Introduction to Deep Learning

https://github.com/kaopanboonyuen/GISTDA2023

Reference:

1. https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html
2. http://introtodeeplearning.com/
3. https://www.simplilearn.com/tutorials/deep-learning-tutorial/introduction-to-deep-learning
4. https://www.geeksforgeeks.org/introduction-deep-learning/
5. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

PyTorch Streamlit

2 Artificial Intelligence Projects

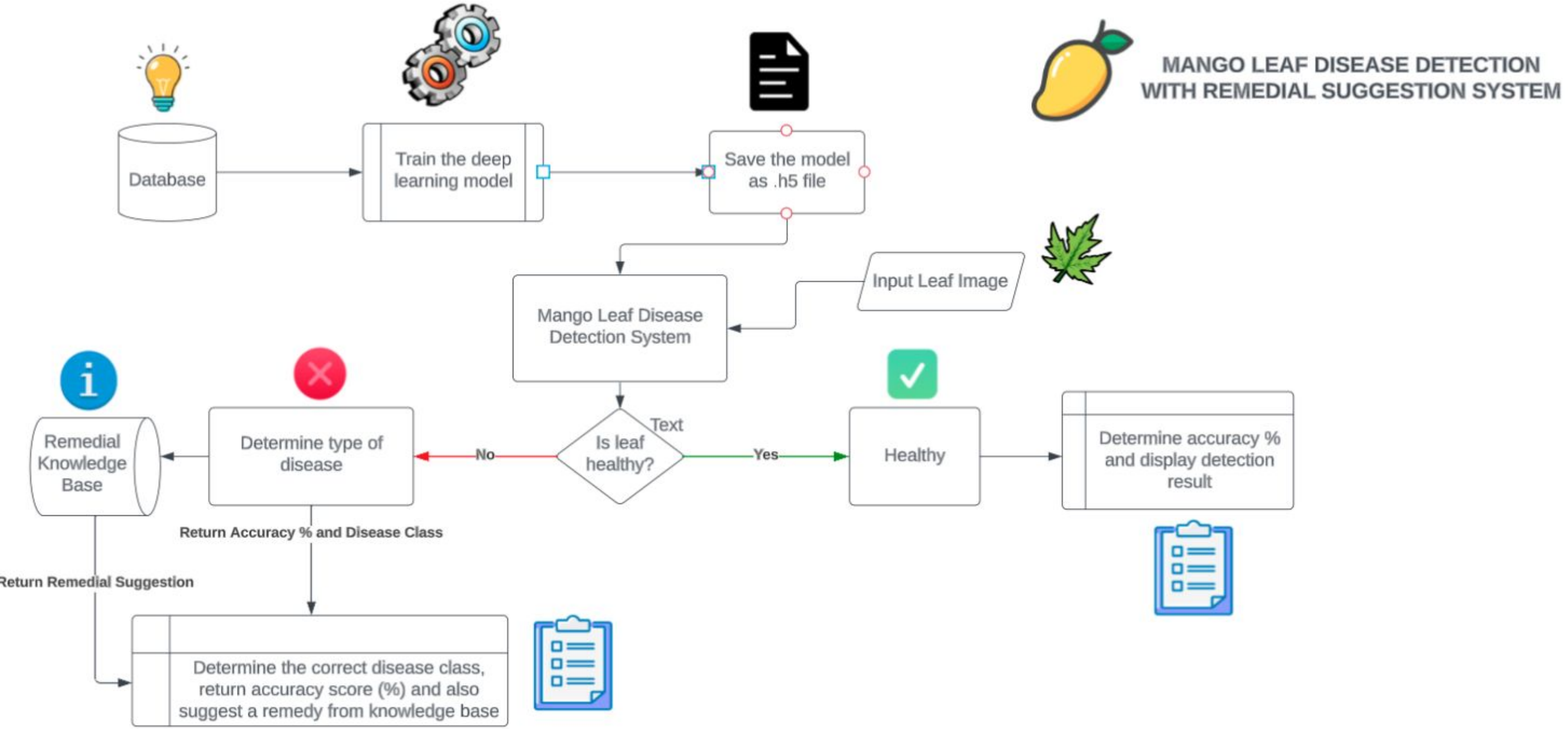
Day1 - Mango🥭 Leaf🍃🍂 Disease Dataset

Day2 - Airbus Aircraft Detection

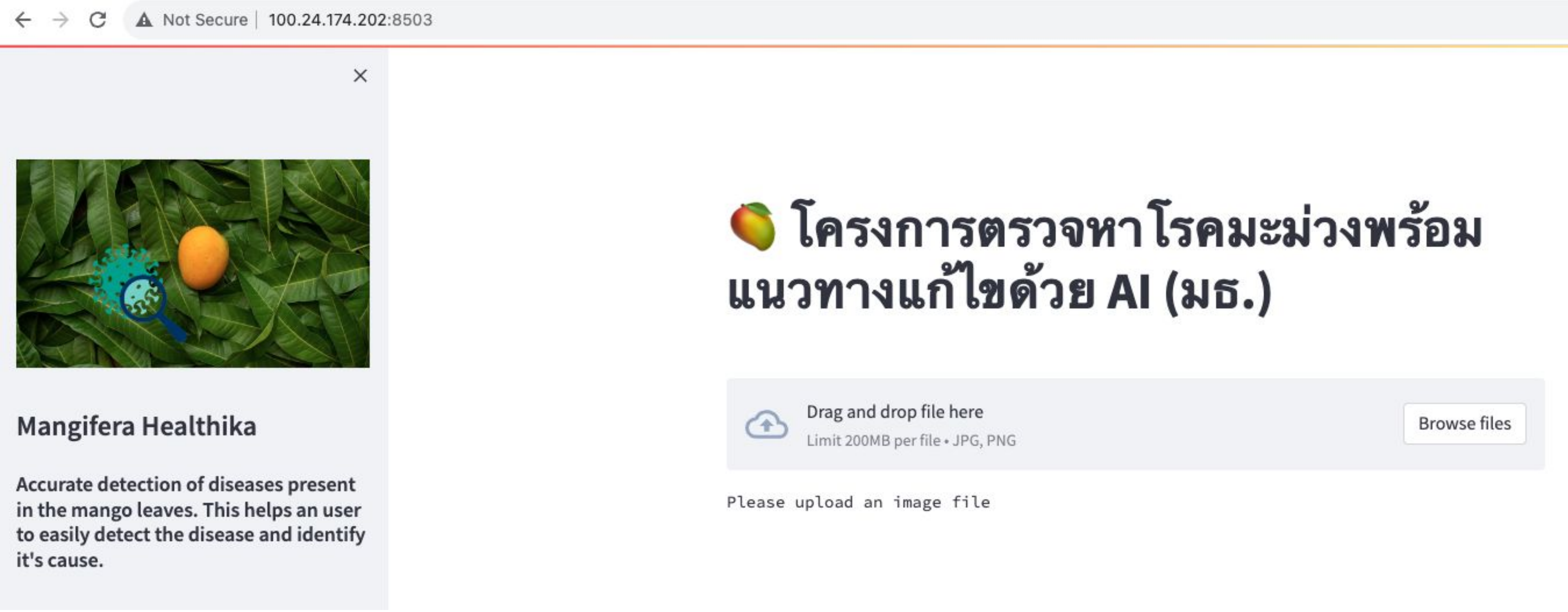
(Tutorial) Fashion-MNIST

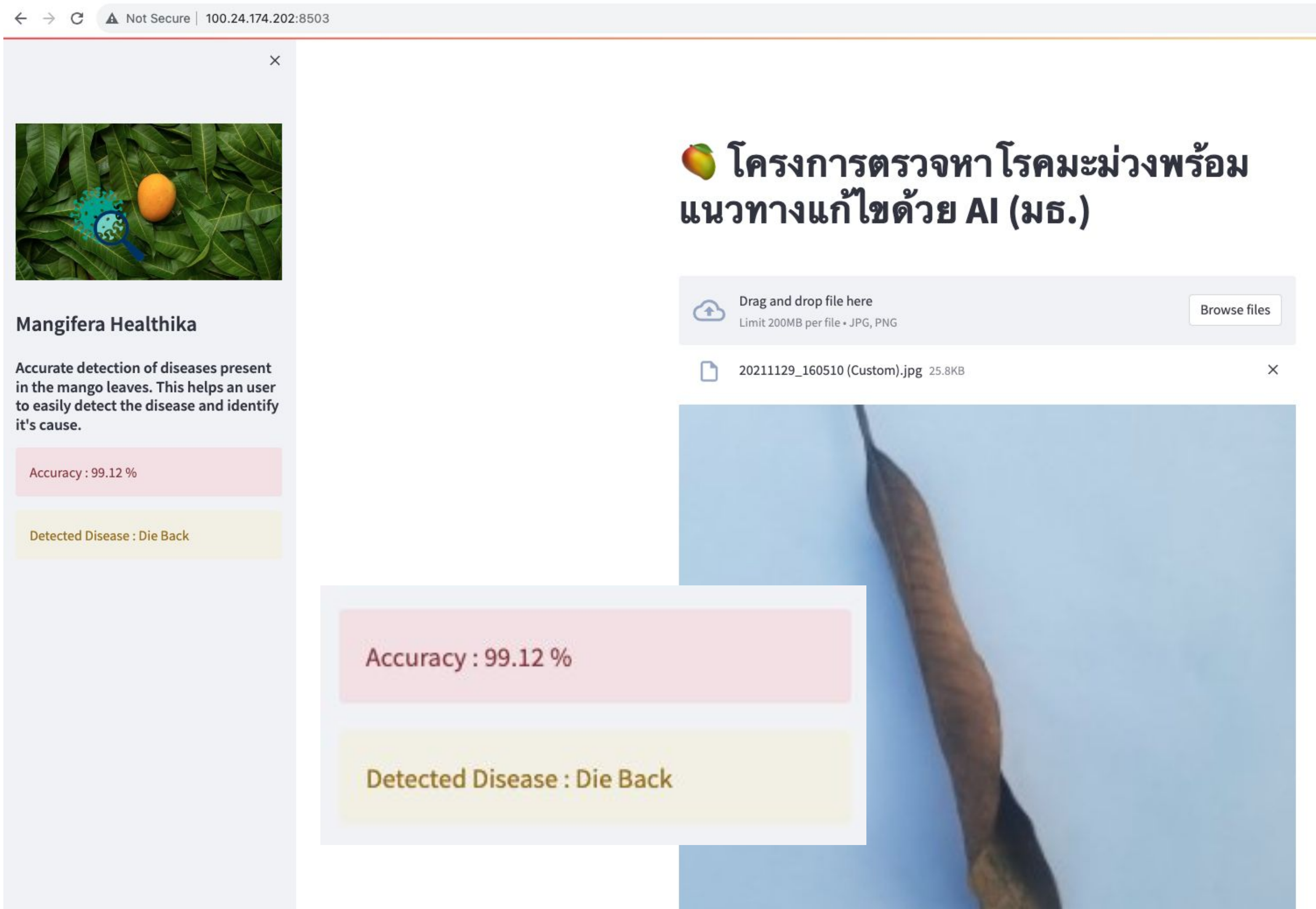
- https://github.com/zalandoresearch/fashion-mnist#labels
- Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot


MANGO LEAF DISEASE DETECTION
WITH REMEDIAL SUGGESTION SYSTEM
Database
Train the deep learning model
Save the model as .h5 file
Input Leaf Image
Mango Leaf Disease Detection System
Is leaf healthy?
Text
No
Yes
Healthy
Determine accuracy % and display detection result
Determine type of disease
Remedial Knowledge Base
Return Accuracy % and Disease Class
Return Remedial Suggestion
Determine the correct disease class, return accuracy score (%) and also suggest a remedy from knowledge base

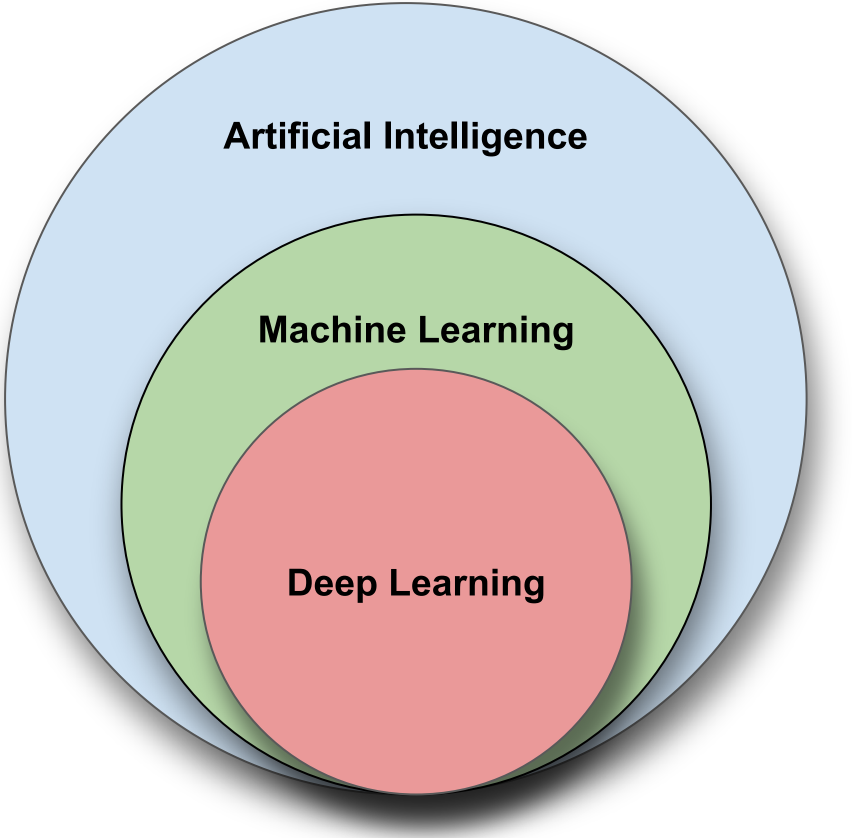
Deploy your AI model to web application



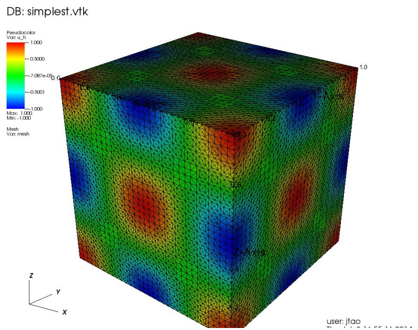

Not Secure | 100.24.174.202:8503
Mangifera Healthika
Accurate detection of diseases present in the mango leaves. This helps an user to easily detect the disease and identify it's cause.
Accuracy : 99.12 %
Detected Disease : Die Back
โครงการตรวจหาโรคมะม่วงพร้อมแนวทางแก้ไขด้วย AI (มธ.)
Drag and drop file here
Limit 200MB per file • JPG, PNG
Browse files
20211129_160510 (Custom).jpg 25.8KB
Accuracy : 99.12 %
Detected Disease : Die Back

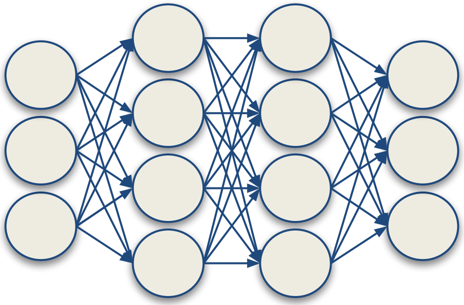
Relationship of AI, ML and DL

- **Artificial Intelligence (AI)** is anything about man-made intelligence exhibited by machines.
- **Machine Learning (ML)** is an approach to achieve **AI**.
- **Deep Learning (DL)** is one technique to implement **ML**.

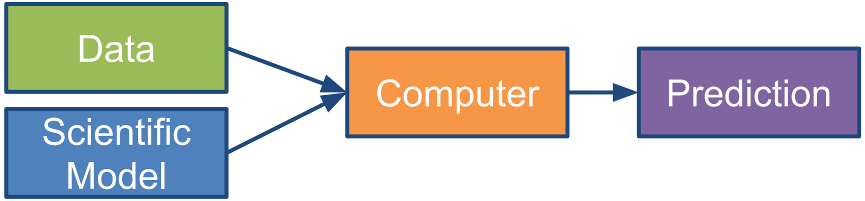


Machine Learning

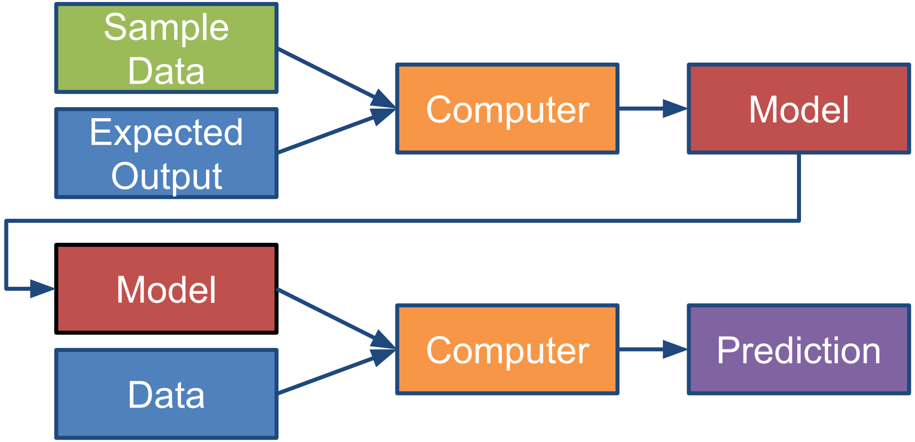




Traditional Modeling

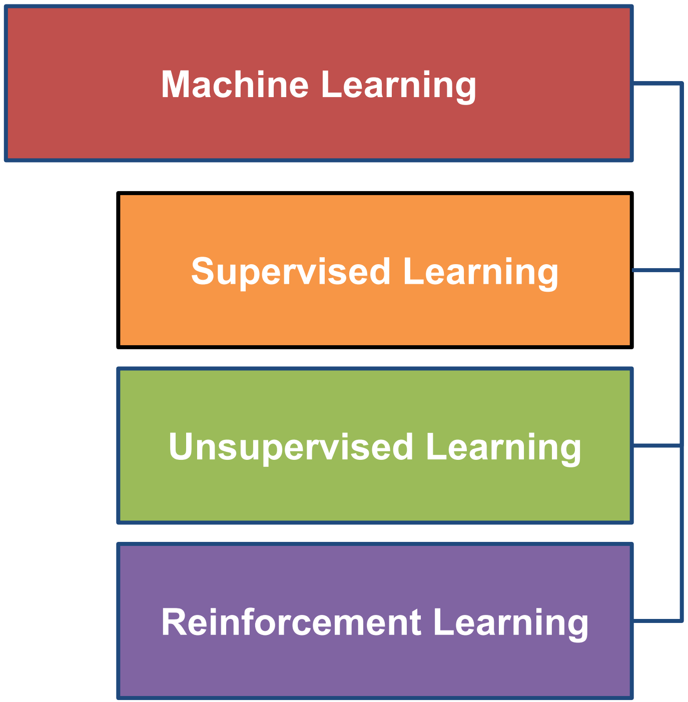


Machine Learning (Supervised Learning)



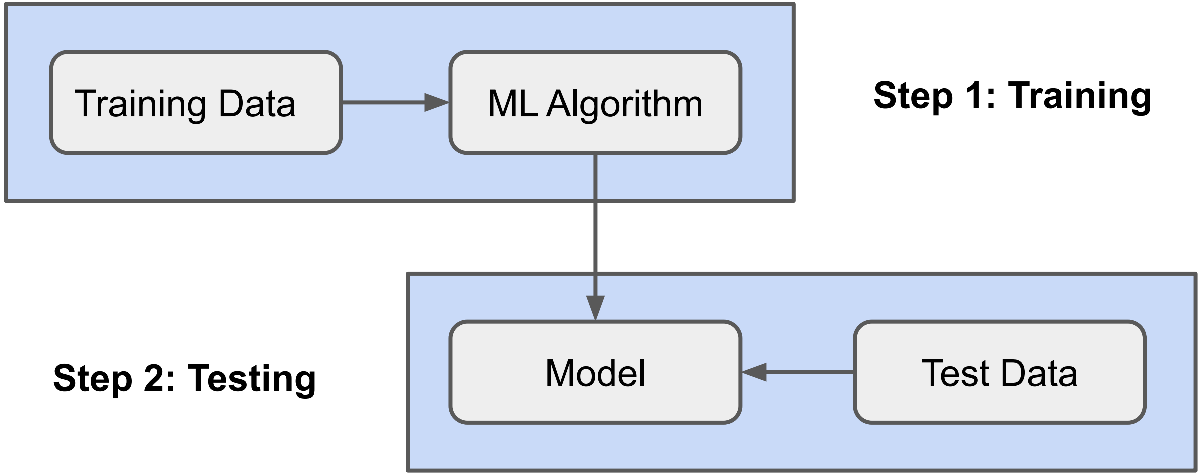
Types of ML Algorithms

- **Supervised Learning**
  - trained with labeled data; including regression and classification problems
- **Unsupervised Learning**
  - trained with unlabeled data; clustering and association rule learning problems.
- **Reinforcement Learning**
  - no training data; stochastic Markov decision process; robotics and self-driving cars.



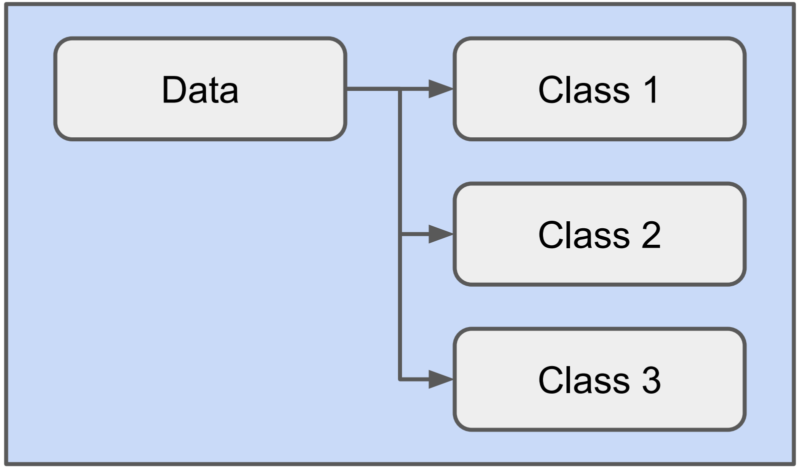
Supervised Learning

When both input variables - X and output variables - Y are known, one can approximate the mapping function from X to Y.



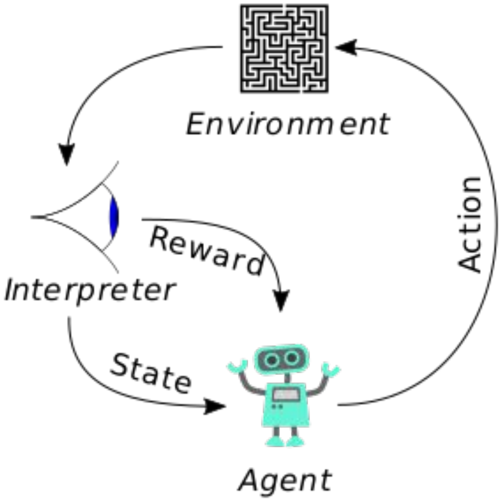
Unsupervised Learning

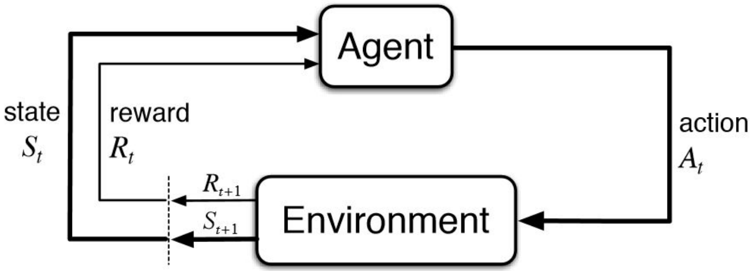
When only input variables - X are known and the training data is neither classified nor labeled. It is usually used for clustering problems.



Reinforcement Learning

When the input variables are only available via interacting with the environment, reinforcement learning can be used to train an "**agent**".





(Image Credit: Wikipedia.org)

(Image Credit: deeplearning4j.org)

Why Deep Learning?

- Limitations of traditional machine learning algorithms
  - not good at handling high dimensional data.
  - difficult to do feature extraction and object recognition.

- Advantages of deep learning
  - DL is computationally expensive, but it is capable of handling high dimensional data.
  - feature extraction is done automatically.

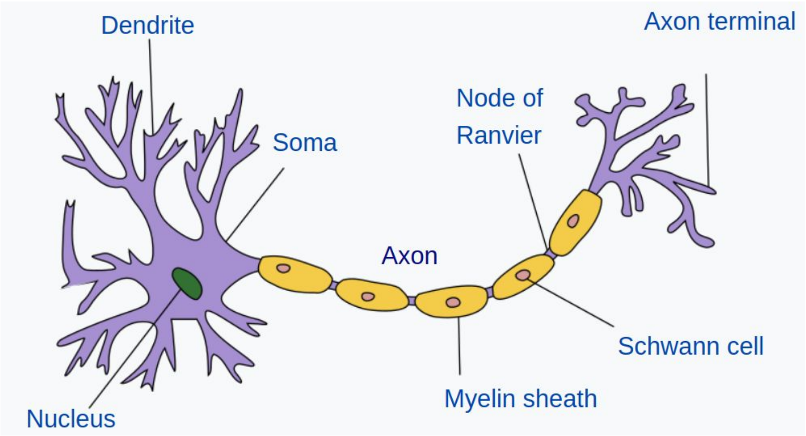
What is Deep Learning?

Deep learning is a class of machine learning algorithms that:

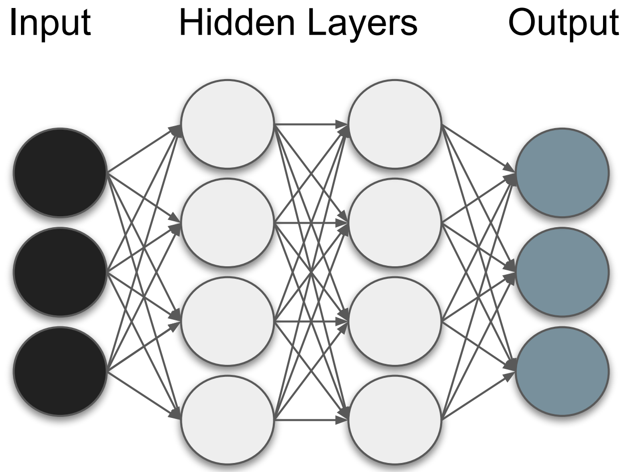
- use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
- learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

(Source: Wikipedia)

Artificial Neural Network



(Image Credit: Wikipedia)



Inputs and Outputs

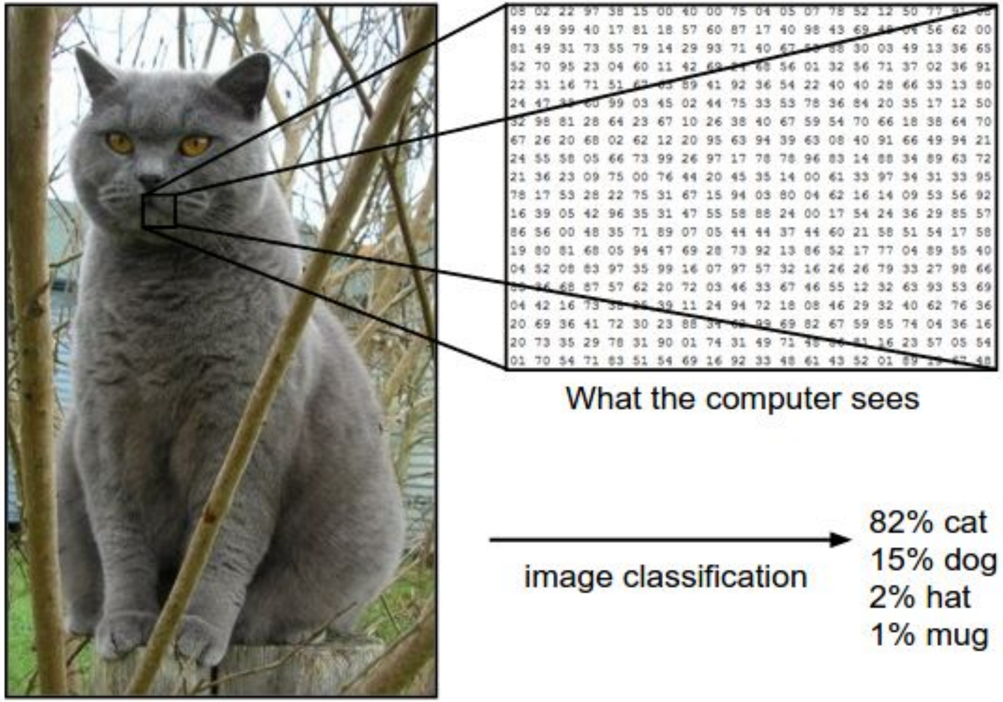
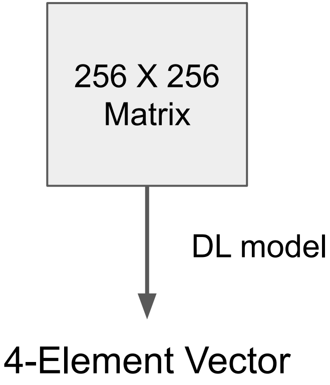
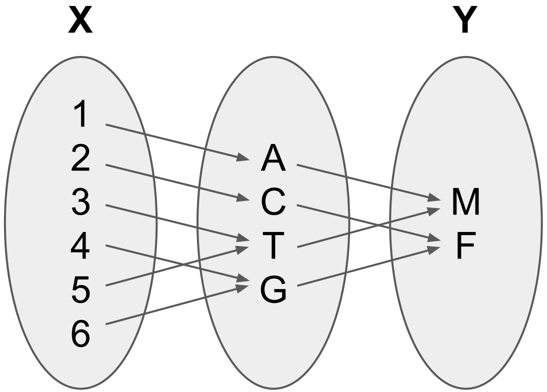


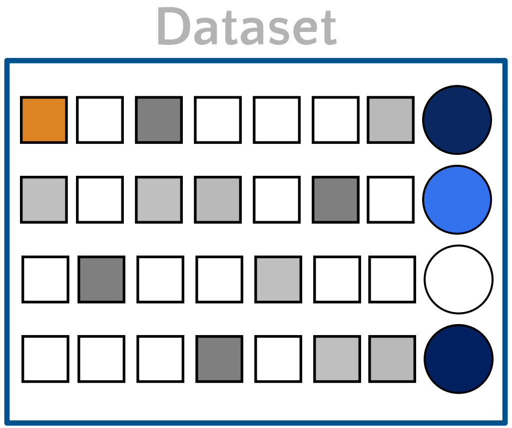
Image from the Stanford CS231 Course

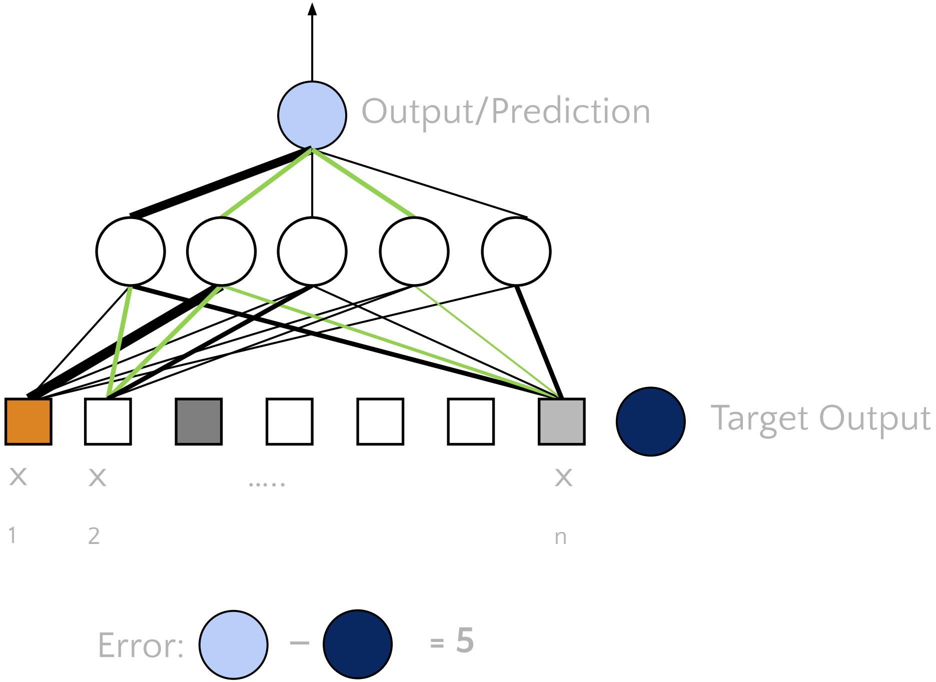




With deep learning, we are searching for a **surjective** (or **onto**) function **f** from a set **X** to a set **Y**.

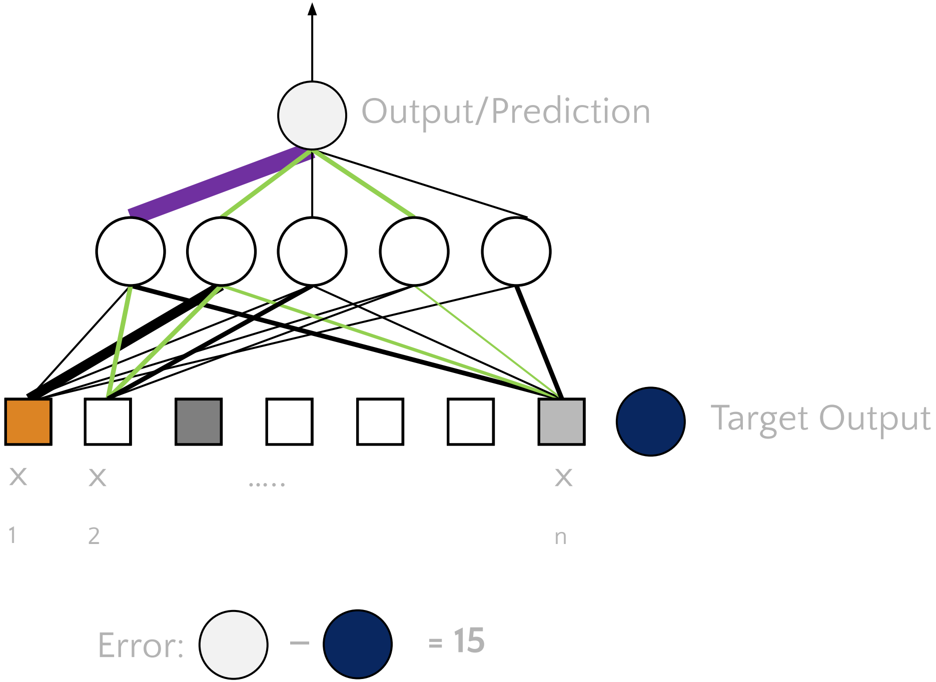
Learning Principle





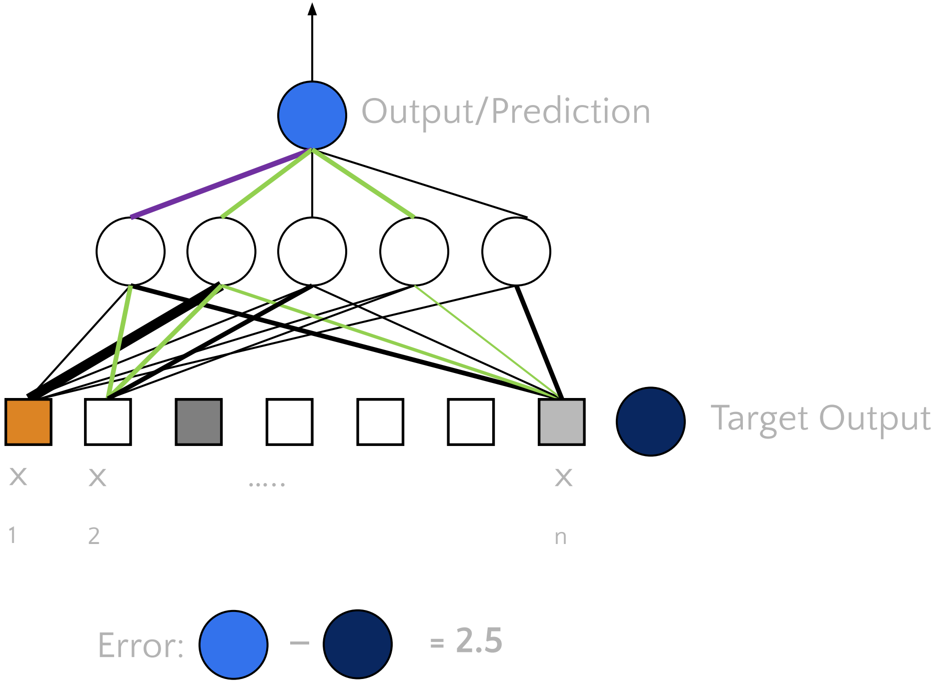
(Image Credit: NVIDIA Deep Learning Institute)

Learning Principle



(Image Credit: NVIDIA Deep Learning Institute)

Learning Principle



(Image Credit: NVIDIA Deep Learning Institute)

Deep Neural Network as a Universal Approximator

Universal Approximation Theorem

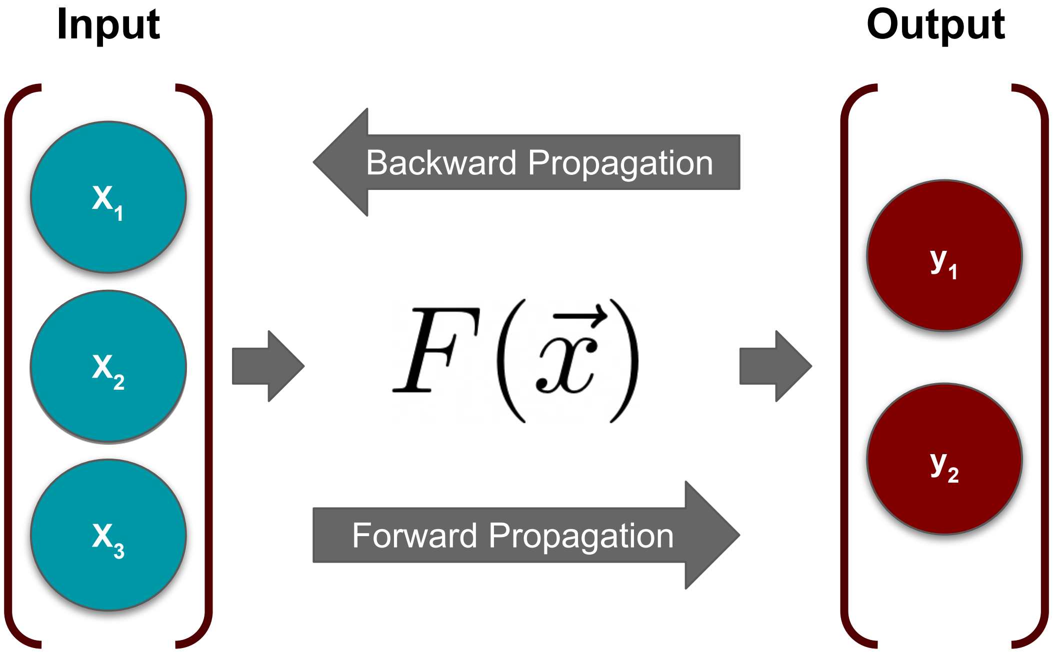
(Cybenko, 1989)

Universal approximation theorems imply that neural networks can represent a wide variety of **functions**.

Pinkus Theorem

(Pinkus, 1999)

Pinkus theorems imply that neural networks can represent **directives of a function** simultaneously.



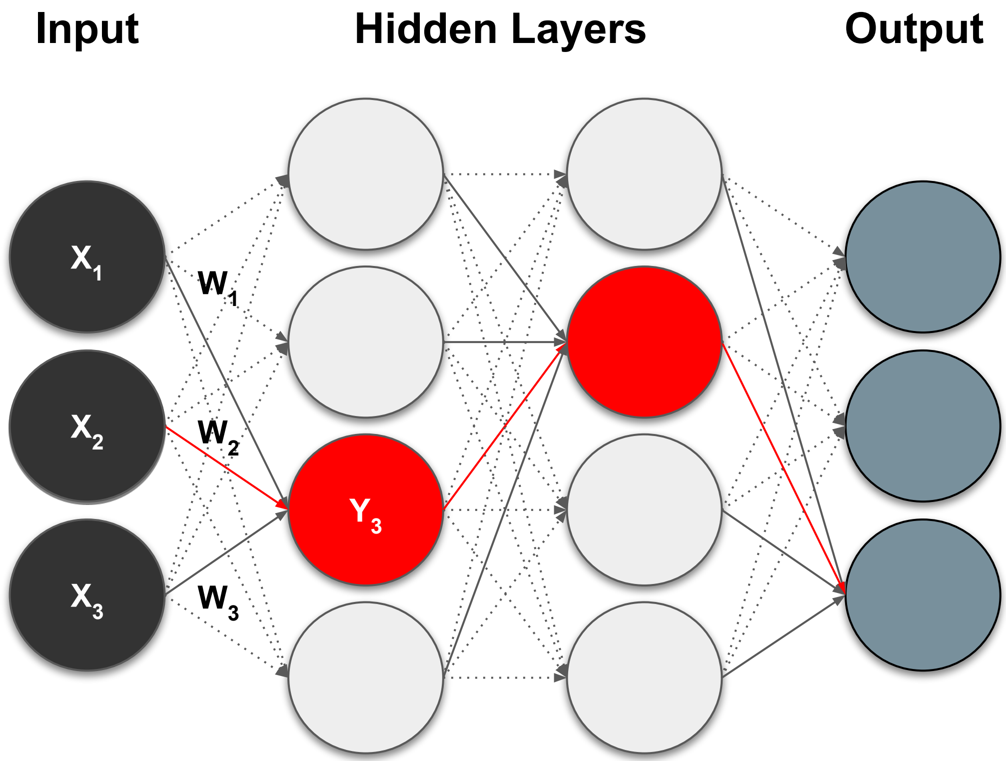
- **Training:** given **input** and **output**, find best-fit ***F***
- **Inference:** given **input** and ***F***, predict **output**

Supervised Deep Learning with Neural Networks

From one layer to the next

$$Y_j = f\left(\sum_i W_i X_i + b_i\right)$$

f is the activation function, W_i is the weight, and b_i is the bias.



Training - Minimizing the Loss

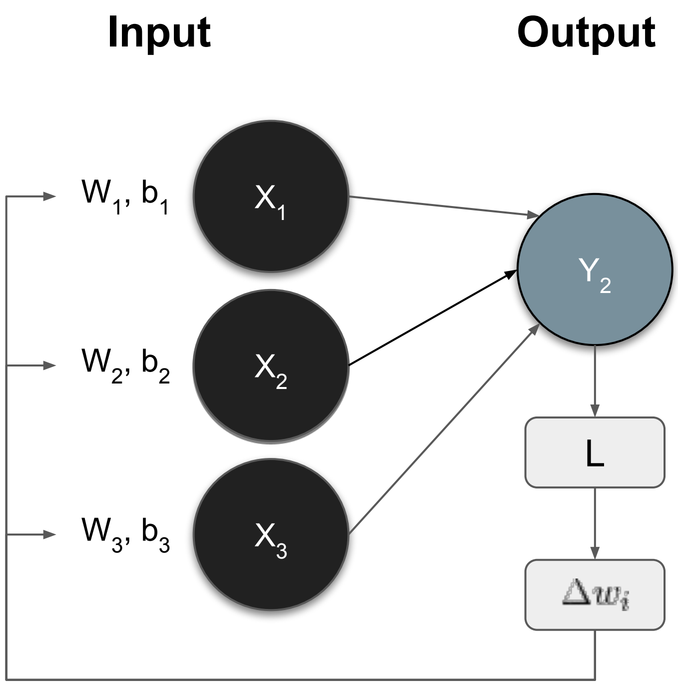
The loss function with regard to weights and biases can be defined as

$$L(\mathbf{w}, \mathbf{b}) = \frac{1}{2} \sum_i (\mathbf{Y}(\mathbf{X}, \mathbf{w}, \mathbf{b}) - \mathbf{Y}'(\mathbf{X}, \mathbf{w}, \mathbf{b}))^2$$

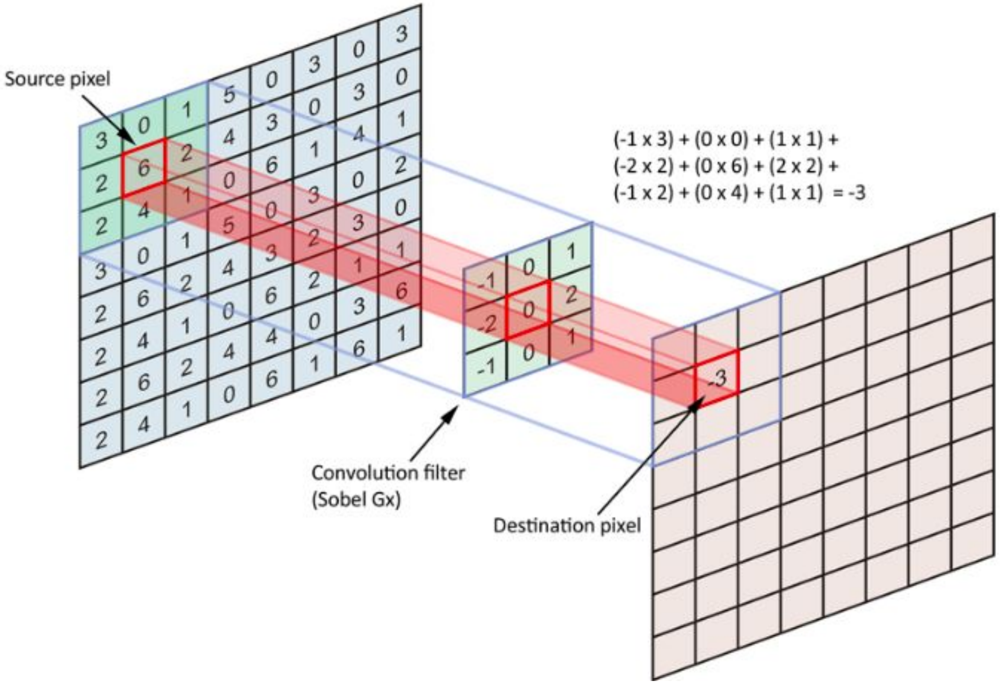
The weight update is computed by moving a step to the opposite direction of the cost gradient.

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

Iterate until L stops decreasing.



Convolution in 2D



(Image Credit: Applied Deep Learning | Arden Dertat)

Convolution Kernel

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

(Image Credit: Applied Deep Learning | Arden Dertat)

Convolution on Image

Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial]

Activation Functions

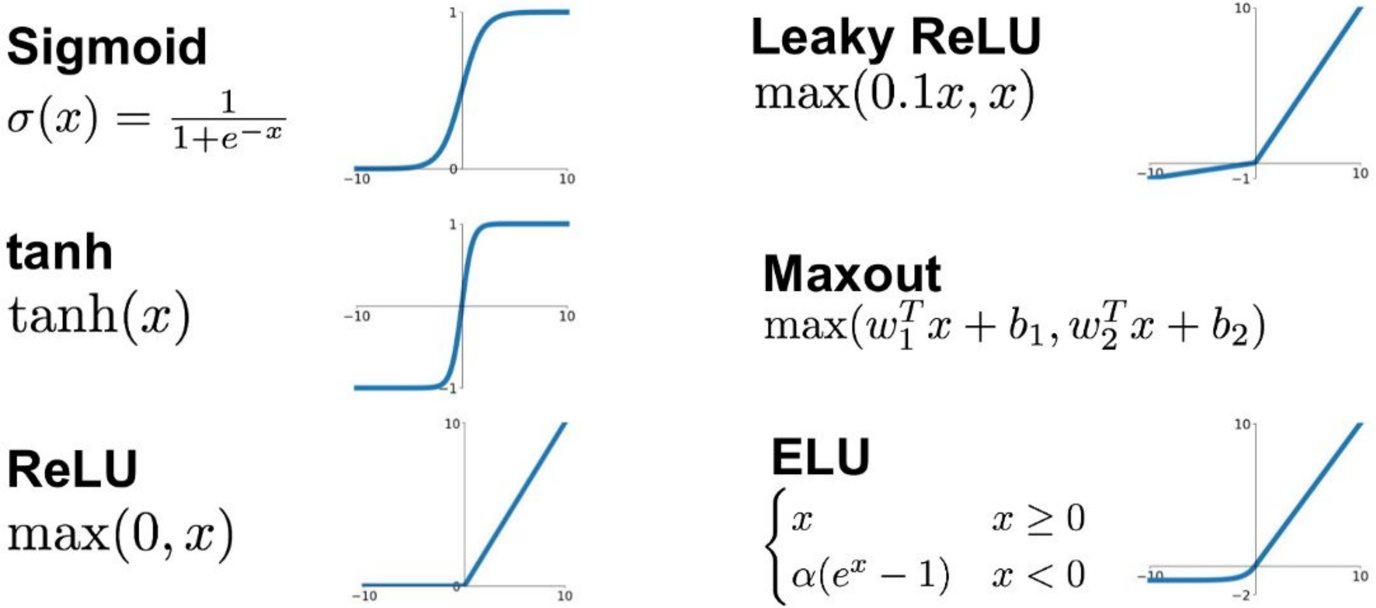


Image Credit: towardsdatascience.com

Introducing Non Linearity (ReLU)

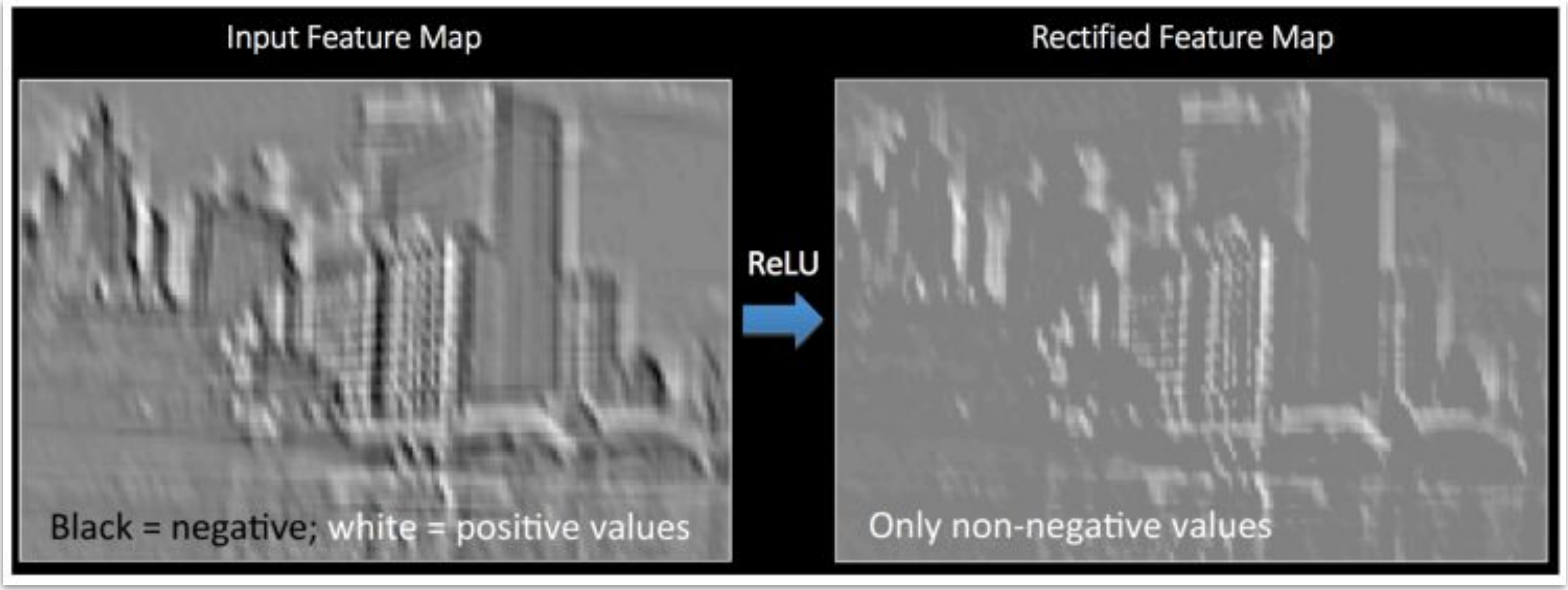
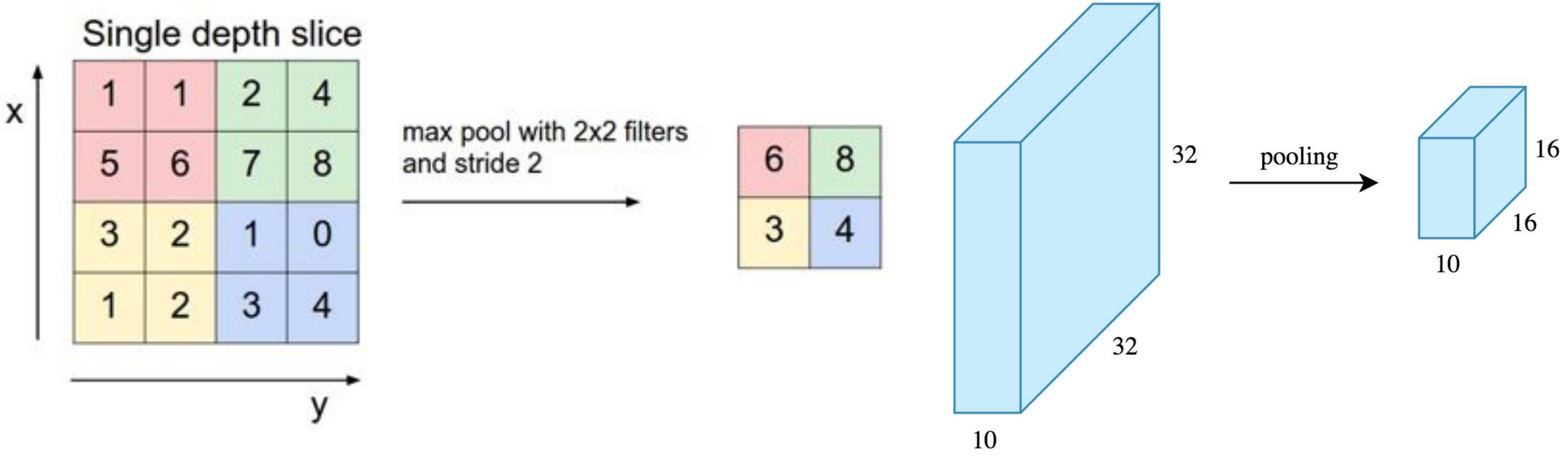


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial]

Max Pooling



(Image Credit: [Applied Deep Learning | Arden Dertat])

Pooling - Max-Pooling and Sum-Pooling

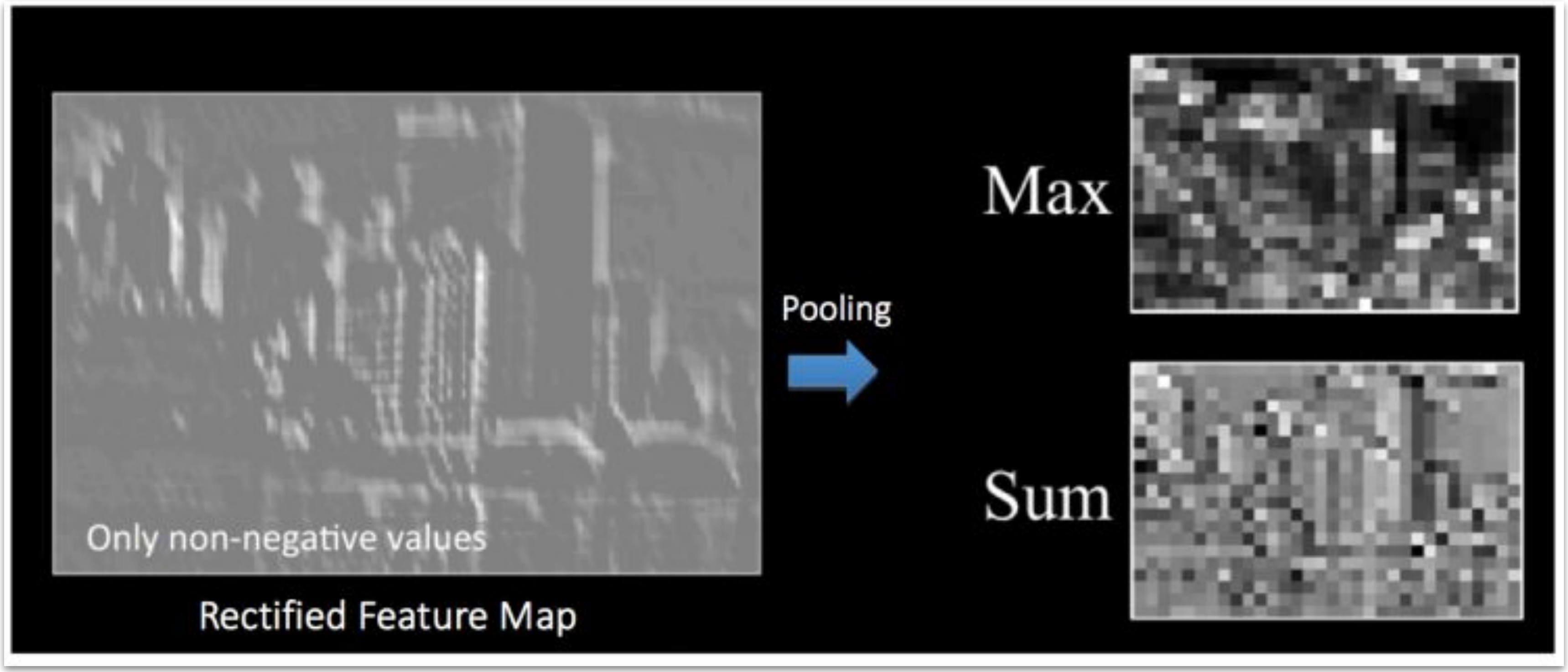
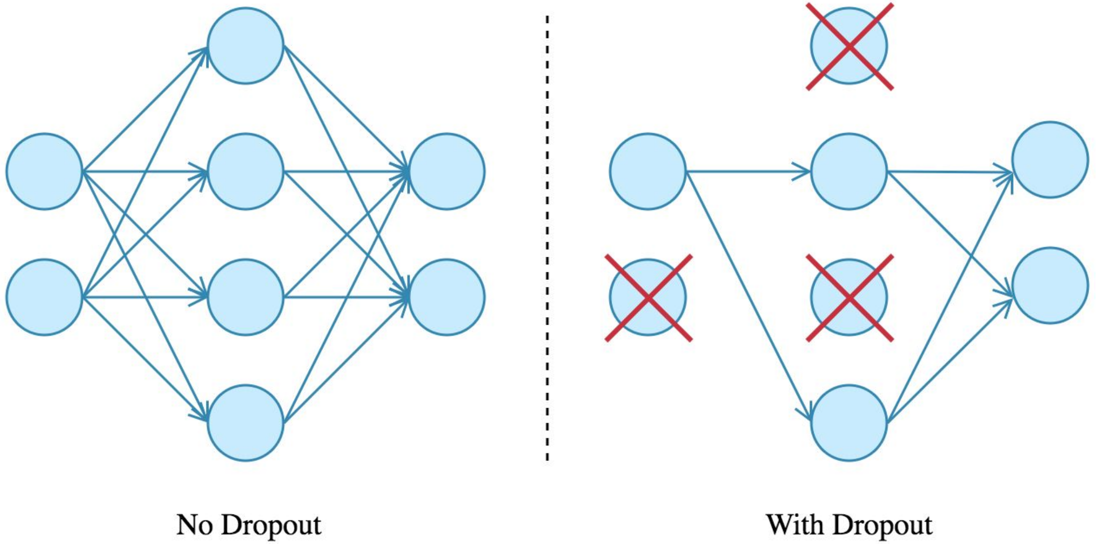


Image Credit: [Deep Learning Methods for Vision | CVPR 2012 Tutorial]

CNN Implementation - Drop Out

Dropout is used to prevent overfitting. A neuron is temporarily "dropped" or disabled with probability P during training.



(Image Credit: Applied Deep Learning | Arden Dertat)

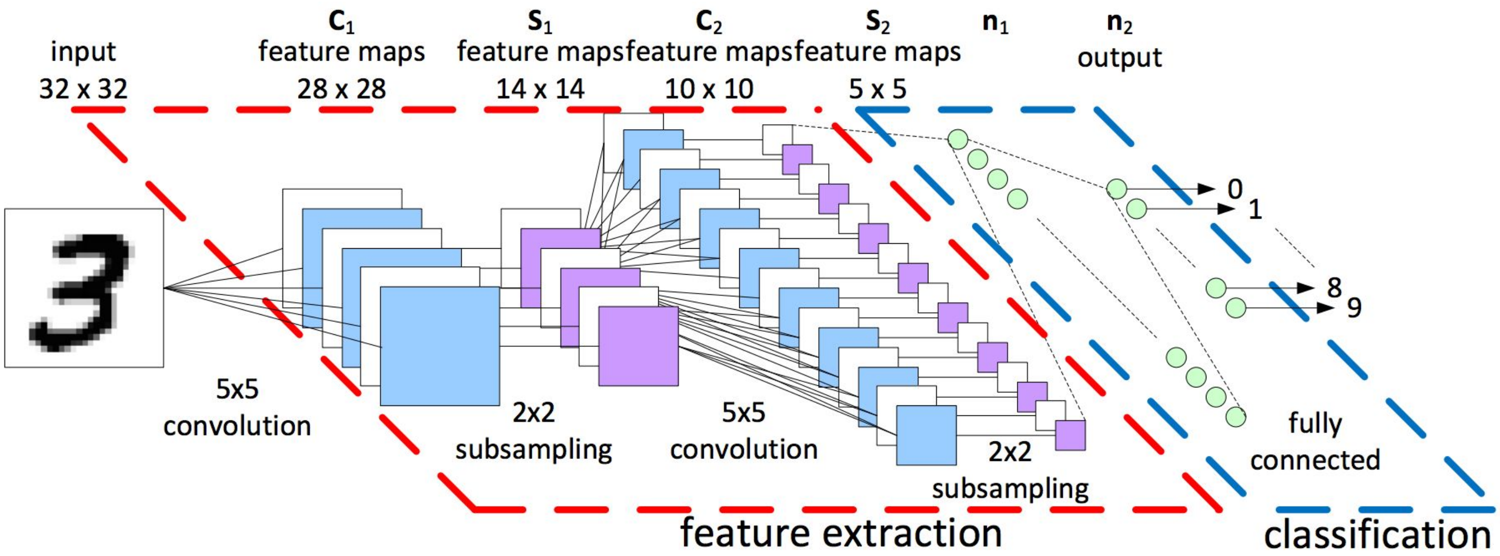
CNN Implementation - Data Augmentation (DA)

DA helps to popular artificial training instances from the existing train data sets.

(Image Credit: [Applied Deep Learning | Arden Dertat])

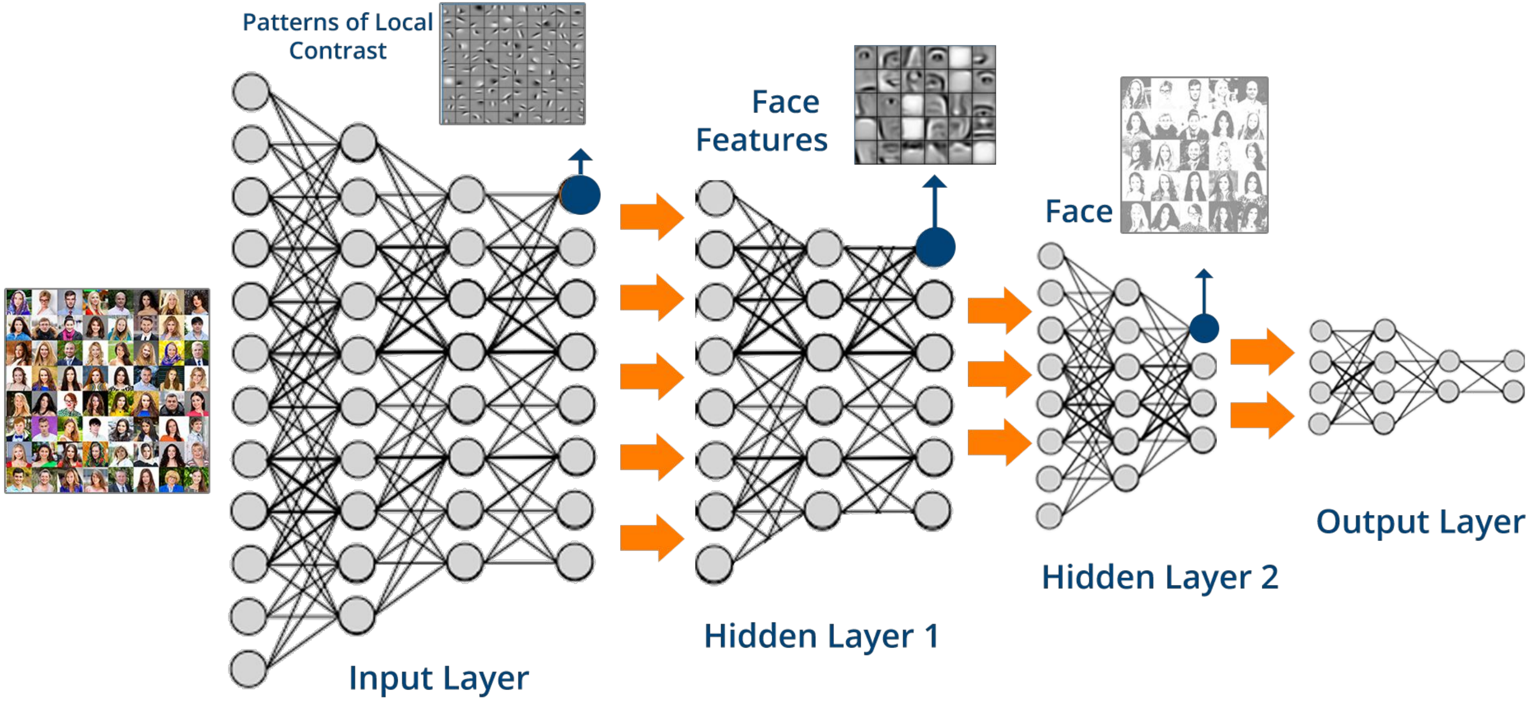
Convolutional Neural Networks

A convolutional neural network (**CNN**, or **ConvNet**) is a class of deep, feed-forward artificial neural networks that explicitly assumes that the inputs are images, which allows us to encode certain properties into the architecture.



LeNet-5 Architecture (Image Credit: https://becominghuman.ai)

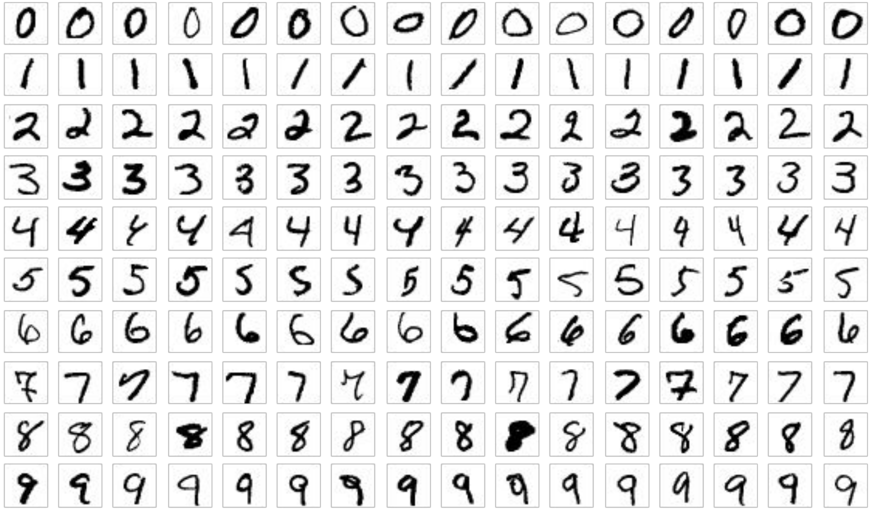
Deep Learning for Facial Recognition



(Image Credit: www.edureka.co)

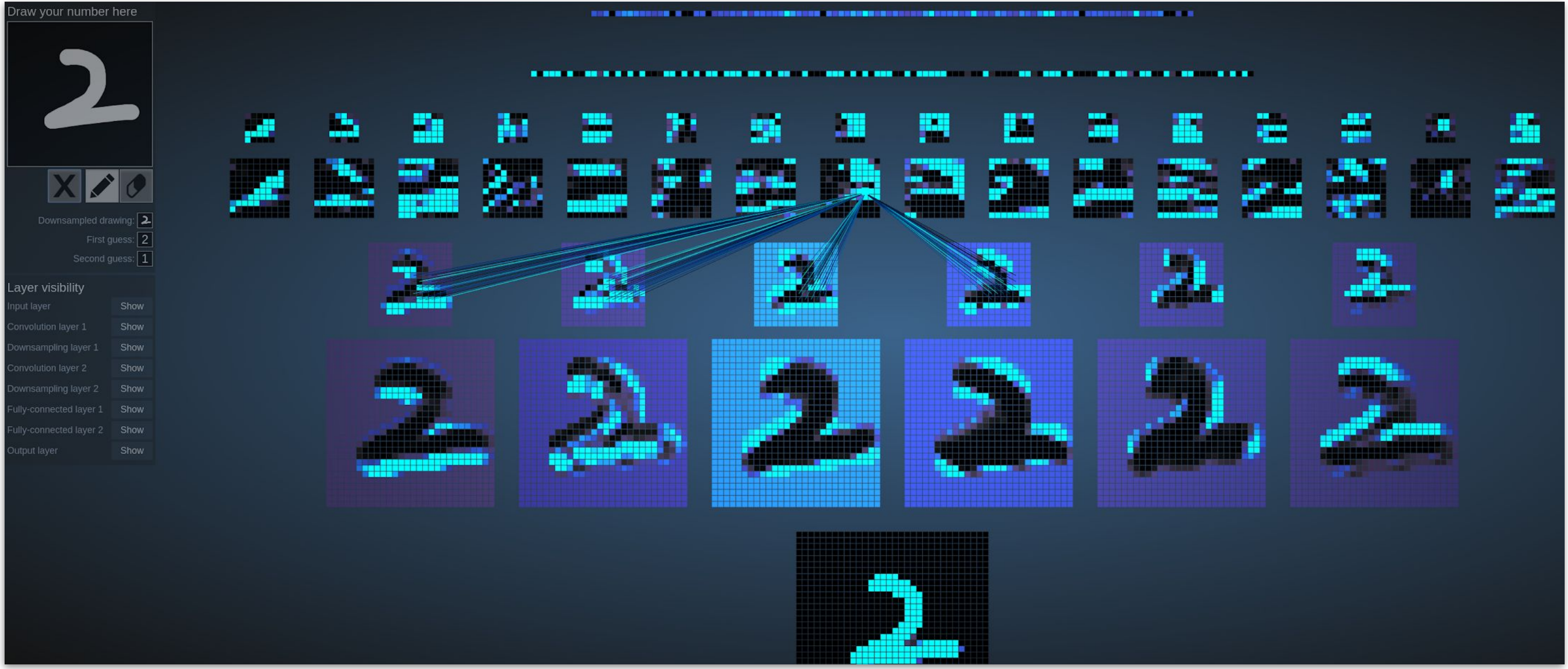
MNIST - Introduction

- **MNIST** (Mixed National Institute of Standards and Technology) is a database for handwritten digits, distributed by Yann Lecun.
- 60,000 examples, and a test set of 10,000 examples.
- 28x28 pixels each.
- Widely used for research and educational purposes.



(Image Credit: Wikipedia)

MNIST - CNN Visualization



(Image Credit: http://scs.ryerson.ca/~aharley/vis/)

Hands-on Session #1

A Simple Deep Learning Example with PyTorch - First Glance

Part III. Introduction to PyTorch

PyTorch website:
https://pytorch.org/

Deep Learning with PyTorch:
https://pytorch.org/tutorials/

A Brief History of PyTorch

PyTorch is an open source machine learning library based on the Torch library, which was first released by Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet in Oct 2002.

- The first official release of PyTorch was by Facebook's AI Research lab (FAIR) in Oct 2016.
- Version 1.0 that integrated both Caffe2 and ONNX was release in May 2018.
- The latest release is version 1.4.0, as of Feb 13 2020.

Overview of PyTorch

PyTorch is an open-source machine learning library written in Python, C++ and CUDA. PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a tape-based autodiff system

In a layman's term, PyTorch is a fancy version of NumPy that runs on GPUs and comes with a lot of machine learning functionalities.

TensorFlow, Keras, and PyTorch

TensorFlow

TensorFlow is an end-to-end open source **platform** for machine learning. It has a comprehensive, flexible ecosystem to build and deploy ML powered applications.

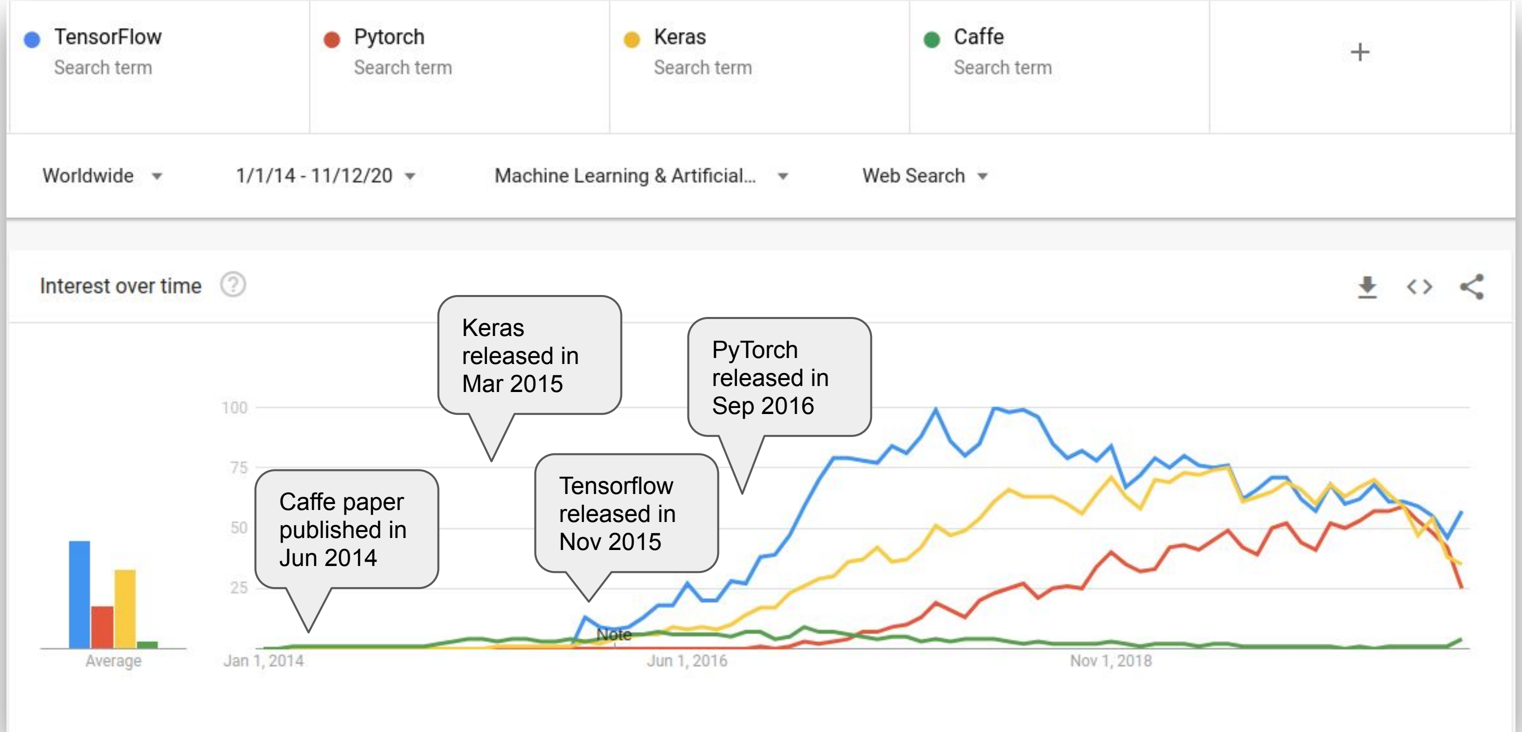
Keras

Keras is a high-level neural networks **API**, written in Python and capable of running on top of *TensorFlow*, *CNTK*, or *Theano*. It was developed with a focus on enabling fast experimentation.

PyTorch

PyTorch is an open source machine learning **framework** that accelerates the path from research prototyping to production deployment.

Google Trends for Popular ML Frameworks



(Image Credit: https://trends.google.com/)

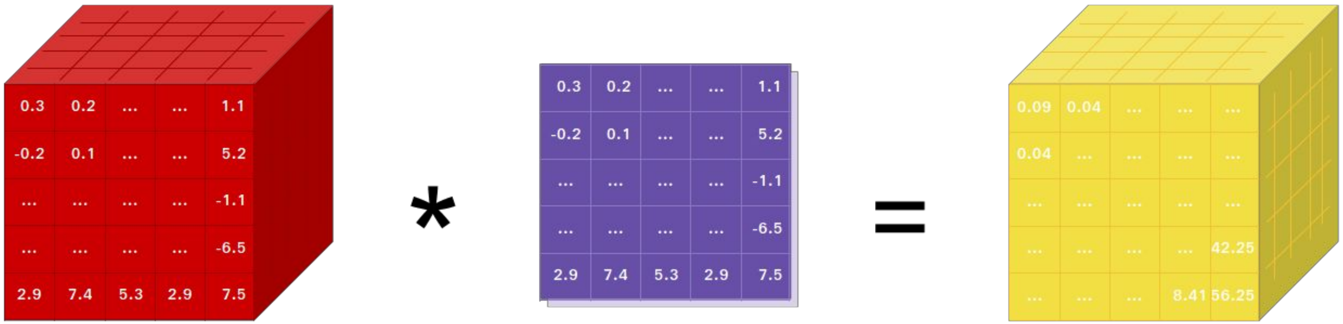
Major Components of PyTorch

Components	Description
torch	a ***Tensor library like NumPy***, with strong ***GPU*** support
torch.autograd	a ***tape-based automatic differentiation library*** that supports all differentiable Tensor operations in torch
torch.jit	a ***compilation stack (TorchScript)*** to create serializable and optimizable models from PyTorch code
torch.nn	a ***neural networks library*** deeply integrated with autograd designed for maximum flexibility
torch.multiprocessing	***Python multiprocessing***, but with magical memory sharing of torch Tensors across processes. Useful for data loading and Hogwild training
torch.utils	***DataLoader and other utility functions*** for convenience

A Powerful Tensor Library - torch

- A PyTorch tensor is an n-dimensional array that can live on either the CPU or GPU. A tensor has a static type, a rank, and a shape.

Name	Rank	Tensor
Scalar	0	[5]
Vector	1	[1 2 3]
Matrix	2	[[1 2 3 4], [5 6 7 8]]
Tensor	3	...



(Image Credit: pytorch.org)

Tensors on CPU and GPU - torch

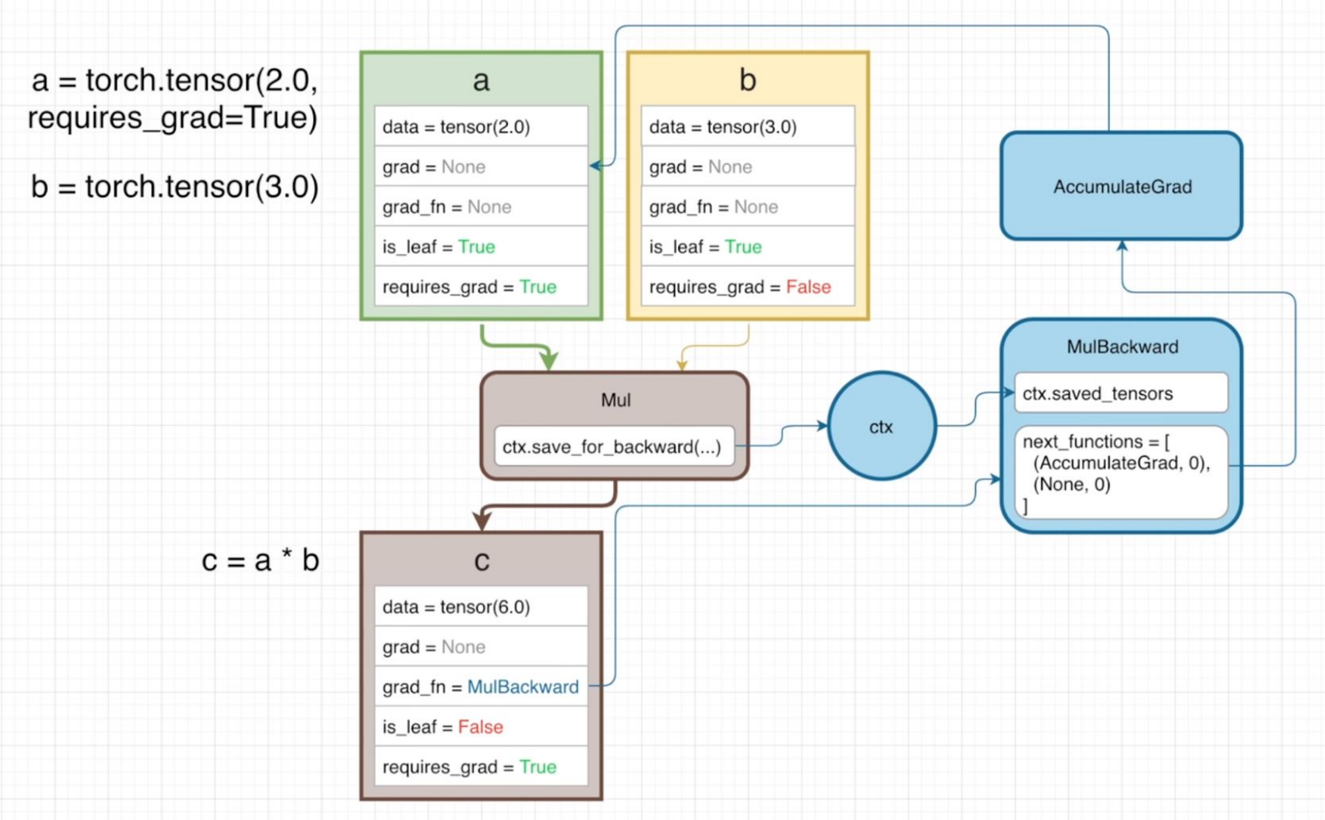
```python
x = touch.randn(1)
# check if a CUDA device is available
if torch.cuda.is_available():

    # a CUDA device object
    device = torch.device("cuda")

    # directly create y
    x = x.to(device)
    y = torch.ones_like(x, device=device)

    z = x + y
    print(z)
    print(z.to("cpu", torch.double))
```

Tape-Based AutoGrad - torch.autograd

- **torch.autograd** is central to all neural networks in PyTorch.
- The **autograd** package provides automatic differentiation for all operations on Tensors.
- Use "`requires_grad=True`" to keep traction operations on a Tensor.

```
# x = tensor([[1., 1.],
             [1., 1.]], requires_grad=True)
x = torch.ones(2, 2, requires_grad=True)

# y = tensor([[3., 3.],
             [3., 3.]], grad_fn=<AddBackward0>)
y = x + 2
```

Tape-Based AutoGrad - torch.autograd



(Image Credit: Elliot Waite: https://youtu.be/MswxJw-8PvE)

- PyTorch uses and replays a "***tape recorder***" to build neural networks.
- The official name of the method is called ***reverse-mode auto-differentiation***.
- The dependent variable is fixed and the derivative is computed with respect to each sub-expression recursively.
- The method requires extra storage to save intermediate states.

Dynamic Graph with PyTorch

A graph is created on the fly


```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

(Image Credit: pytorch.org)

Neural Network - torch.nn

- **torch.nn** depends on **autograd** to define models and differentiate them.
- An **nn.Module** contains layers, and a method **forward(input)** that returns the output.

```
import torch
import torch.nn as nn

# define a neural network model
class Net(nn.Module):

    def __init__(self, param):
        super(Net, self).__init__()
        self.param = param

    def forward(self, x):
        return x * self.param

net = Net(torch.Tensor([3, 4, 5]))
print(net)
```

Procedure to Train a Neural Network - Given a Data Set

Step 1

Definition

Define the neural network that has some learnable parameters (or weights)

Step 2

Iteration

Iterate over a dataset of inputs

Step 3

Forward Propagation

Process input through the network

Step 4

Loss Calculation

Compute the loss (how far is the output from being correct)

Step 5

Backward Propagation

Propagate gradients back into the network's parameters

Step 6

Updating

Update the weights of the network, typically using a simple update rule: weight = weight - learning_rate * gradient

Train a Neural Network - torch.nn

- **Define** the neural network that has some learnable parameters.
- **Iterate** over a dataset of inputs
- **Process** input through the network
- **Compute the loss** (how far is the output from being correct)
- **Propagate gradients back** into the network's parameters
- **Update the weights** of the network.

```
import torch.optim as optim

# Net is a predefined nn model
net = Net(torch.Tensor([3, 4, 5]))
output = net(input)

# define a dummy target
target = torch.randn(10)
target = target.view(1, -1)
criterion = nn.MSELoss()
loss = criterion(output, target)

# use one of the update rules such as SGD,
Nesterov-SGD, Adam, RMSProp, etc
optimizer = optim.SGD(net.parameters(),
lr=0.01)

# zero the gradient buffers
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Preparing Datasets for PyTorch

In order to train a decent deep neural network model with PyTorch, the input data sets needs to be **cleaned**, **balanced**, **transformed**, **scaled**, and **splitted**.

- Balance the classes. Unbalanced classes will interfere with training.
- Transform the categorical variables into one-hot encoded variables.
- Extract the X (variables) and y (targets) values for the training and testing datasets.
- Scale/normalize the variables.
- Shuffle and split the dataset into training and testing datasets

One-hot encoding

Dog	Cat	Horse
1	0	0
0	1	0
0	0	1

Numerical encoding

Dog	Cat	Horse
1	2	3

Predefined Datasets in torchvision

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision. The datasets include but not limited to MNIST, Fashion-MNIST, ImageNet, CIFAR, etc. They all have two common arguments:

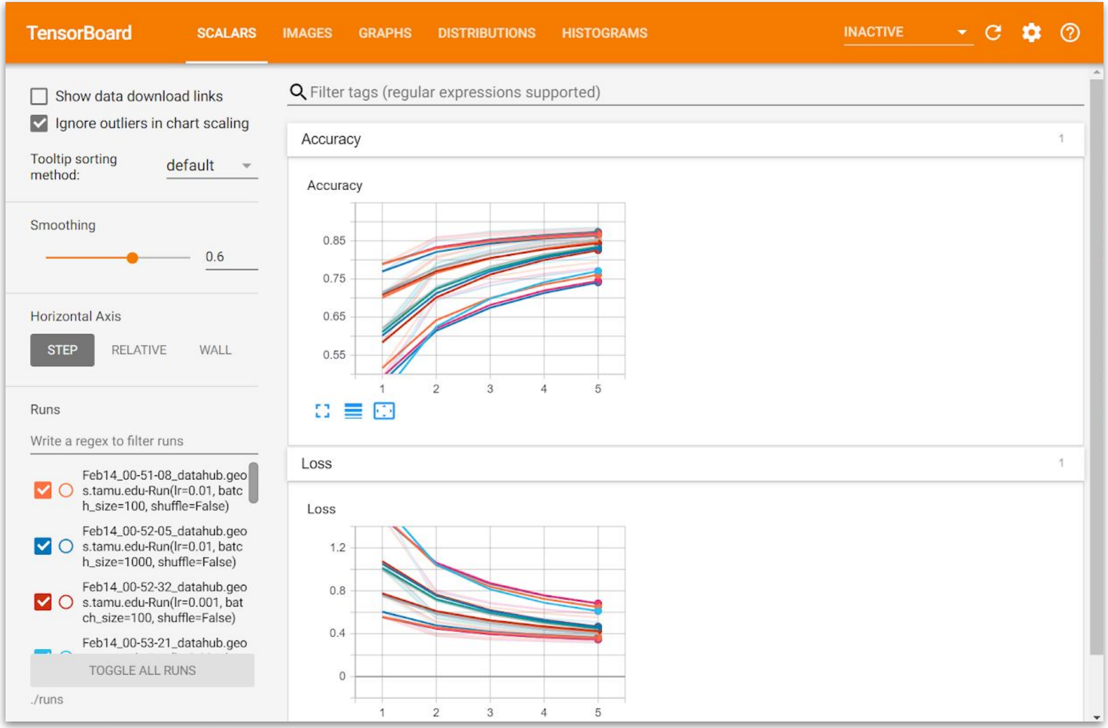
- **transform** to transform the input.
- **target_transform** to transform the target

The datasets can all be passed to a **torch.utils.data.DataLoader,** which can load multiple samples parallelly using **torch.multiprocessing** workers.

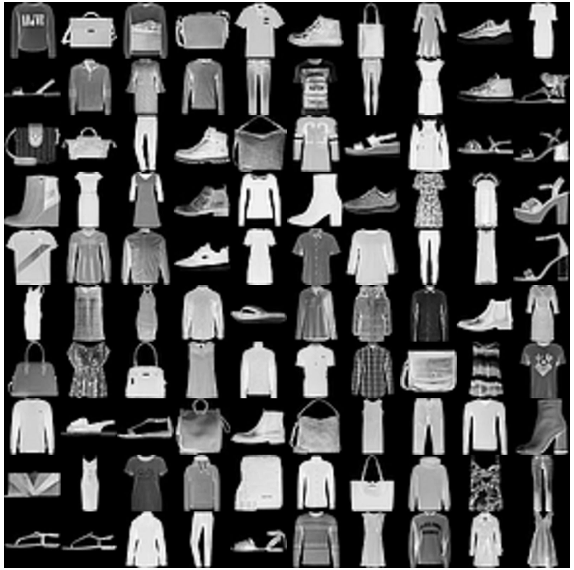
```
from torchvision import datasets

# import ImageNet data set
imagenet_data =
datasets.ImageNet('./imagenet')

data_loader =
torch.utils.data.DataLoader(
              imagenet_data,
              batch_size=4,
              shuffle=True,
num_workers=args.nThreads)
```

Monitoring Training with Tensorboard

- TensorBoard is a User Interface (UI) tools designed for TensorFlow.
- More details on TensorBoard can be found at TensorBoard.
- Once you've installed TensorBoard, these utilities let you log PyTorch models and metrics into a directory for visualization within the TensorBoard UI.



Hands-on Session #3
Classify Fashion-MNIST with PyTorch



- Fashion-MNIST is a dataset of Zalando's article images
- consisting of a training set of 60,000 examples and a test set of 10,000 examples.
- Each example is a 28x28 grayscale image, associated with a label from 10 classes.