MCIT AWS Machine Learning Training

Use Case – Bees Object Detection

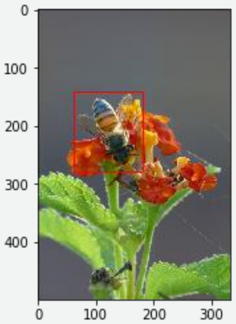
Amgad Zaki, AWS Sr. Solutions Architect

amgadz@amazon.com

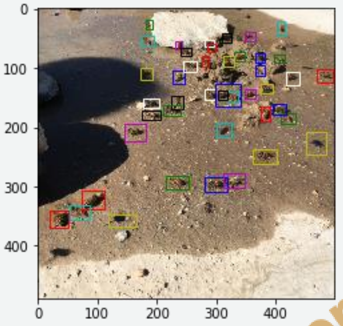
Use Case

- Using Python code, you need to create a training job for Bees Imgaes using Amazon SageMaker Object Detection algorithm.
- Dataset is shared with class materials.
- Use only 2% of radomly selected images.
- Amazon SageMaker Ground Truth is free for 500 images labeling only.

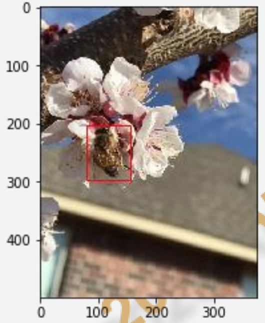
Dataset



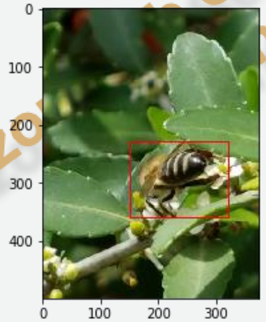
0
100
200
300
400
0
100
200
300



0
100
200
300
400
0
100
200
300
400



0
100
200
300
400
0
100
200
300



0
100
200
300
400
0
100
200
300

Use Case

- Bonus task
  - Create Lambda function that invokes created endpoint.
  - Use API Gateway to create REST API by invoking the lambda function.

Use Case: Submission

Please submit the notebook by email, and avoid last minute submissions.

Failing to submit your notebook will impact your overall progress grading.

Introduction to Computer Vision (CV)

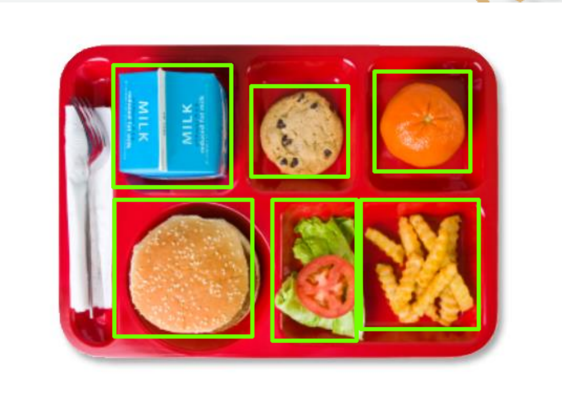
CV Problems

Image Classification: What category(ies) does this image belong to?

Breakfast ?
Lunch ?
Dinner ?

CV Problems

Object Detection: What are the locations of detected object instances?



Milk
Hamburger
Cookie
Salad
Orange
French fries

CV Problems

Semantic Segmentation: What are the boundaries and locations of detected objects?

Milk
Hamburger
Cookie
Salad
Orange
French fries

Image Representations

Image Representation

- Images are made of pixels. Pixels have values between 0-255.
- We usually consider color and gray level images.

Color Images:
RGB (Red-Green-Blue) is a common representation.
We have 3 channels (Red, Green and Blue).

Grayscale Images:
- Made of single channel.
- Pixels have values between 0-255.
- 0: Darkest intensity
- 255: Brightest intensity

Image Representation

On a N x M grayscale image (N: # of rows, M: # of columns):

Columns

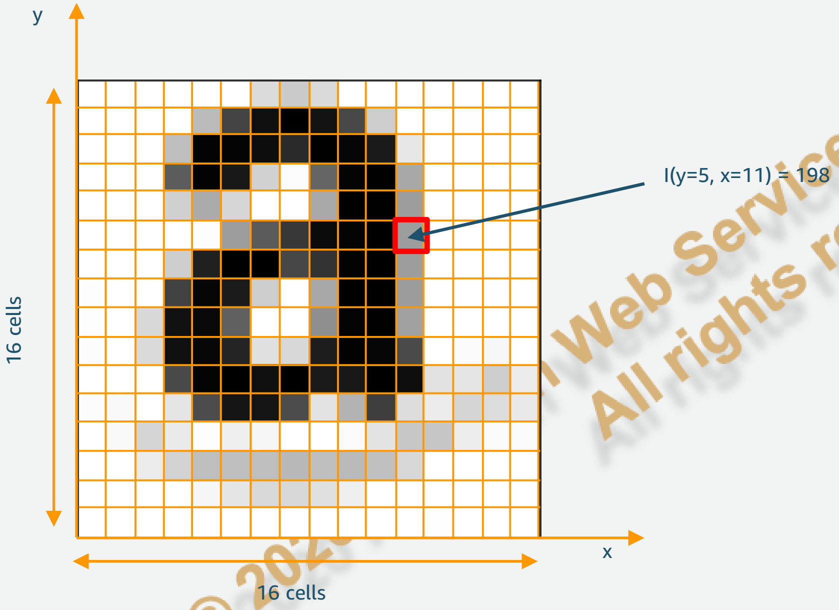
Rows

112	103	121	99	76	45	44	61
98	123	131	132	74	49	[illegible]	66
157	158	165	134	78	51	52	59
145	143	134	131	124	121	125	123
141	141	143	92	91	80	83	88
142	145	151	93	88	86	85	79
223	221	225	226	221	235	222	231
209	201	214	216	226	237	231	230

image[0, 0] => Top left corner of image
image[y, x] => y pixels down, x pixels right
image[N-1, M-1] => Bottom right corner

Image Representation

E.g., on a 16x16 grayscale image:



16x16 grayscale image.

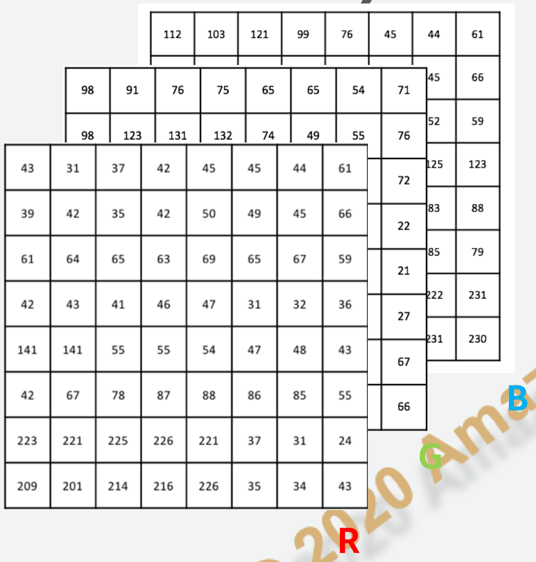
- x indexes => [0,.., 15]
- y indexes => [0,.., 15]

Color Images

- Made of Red, Green and Blue channels.
- Each channel has values (intensity) between 0-255.
- 0: Darkest intensity
- 255: Brightest intensity

Color Images

On a N x M x 3 color image (N: # of rows, M: # of columns)

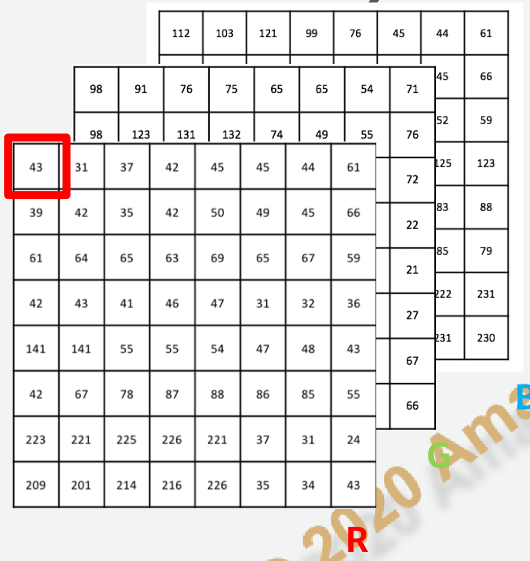


im[0, 0, 0] => Top left corner of “R” image (43)
im[0, 0, 1] => Top left corner of “G” image (98)
im[0, 0, 2] => Top left corner of “B” image (112)

im[N-1, M-1, 0] => Down right corner of “R” (43)
im[N-1, M-1, 1] => Down right corner of “G” (66)
im[N-1, M-1, 2] => Down right corner of “B” (230)

Color Images

On a N x M x 3 color image (N: # of rows, M: # of columns)

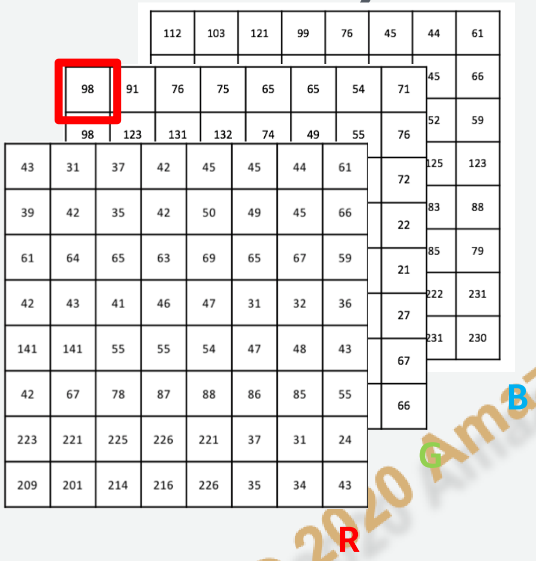


im[0, 0, 0] => Top left corner of “R” image (43)
im[0, 0, 1] => Top left corner of “G” image (98)
im[0, 0, 2] => Top left corner of “B” image (112)

im[N-1, M-1, 0] => Down right corner of “R” (43)
im[N-1, M-1, 1] => Down right corner of “G” (66)
im[N-1, M-1, 2] => Down right corner of “B” (230)

Color Images

On a N x M x 3 color image (N: # of rows, M: # of columns)

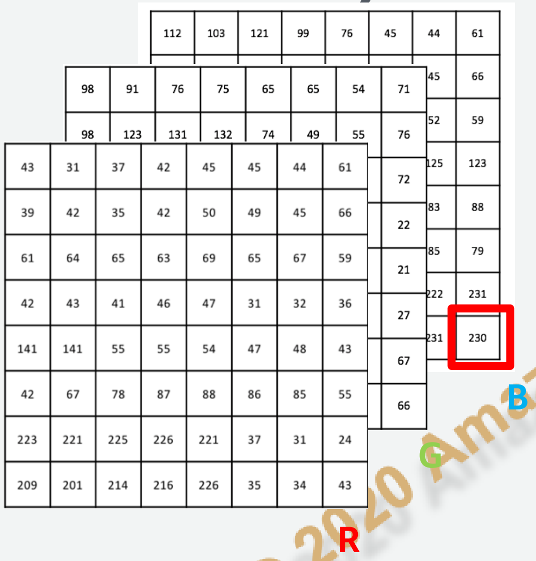


im[0, 0, 0] => Top left corner of “R” image (43)
im[0, 0, 1] => Top left corner of “G” image (98)
im[0, 0, 2] => Top left corner of “B” image (112)

im[N-1, M-1, 0] => Down right corner of “R” (43)
im[N-1, M-1, 1] => Down right corner of “G” (66)
im[N-1, M-1, 2] => Down right corner of “B” (230)

Color Images

On a N x M x 3 color image (N: # of rows, M: # of columns)

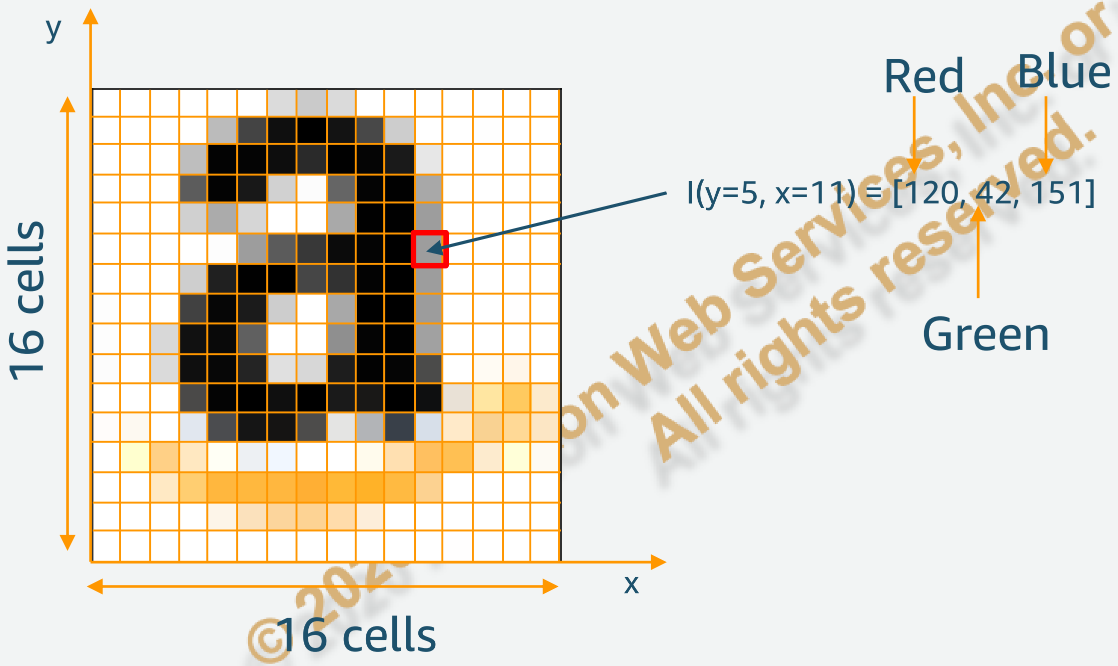


im[0, 0, 0] => Top left corner of “R” image (43)
im[0, 0, 1] => Top left corner of “G” image (98)
im[0, 0, 2] => Top left corner of “B” image (112)

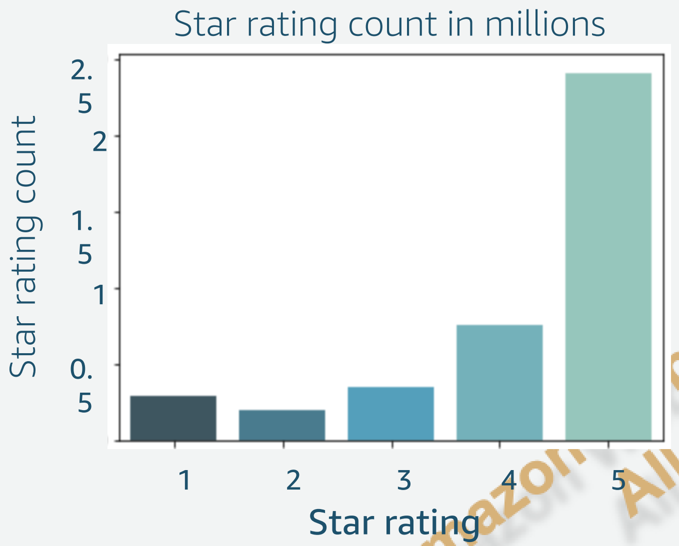
im[N-1, M-1, 0] => Down right corner of “R” (43)
im[N-1, M-1, 1] => Down right corner of “G” (66)
im[N-1, M-1, 2] => Down right corner of “B” (230)

Color Images

E.g., on a 16x16 RGB image:



Class Imbalance



Number of samples per class is **not equally distributed.**

The ML model may not work well for the **infrequent classes**.

Examples:

- Fraud Detection
- Anomaly Detection
- Medical Diagnosis

Amazon review dataset: The number of 5 star reviews almost equals the total of the other 4 types of star reviews combined.

Image Augmentation

Make series of random changes to the training images to produce similar, but different, training examples (**w/o affecting the label!**)

For improving model's capability for **generalization**

Common methods:

- Resizing (with different scales and ratios), zoom in/out
- Cropping and flipping, translations, rotations
- Changing color, brightness, contrast, add noise (salt and pepper noise, etc)
- Apply these transformations during training, do not cache augmented images, generate them during training time!

[More details]

Image Augmentation

Cropping and flipping

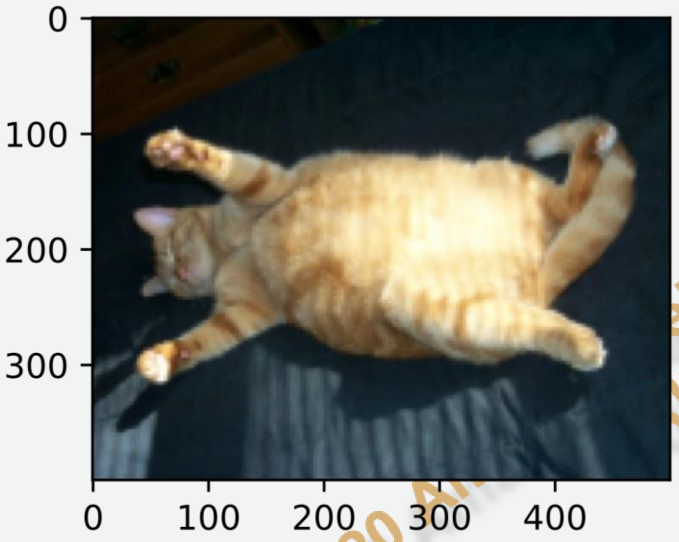
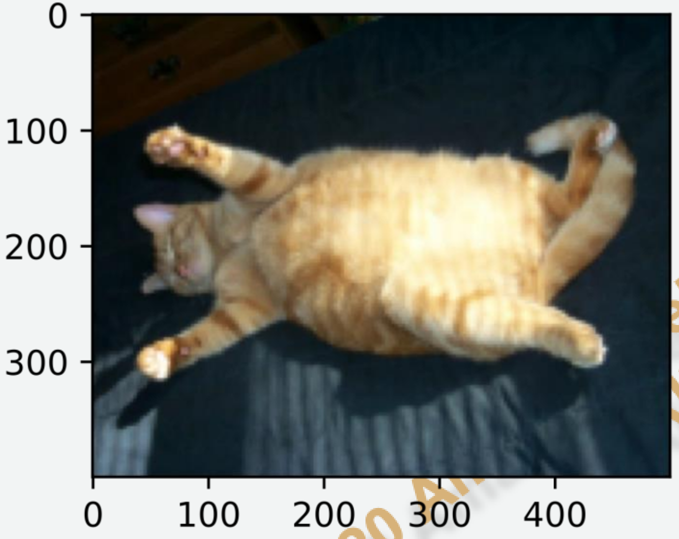


Image Augmentation

Changing color and brightness



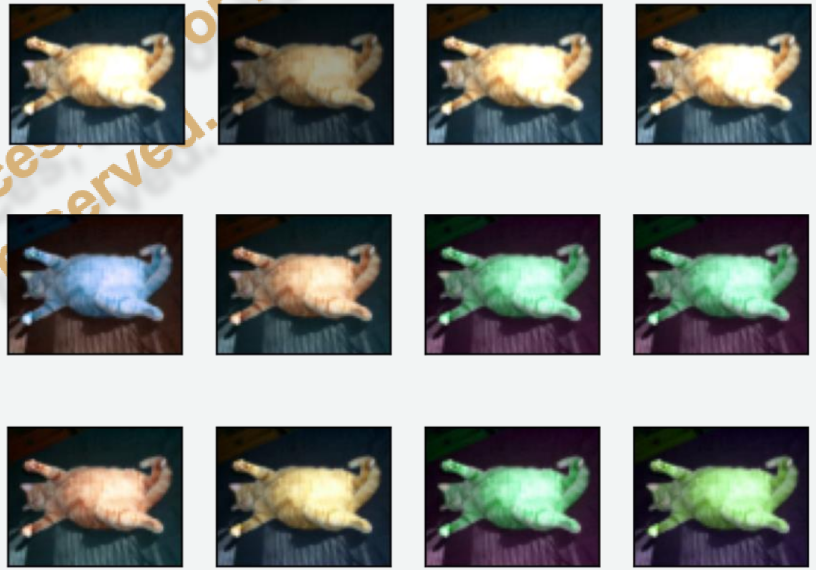


Image Datasets

Image Classification Dataset

Publicly available CV datasets or benchmarks for classification task

- MNIST: http://yann.lecun.com/exdb/mnist/
- Fashion – MNIST: https://github.com/zalandoresearch/fashion-mnist
- CIFAR 10: https://www.cs.toronto.edu/~kriz/cifar.html
- Open Images: https://github.com/openimages/dataset
- Places: http://places2.csail.mit.edu/index.html
- ImageNet: http://www.image-net.org/
- Caltech 101: http://www.vision.caltech.edu/Image_Datasets/Caltech101/
- Caltech 256: http://www.vision.caltech.edu/Image_Datasets/Caltech256/
- Vehicle make and model rec. dataset: http://vmmrdb.cecsresearch.org/

Many pretrained models here

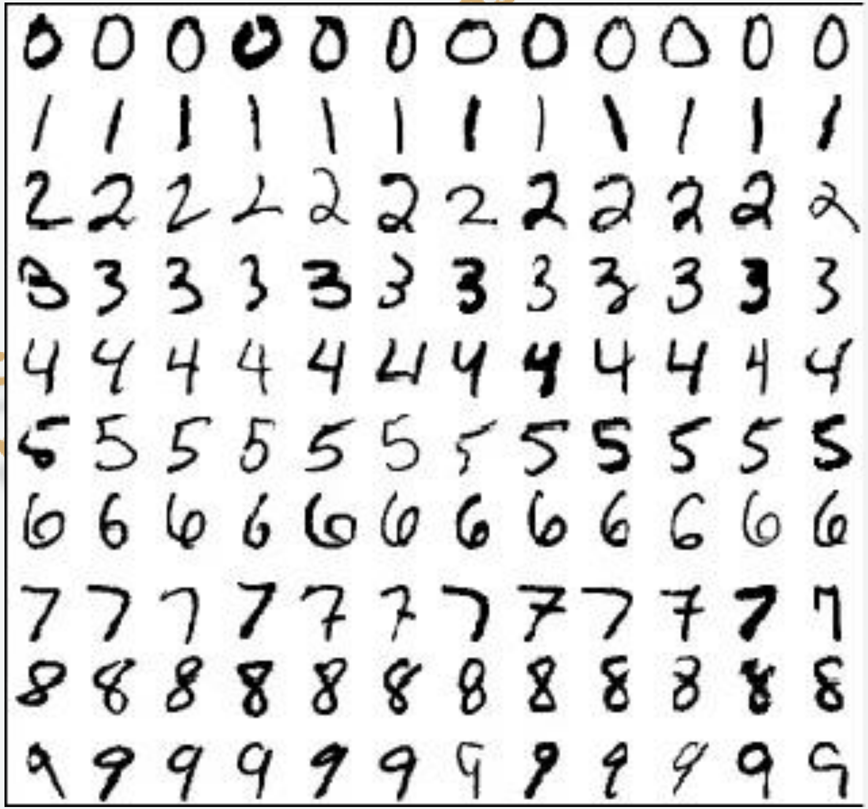
MNIST

A dataset of handwritten digits (0,1..,9) images

- 28 x 28 grayscale
- Centered and scaled
- 50,000 training data
- 10,000 test data

Available for downloading here

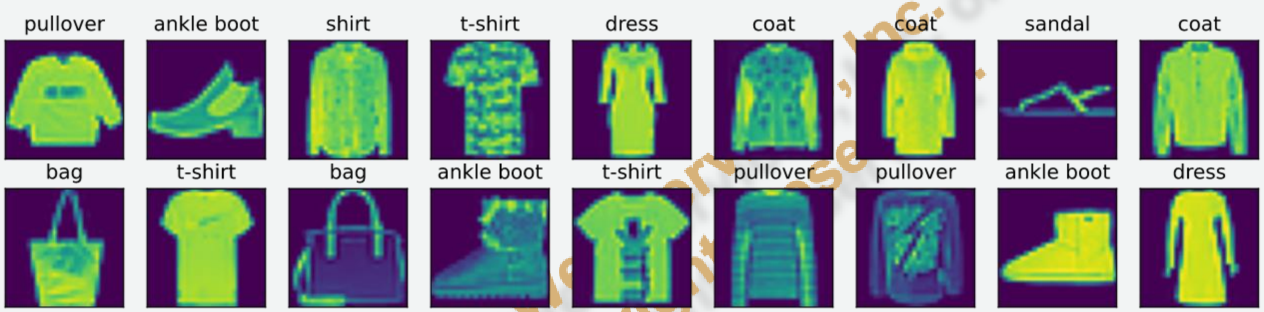
Modified National Institute of Standards and Technology (MNIST)



Fashion - MNIST

A drop-in replacement of MNIST

- There are 10 classes.



- 28 x 28 grayscale; Centered and scaled
- 60,000 training data and10,000 test data

[More details]

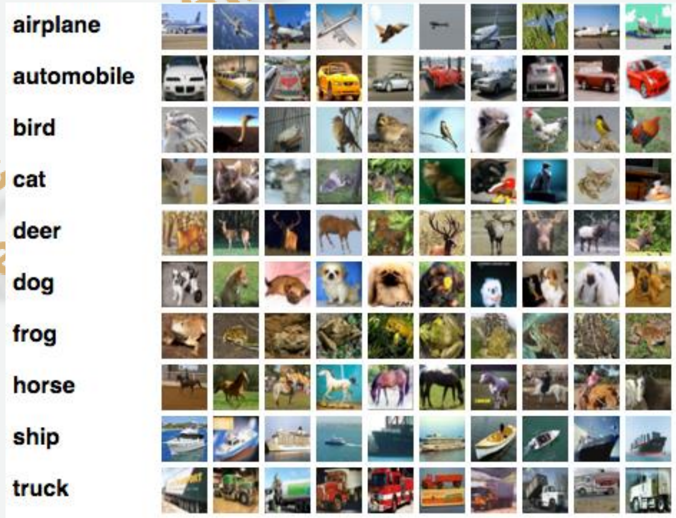
CIFAR 10

An established computer-vision dataset used for object recognition.

- 32x32 color images
- 10 classes: 6,000 images per class
- 50000 training and 10000 test images
- Not centered and scaled

Canadian Institute For Advanced Research (CIFAR)

[More details]



Places Dataset*

More than 10M images with 400+ unique scene categories

- 5000 - 30,000 training images per class
- Consistent with real-world frequencies of occurrence

More details (MIT)

ImageNet Dataset

One of the most popular image classification datasets

- The largest dataset by 2012 (at the time orders of magnitude)
- 22,000 distinct categories over 14M images
- Li Fei-Fei, Stanford University

http://www.image-net.org/

ImageNet Competition

ImageNet Large Scale Visual Recognition Challenge (ILSVRC):

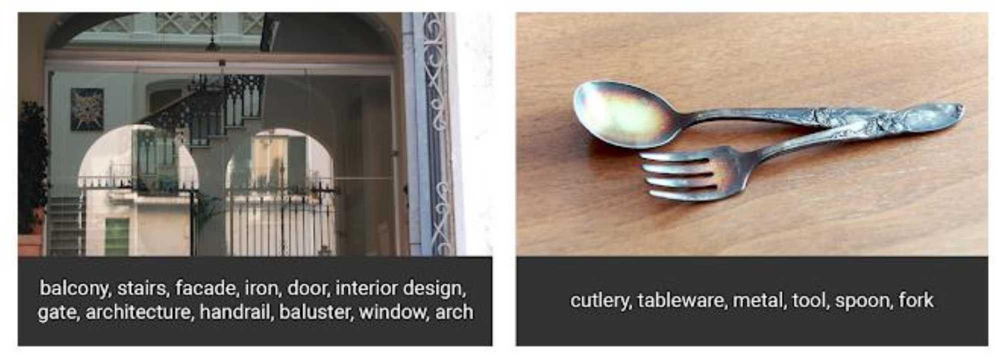
- Image classification competition, started since 2010
- ML models are trained to classify 1000 image categories
- Training size: ~1.2M; Validation size: 50,000; Test size: 100,000
- Dominated by deep neural networks since 2012
- AlexNet, VGGNet and ResNet models that were trained on this dataset (Coming soon!)

More details

Open Images Dataset

The largest existing dataset with object location annotations

- About 9M images with over 6000 categories
- 16M bounding boxes for 600 object classes on 1.9M images



More details

Some Important CNNs

Convolutional Neural Networks:

- Alternate between convolutions, nonlinearities and pooling
- Ultimately the resolution of the input image is reduced prior to emitting an output via one (or more) dense layers

Important Convolutional Neural Networks:

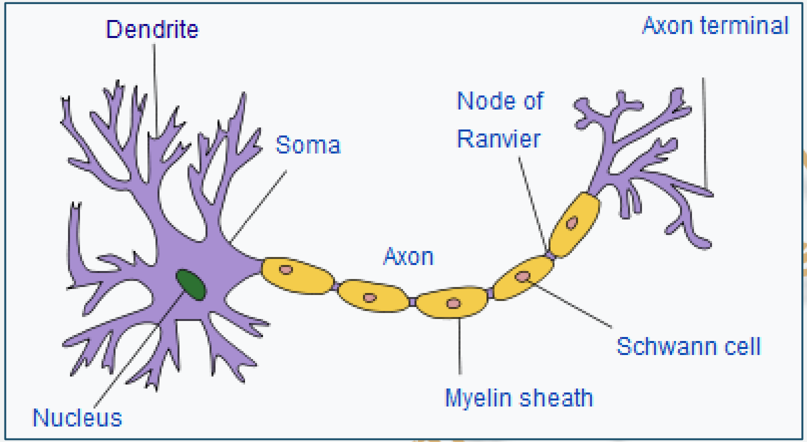
- LeNet (1989, 1998) – Yann LeCun (NYU) (Facebook AI Research)
- AlexNet (2012) - Alex Krizhevsky (U Toronto) (Google)
- VGGNet (2014) (Visual Geometry Group Oxford, Google DeepMind)
- ResNet (2015) (Kaiming He, Microsoft Research) (Facebook AI Research)

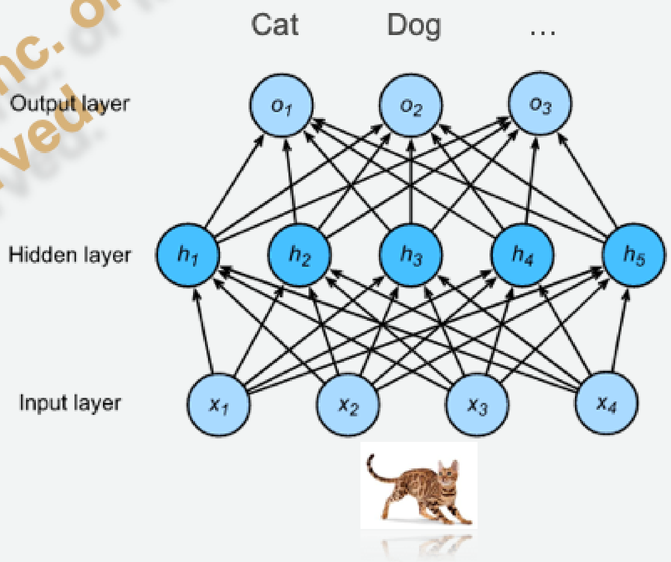
Neural Networks

Perceptron

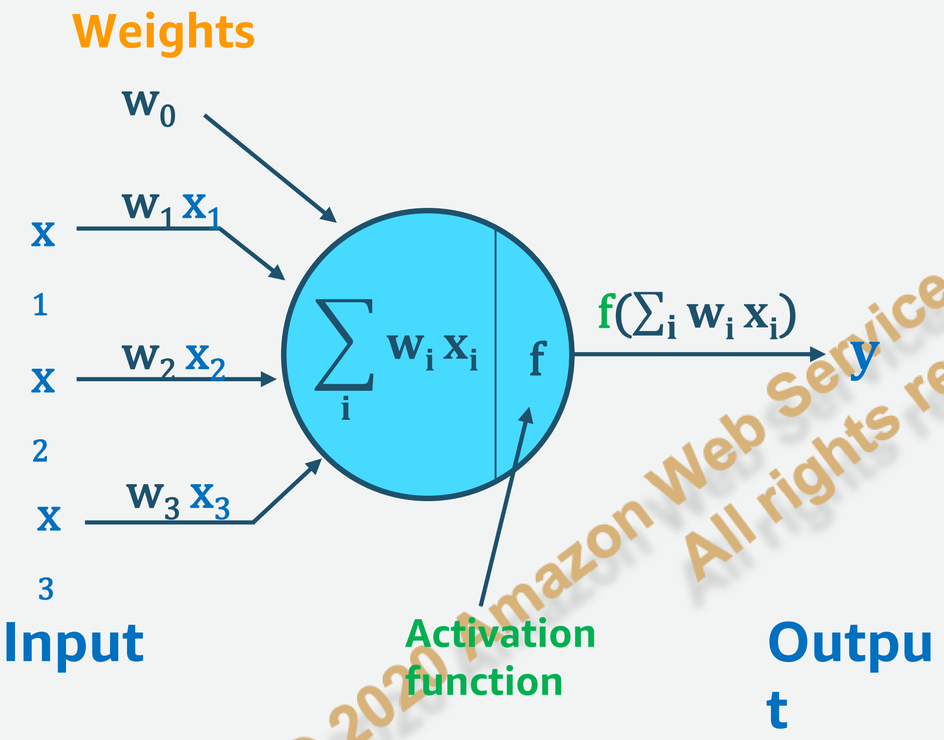
Neuron

Inspired by human brain with billions of connected neurons.





Perceptron



Perceptron*: Given $\{x_i\}$ predict y, where $y \in \{-1, 1\}$:

$$y = f(w_0 + w_1x_1 + \ \ldots\ + w_mx_m),$$

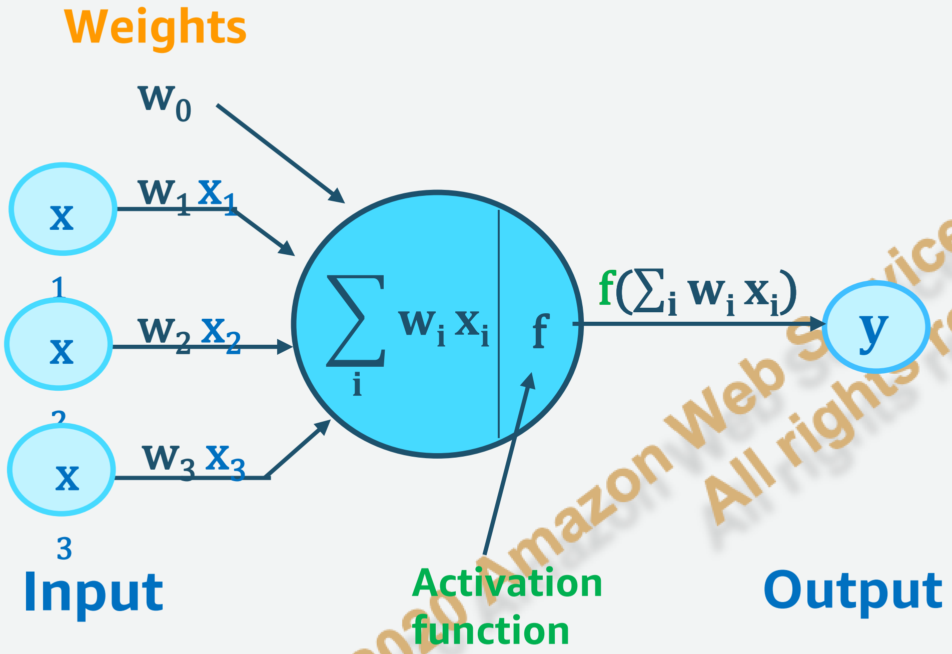
Weighted Sum

here f is the **step function**:

$$f(x) = \begin{cases} 1, \text{if } x \geq 0 \\ -1, \text{if } x < 0 \end{cases}$$

** Another non-linear activation function*

Artificial Neuron



Artificial Neuron*: Given $\{x_i\}$ predict y:

$$y = f(w_0 + w_1 x_1 + \ldots + w_m x_m),$$

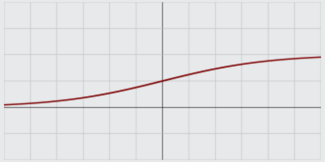
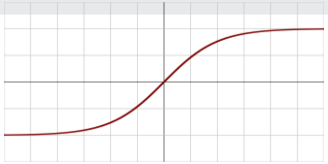
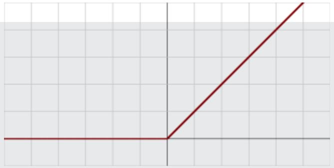
Weighted Sum

here f is a **nonlinear activation function** (sigmoid, tanh, ReLU, etc.)

** Similar to how neurons in the brain function*

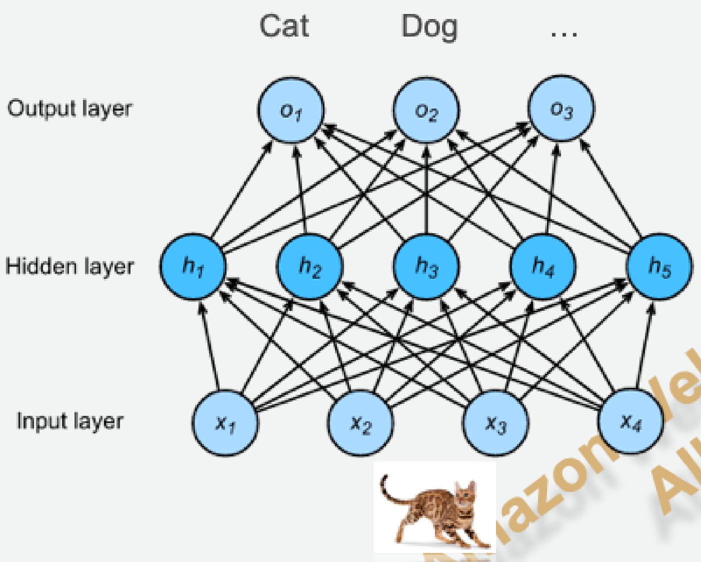
aws

Activation Functions

Name	Plot	Description
Logistic (sigmoid)		• Range: (0,1) • Outputs always greater than 0 • Computationally **expensive** • Cons: vanishing gradients
Hyperbolic tangent (tanh)		• Range: (-1,1) • **0 centered** • Computationally **expensive** • Cons: vanishing gradients
Rectified Linear Unit (ReLU)		• Range: (0,+∞) • Output always greater than 0 • Computationally **cheap** • Cons: "dead" neuron when inputs smaller than 0

Neural Networks Training

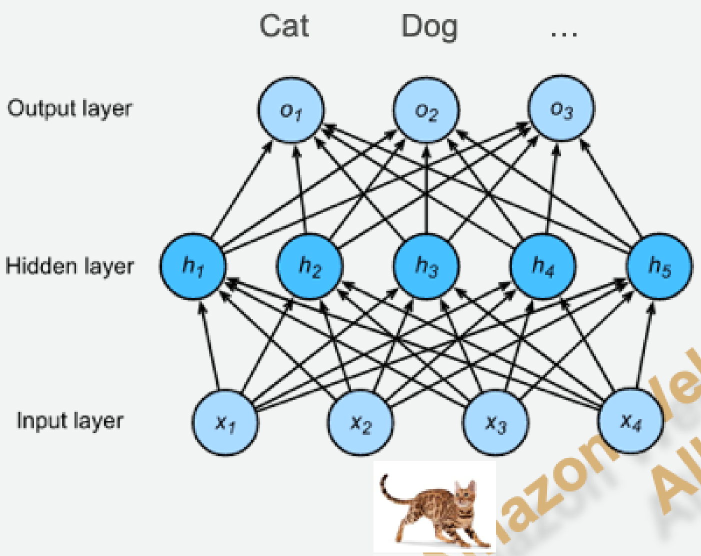
Multilayer Perceptron (MLP)



A standard *Neural Network* (or a *Multilayer Perceptron*):

- Consisting of **input**, **hidden** and **output** layers
- Layers are connected
- An activation function is applied on each hidden layer
- Can be "trained" to perform tasks
- More details

Forward Propagation



$\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$

$\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

- Given weights

$\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

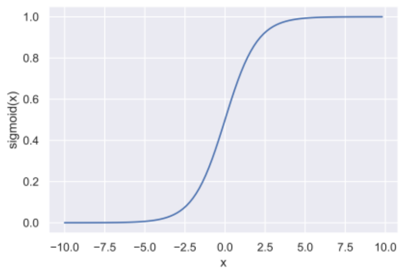
$$\mathbf{y} = \mathrm{softmax}(o)$$

- Input
- Hidden
- Output

Output Function

"How to output/predict a result"

- **Binary classification: Sigmoid**
  - Outputs P(target class | x) in (0,1)
  - Logistic Regression of output of last layer



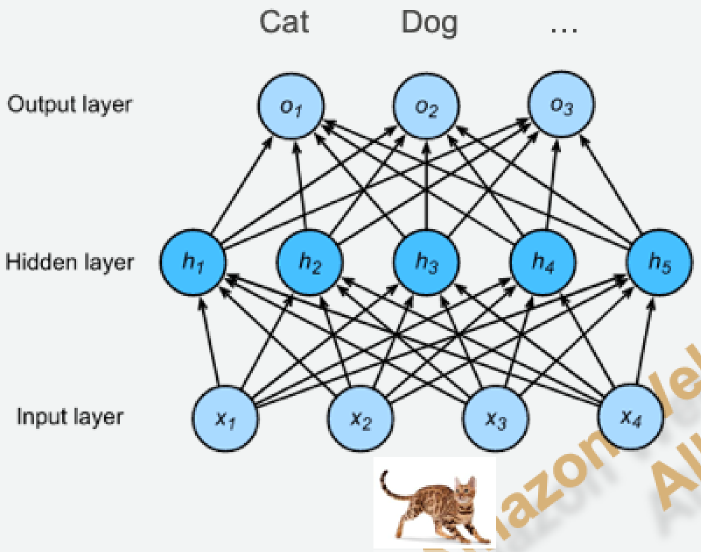
$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

- **Multi-class classification: Softmax**
  - Still want probability for each class output in (0,1)
  - Want sum of output to be 1 (probability distributi
  - Training drives value for target class up, others down

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

- **Regression:** Output activation can be linear or ReLU

Forward Propagation



$\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$

$\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

- Given weights

$\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \mathrm{softmax}(o)$$

- Input
- Hidden
- Output

Cost Functions

"How to compare the outputs with the truth?"

- **Binary classification:** Cross entropy for logistic

$$C = -\frac{1}{n} \sum_{\text{examples}} y \ln p + (1 - y) \ln(1 - p)$$

- **Multiclass classification:** Cross entropy for Softmax

$$C = -\frac{1}{n} \sum_{\text{examples}} \sum_{\text{classes}} y_j \ln p_j$$

- **Regression:** Mean Squared Error:

$$C = \frac{1}{n} \sum_{\text{examples}} (y - p)^2$$

Notation for Classification

- n = training examples
- j = classes
- p = prediction (probability)
- y = true class (1/yes, 0/no)

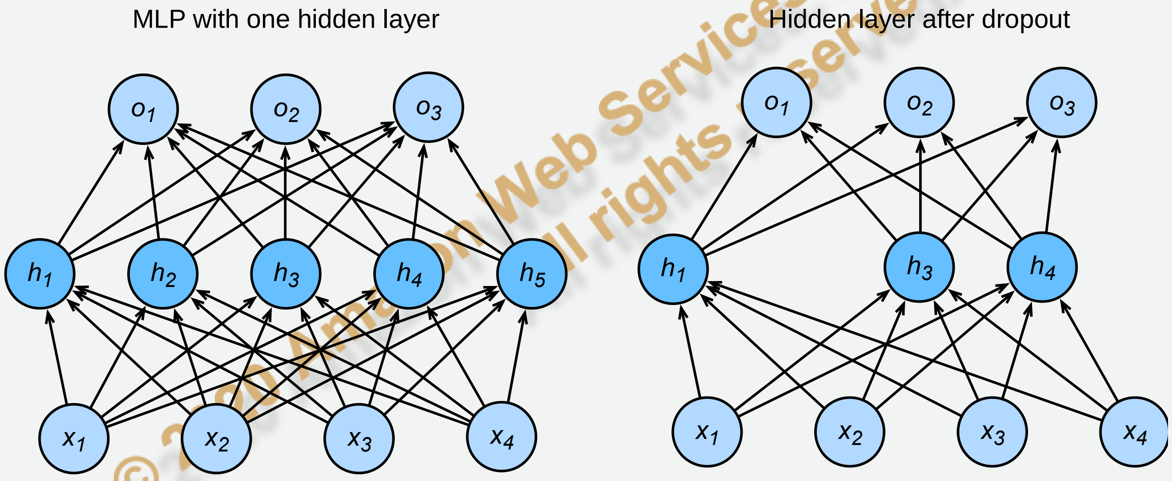
Notation for Regression

- n = training examples
- p = prediction (numeric, $\hat{y}$)
- y = true value

Dropout

Regularization technique to **prevent overfitting**.

Randomly removes some nodes with a fixed probability during the training.



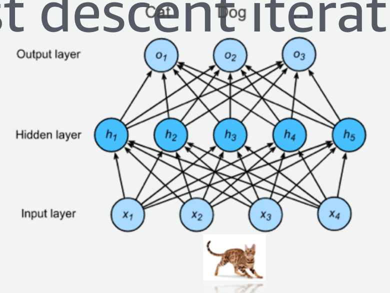
[More details]

Backpropagation

Gradient descent:

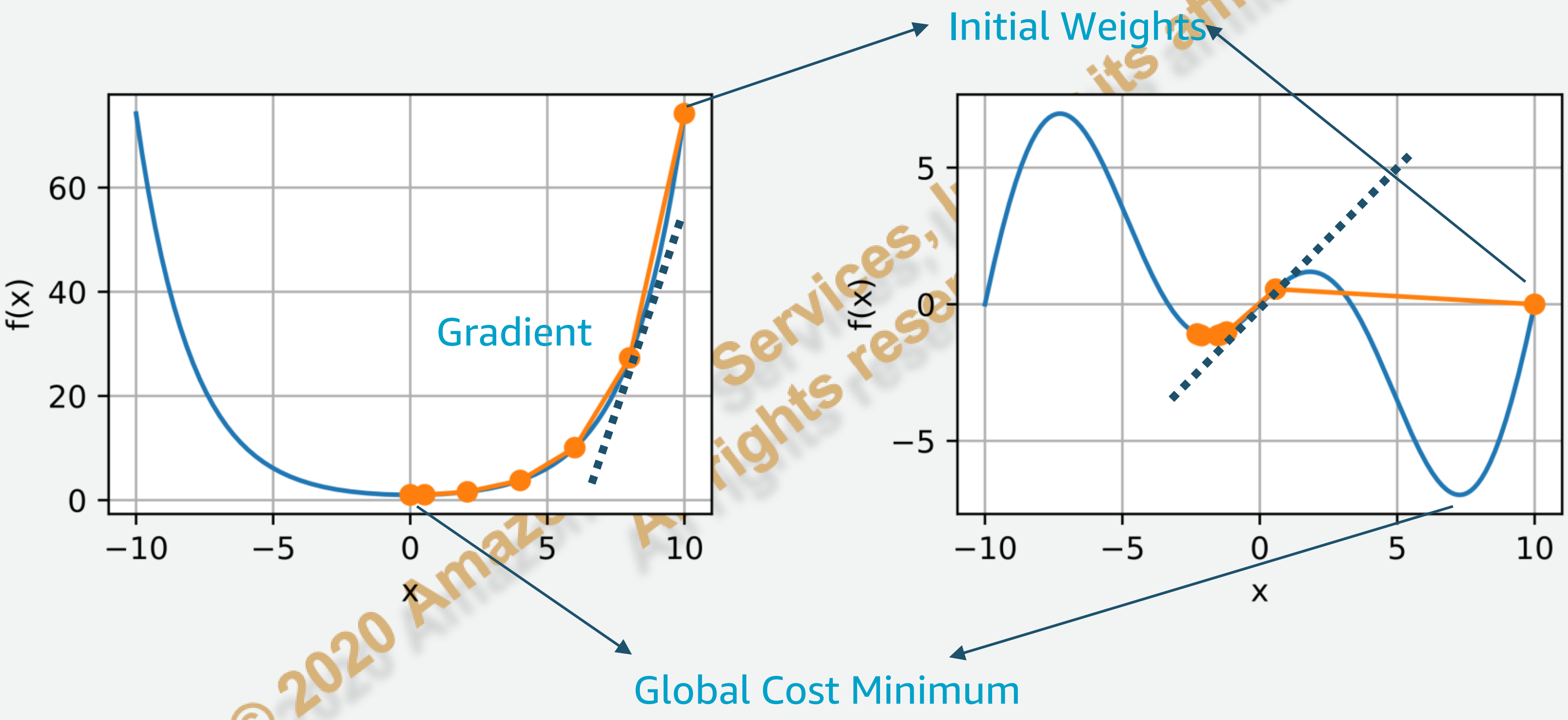
- Optimization method used to ‘learn’/‘train’ neural networks
- Finds the **minimum** of an **objective/cost** function **C(w)** by moving in the direction of the steepest descent iteratively.
- At each update:

$$w := w - \text{learning\_rate} * \frac{\partial C}{\partial w}$$

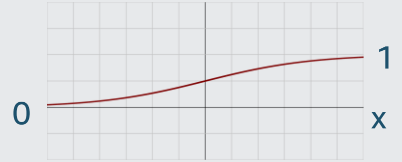
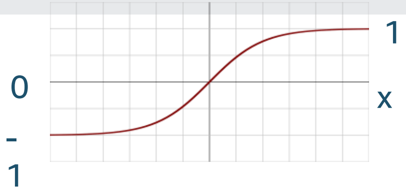
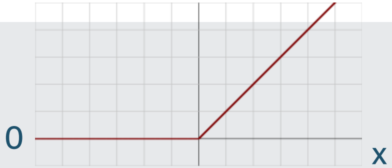


Not too small, not too large… Gradient with respect to w

Gradient Descent: Learning Rate



Review: Activation Functions

Name	Plot	Description
Logistic (sigmoid)		• Range: (0,1) • Outputs always greater than 0 • Computationally **expensive** • Cons: vanishing gradients
Hyperbolic tangent (tanh)		• Range: (-1,1) • **0 centered** • Computationally **expensive** • Cons: vanishing gradients
Rectified Linear Unit (ReLU)		• Range: (0,+∞) • Output always greater than 0 • Computationally **cheap** • Cons: "dead" neuron when inputs smaller than 0

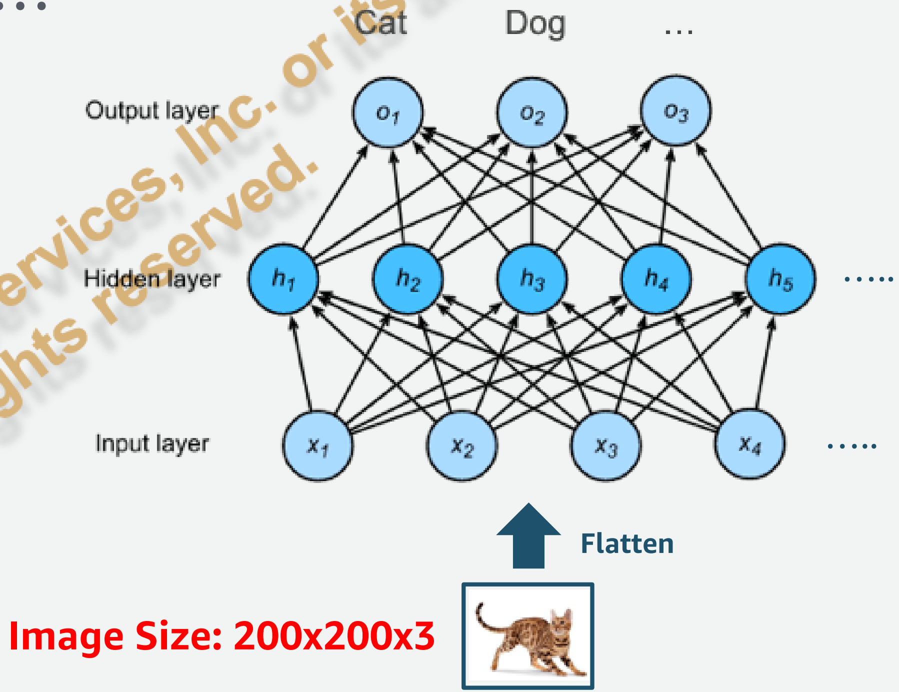
Neural Networks
Convolutions & Poolings

Problem of MLPs

Too many parameters to train…

 # inputs: 200x200x3
=120,000

 # units: 100
 # parameters:
100x120,000
=12 million



Can we reduce parameters?

Where is Waldo?

Our vision system…

1. Translation Invariance:

Our vision systems should, in some sense, **respond similarly to the same object regardless of where it appears** on the image.

Our vision system…

2. Locality:

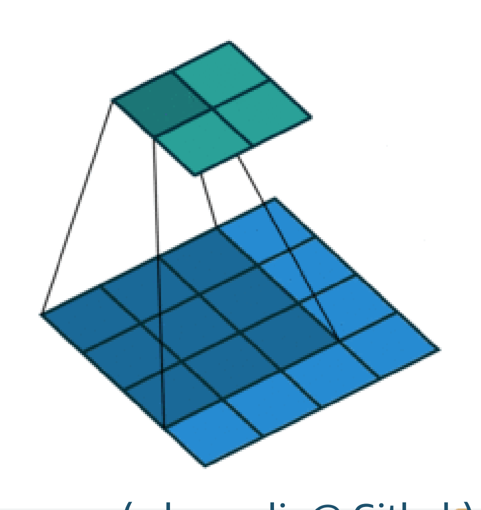
Our visions systems should, in some sense, **focus on somewhat local regions**, without regard for what else is happening on the image at greater distances.

The idea of convolution

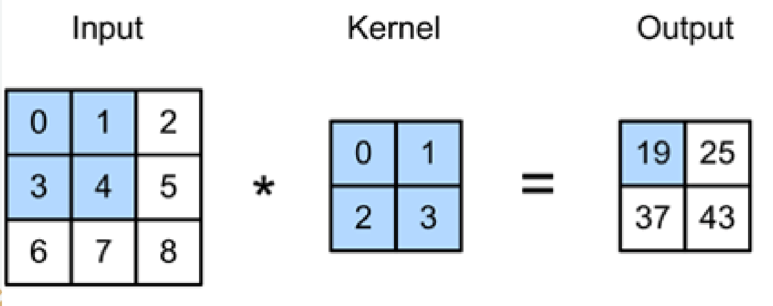
- Each pixel on the image can be realized as $x_{i,j}$
- Each "weight" on the filter window can be defined by $v_{a,b}$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

2D Convolution Example

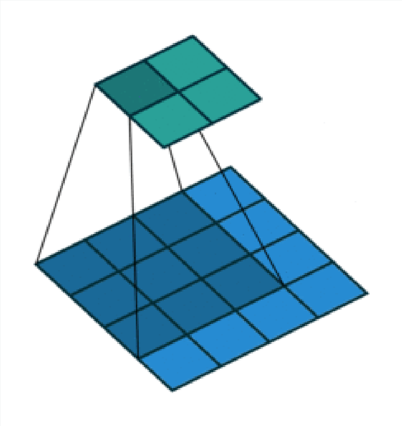


(vdumoulin@ Github)



$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

2D Convolution Example



(vdumoulin@ Github)

Input * Kernel = Output

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

- $\mathbf{X}$: $n_h \times n_w$ input matrix
- $\mathbf{W}$: $k_h \times k_w$ kernel matrix
- b : the bias scalar
- $\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- W and b are the trainable parameters

Kernels (Filters)

(wikipedia)

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

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

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

What will happen after the filters?

Kernels (Filters)

(wikipedia)

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

Edge Detection

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

Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

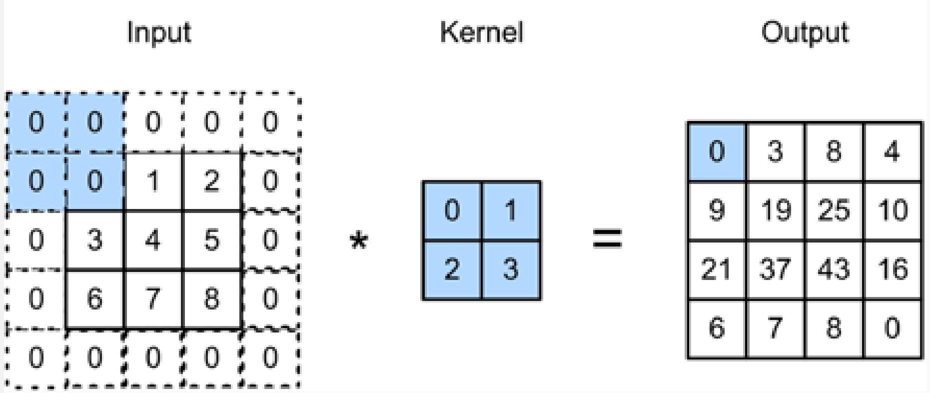
Gaussian Blur

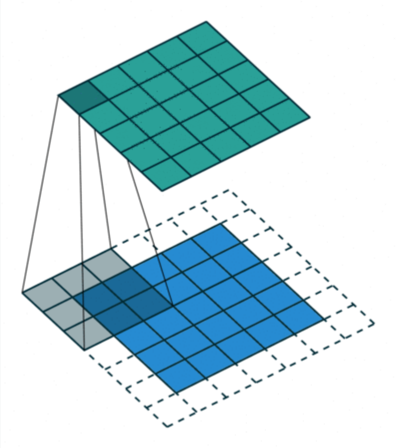
What do we do near the boundary?

The original convolution window may ignore this Waldo at the boundary …

Padding

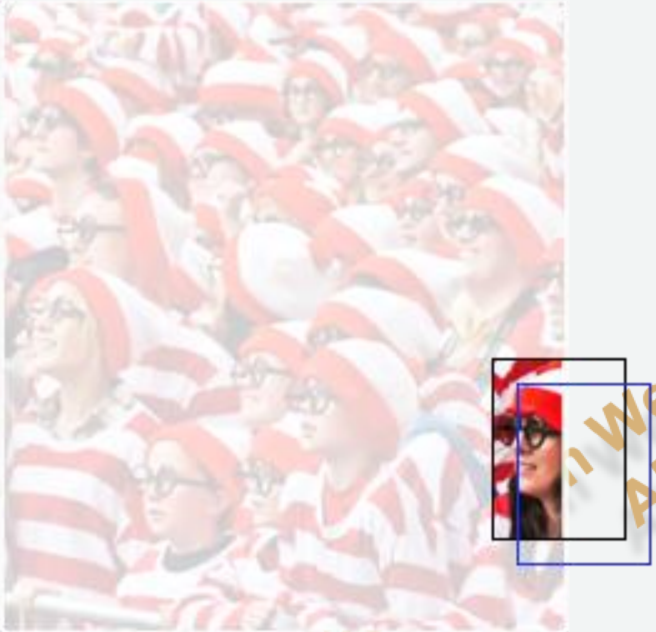
Padding adds rows/columns around the input.





(vdumoulin@ Github)

How about two nearly identical windows?



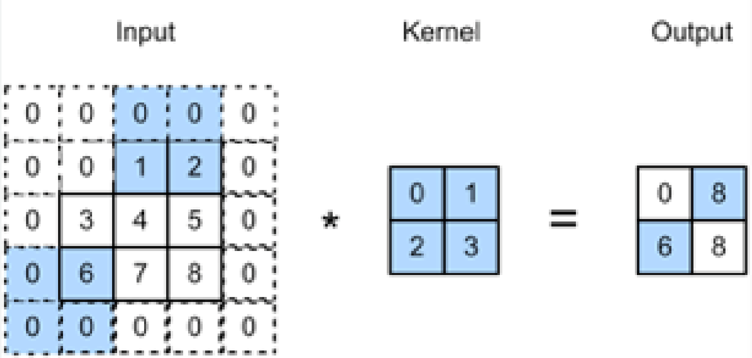
The original convolution window may

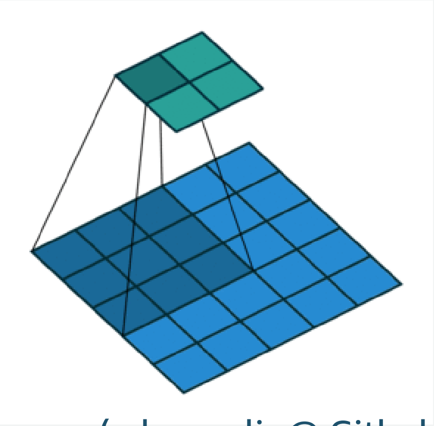
be too computational expensive to slide one pixel at a time…

Stride

Stride is the number of "unit" the kernel shifted per slide over rows/columns. E.g.,

Strides of 3 for height and 2 for width

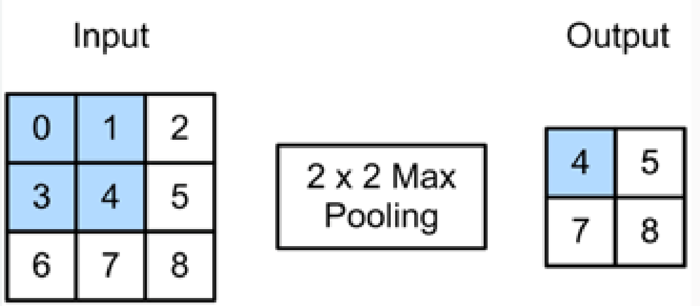




(vdumoulin@ Github)

Pooling

Pooling is used to reduce size (height and width) of the feature map.



Max Pooling: Returns the maximal value in the pooling window

Average Pooling: Returns the average in the window

Why Pooling?

Compared with convolutions, pooling progressively reduce

- the spatial size of the representation
- the amount of parameters and computation

Pooling can operate with padding and stride.

Thank you!