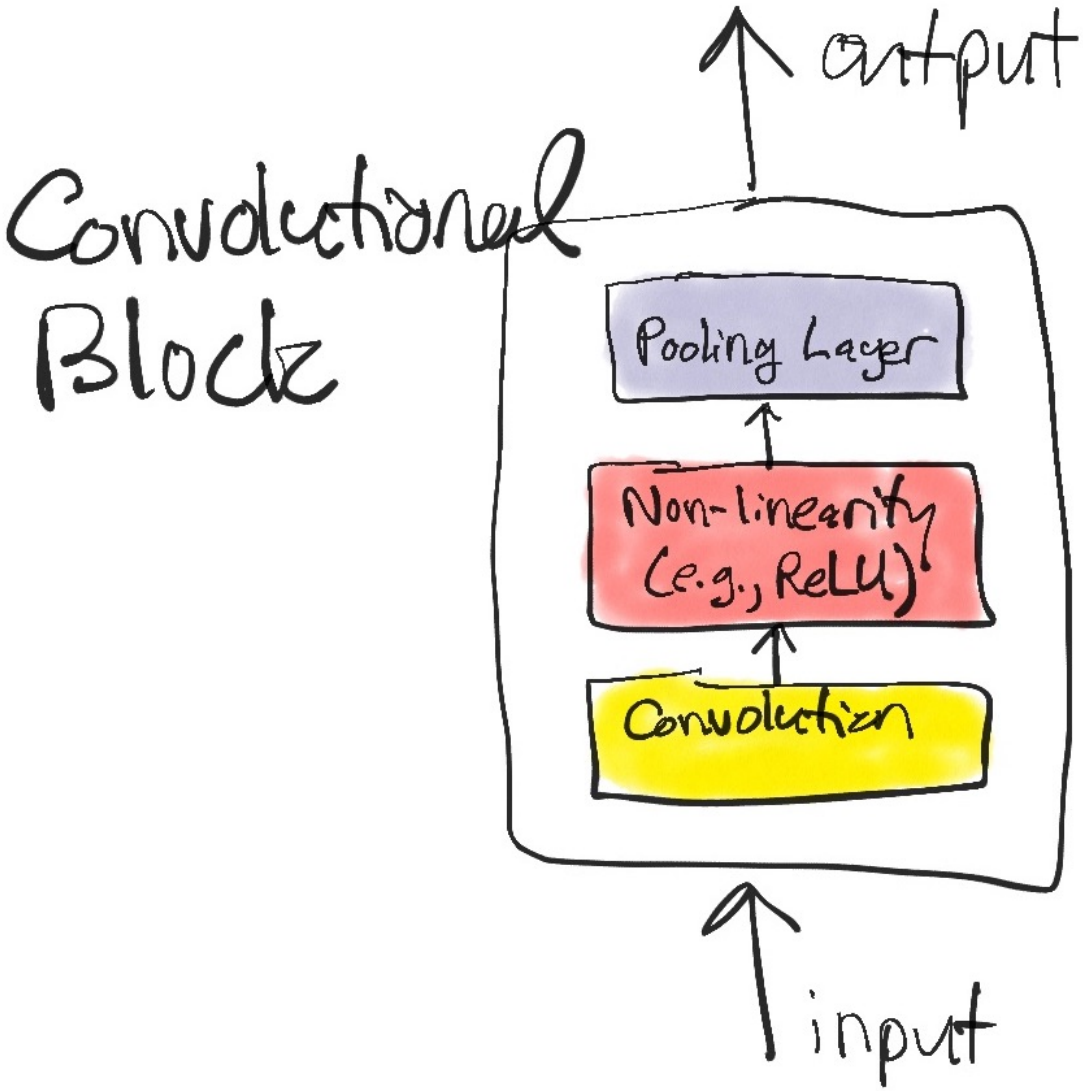


Figure 9.9: Example of learned invariances. A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow *et al.*, 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multichannel approach is only necessary for learning other transformations.

Convolutional Layers/Blocks

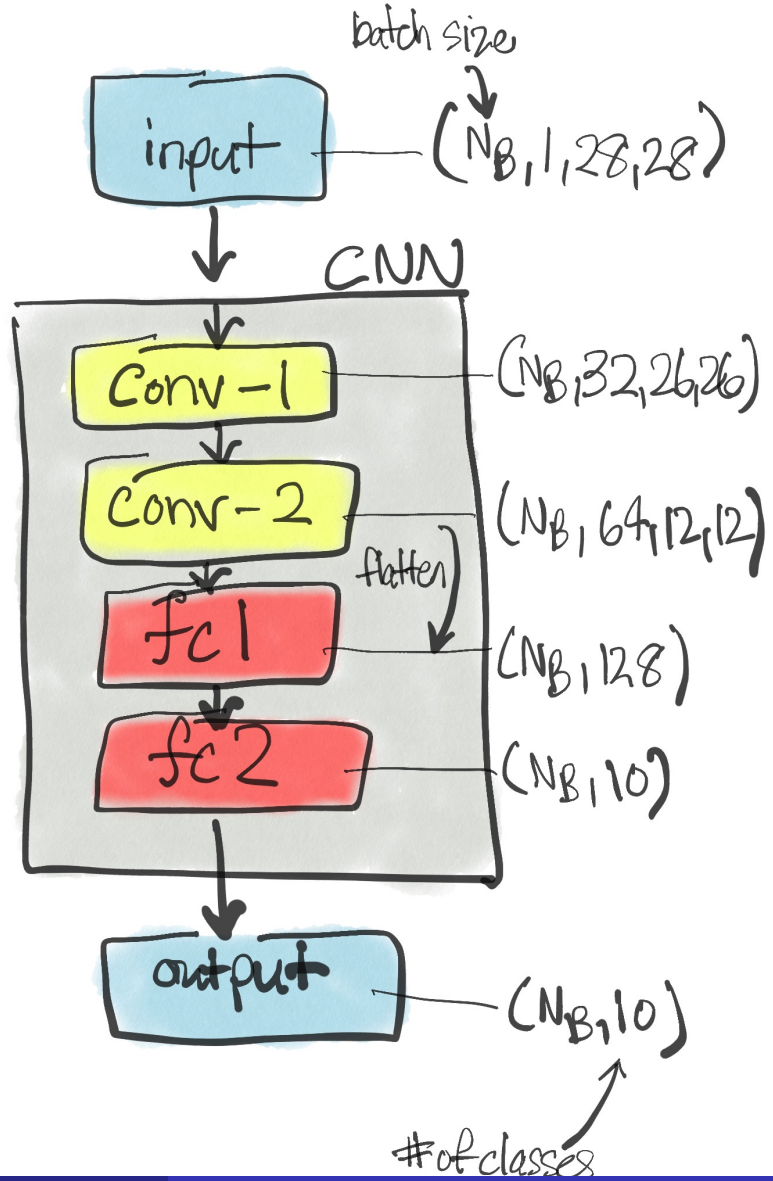


Putting it all together - CNN

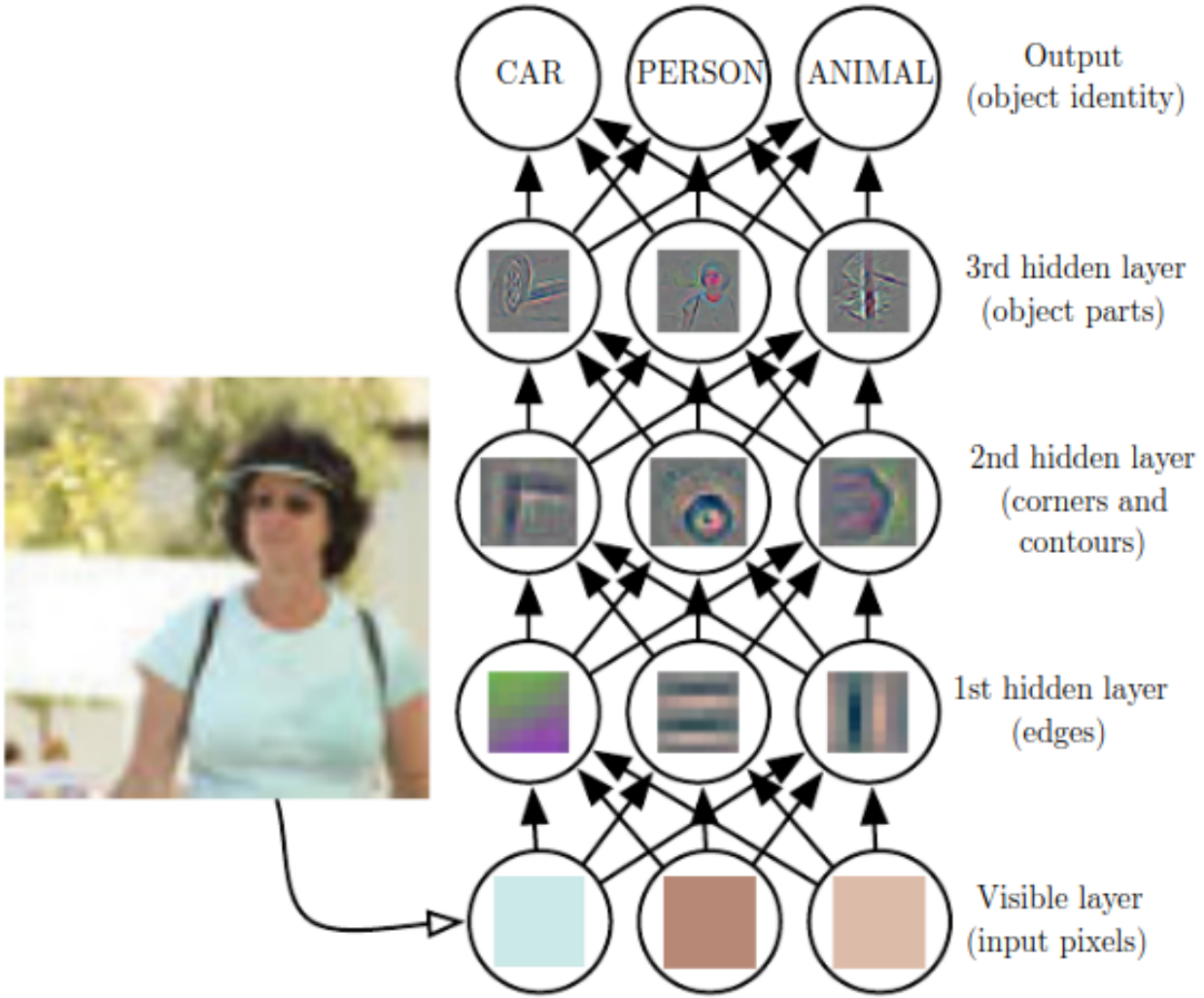
[15]
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        ## convolutional block (no max pool - not sure why)
        x = self.conv1(x)
        x = F.relu(x)

        ## convolutional block
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)

        ## fully connected layer 1
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)

        ## fully connected layer 2
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output

```

Putting it all together - CNN

```
[15]  1 class Net(nn.Module):
      2     def __init__(self):
      3         super(Net, self).__init__()
      4         self.conv1 = nn.Conv2d(1, 32, 3, 1)
      5         self.conv2 = nn.Conv2d(32, 64, 3, 1)
      6         self.dropout1 = nn.Dropout(0.25)
      7         self.dropout2 = nn.Dropout(0.5)
      8         self.fc1 = nn.Linear(9216, 128)
      9         self.fc2 = nn.Linear(128, 10)
     10
     11     def forward(self, x):
     12         ## convolutional block (no max pool - not sure why)
     13         x = self.conv1(x)
     14         x = F.relu(x)
     15
     16         ## convolutional block
     17         x = self.conv2(x)
     18         x = F.relu(x)
     19         x = F.max_pool2d(x, 2)
     20         x = self.dropout1(x)
     21
     22         ## fully connected layer 1
     23         x = torch.flatten(x, 1)
     24         x = self.fc1(x)
     25         x = F.relu(x)
     26         x = self.dropout2(x)
     27
     28         ## fully connected layer 2
     29         x = self.fc2(x)
     30         output = F.log_softmax(x, dim=1)
     31         return output
     32
```



CNNs Learn Hierarchical Features



Source: Goodfellow et al, Figure 1.2, adapted from Zeiler and Fegus (2014): https://arxiv.org/pdf/1311.2901.pdf

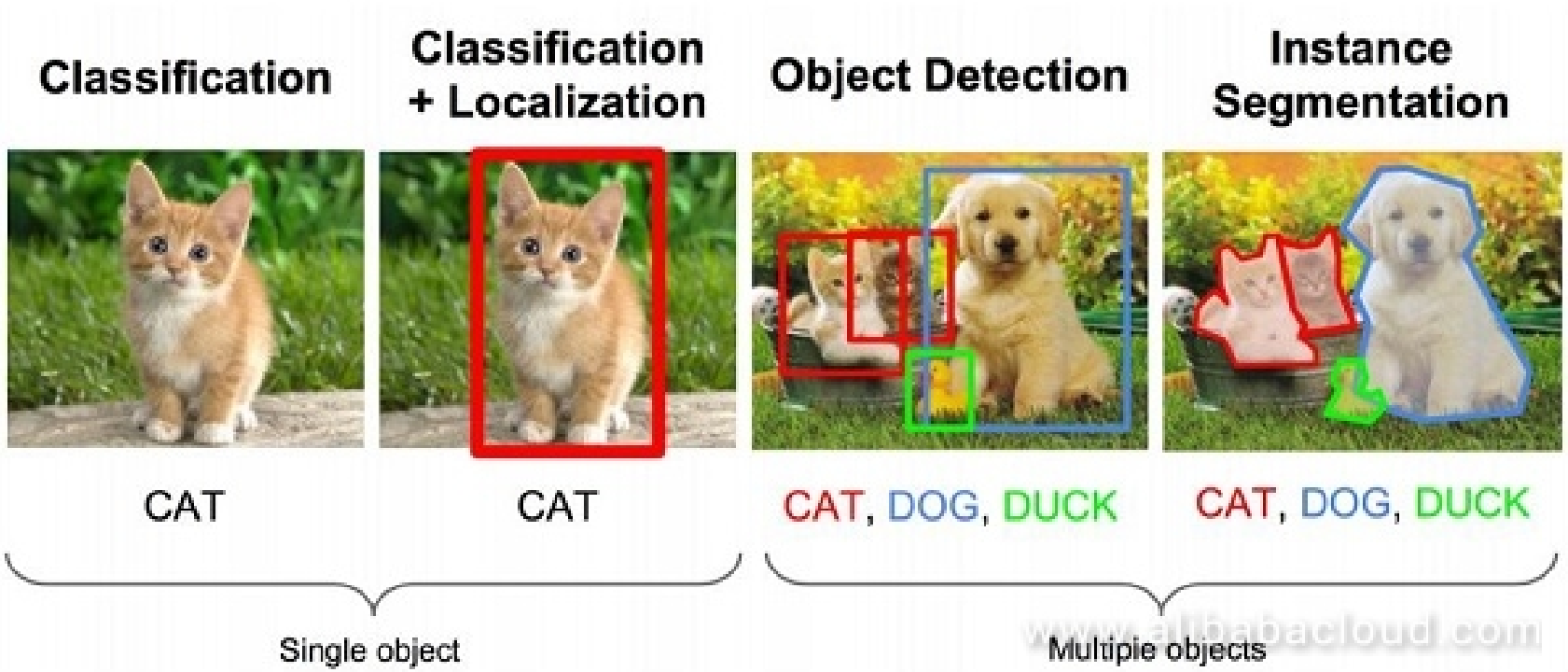
Advanced computer vision models

Useful References

- Lillian Weng's (OpenAI) excellent series of https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

Advanced computer vision models

- Thus far we have discussed simple CNNs for classification
- Extensions of CNNs can be used for more interesting applications, such as object detection/ instance segmentation



Source: https://www.alibabacloud.com/blog/from-r-cnn-to-faster-r-cnn-the-evolution-of-object-detection-technology_593829

Advanced computer vision models

- Covering these models is beyond the scope of this class (perhaps we'll cover them in the Advanced AI class)
- An excellent online tool is Facebook's Detectron2