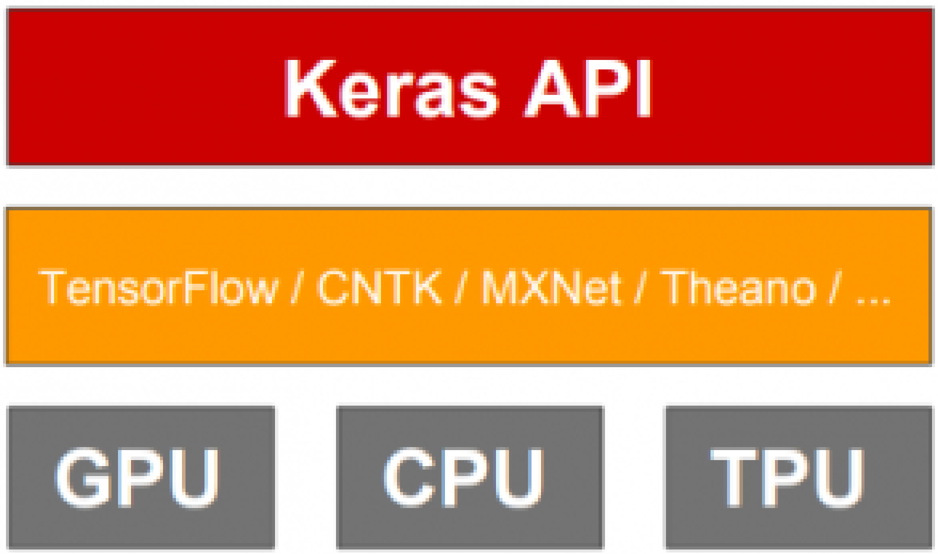
Memo - Deep Learning - Tensorflow

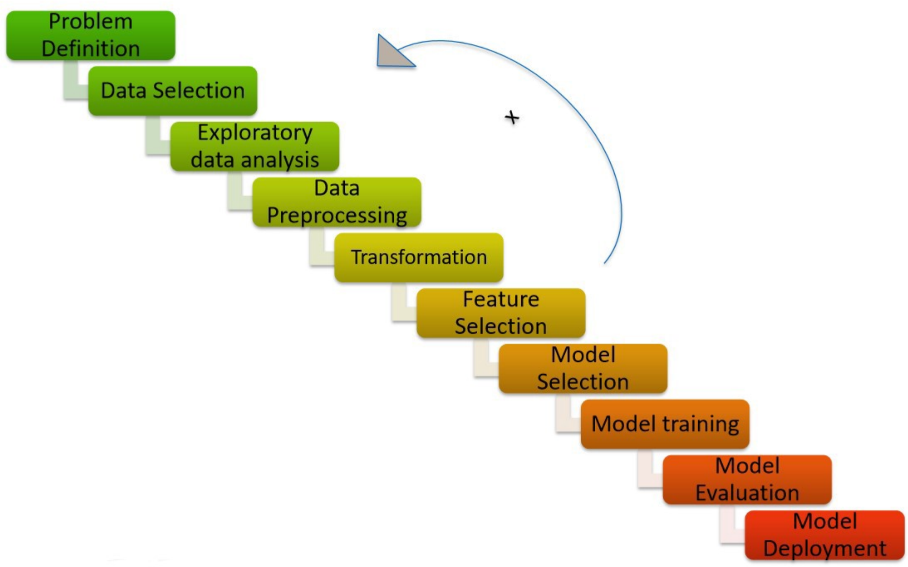
In [1]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

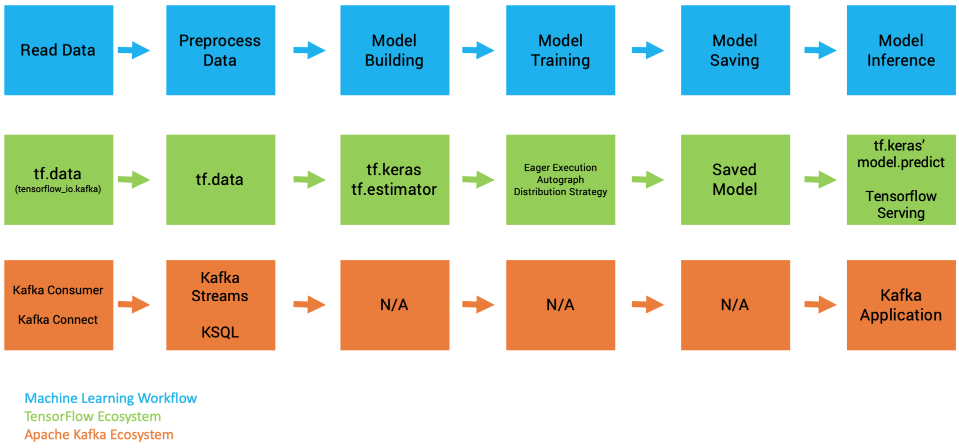
```
Mounted at /content/drive
```

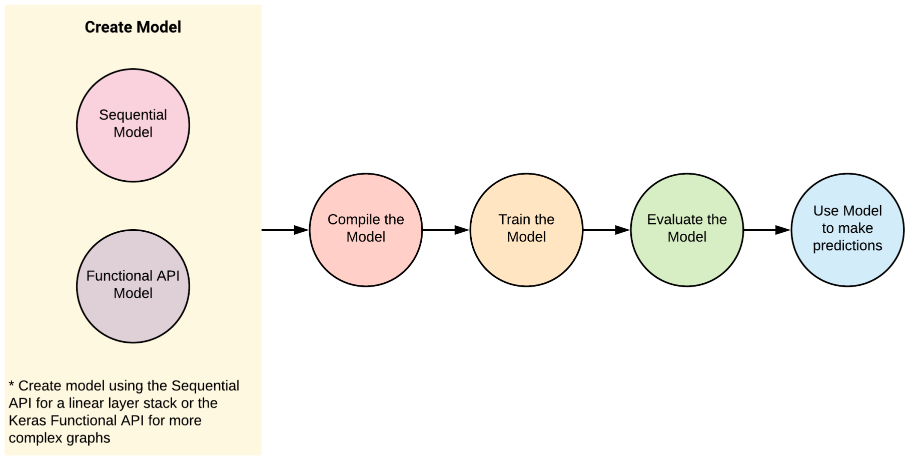
Tensorflow

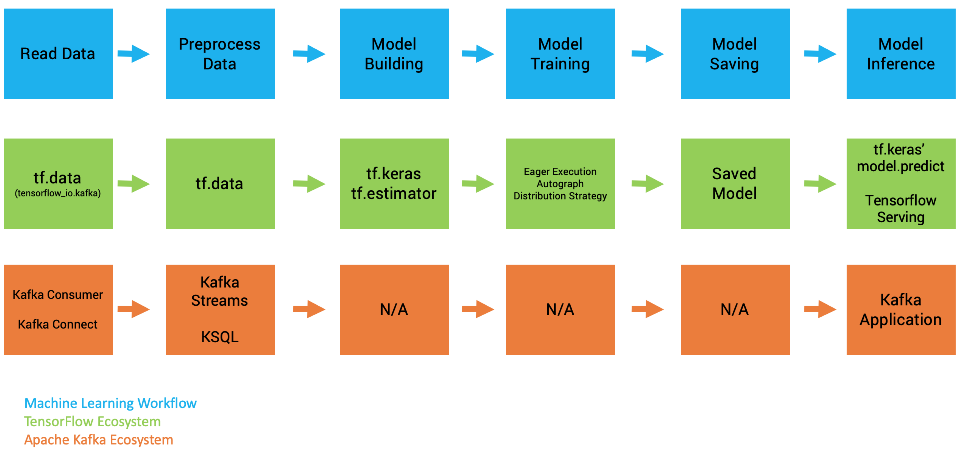


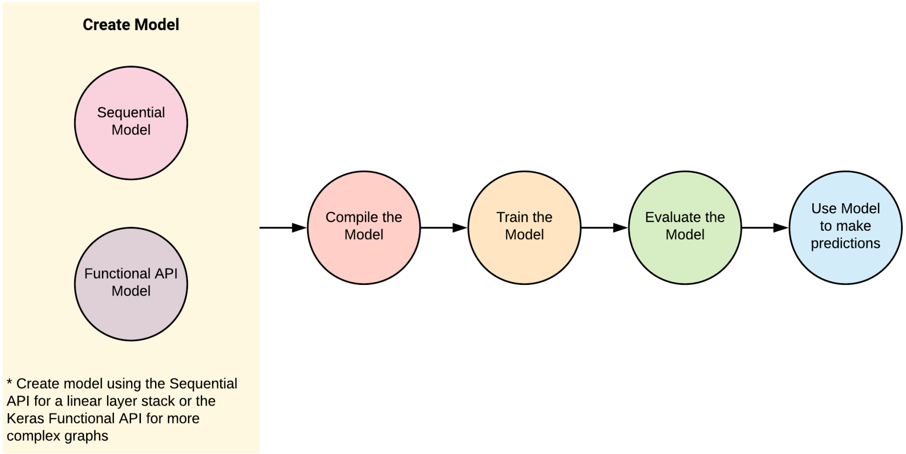
Machine Learngin / Deep Learning - Workflow

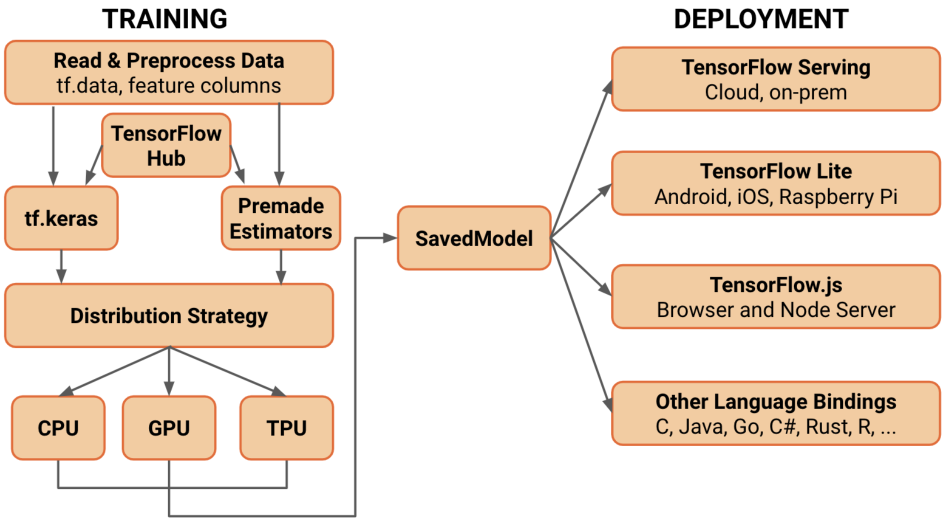



Read Data
Preprocess Data
Model Building
Model Training
Model Saving
Model Inference
tf.data (tensorflow_io.kafka)
tf.data
tf.keras tf.estimator
Eager Execution Autograph Distribution Strategy
Saved Model
tf.keras' model.predict
Tensorflow Serving
Kafka Consumer
Kafka Connect
Kafka Streams
KSQL
N/A
N/A
N/A
Kafka Application
Machine Learning Workflow
TensorFlow Ecosystem
Apache Kafka Ecosystem


Create Model
Sequential Model
Functional API Model
* Create model using the Sequential API for a linear layer stack or the Keras Functional API for more complex graphs
Compile the Model
Train the Model
Evaluate the Model
Use Model to make predictions


Read Data
Preprocess Data
Model Building
Model Training
Model Saving
Model Inference
tf.data (tensorflow_io.kafka)
tf.data
tf.keras tf.estimator
Eager Execution Autograph Distribution Strategy
Saved Model
tf.keras' model.predict
Tensorflow Serving
Kafka Consumer
Kafka Connect
Kafka Streams
KSQL
N/A
N/A
N/A
Kafka Application
Machine Learning Workflow
TensorFlow Ecosystem
Apache Kafka Ecosystem

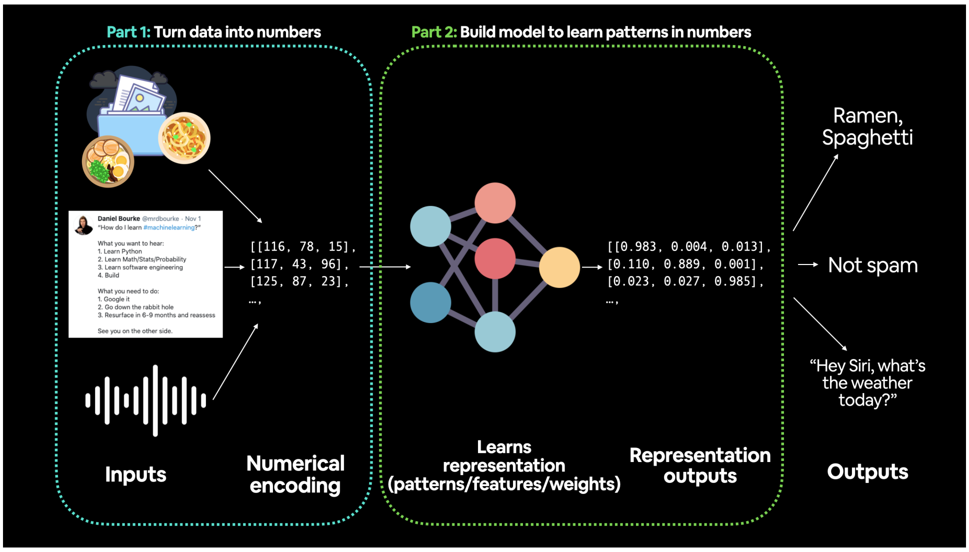




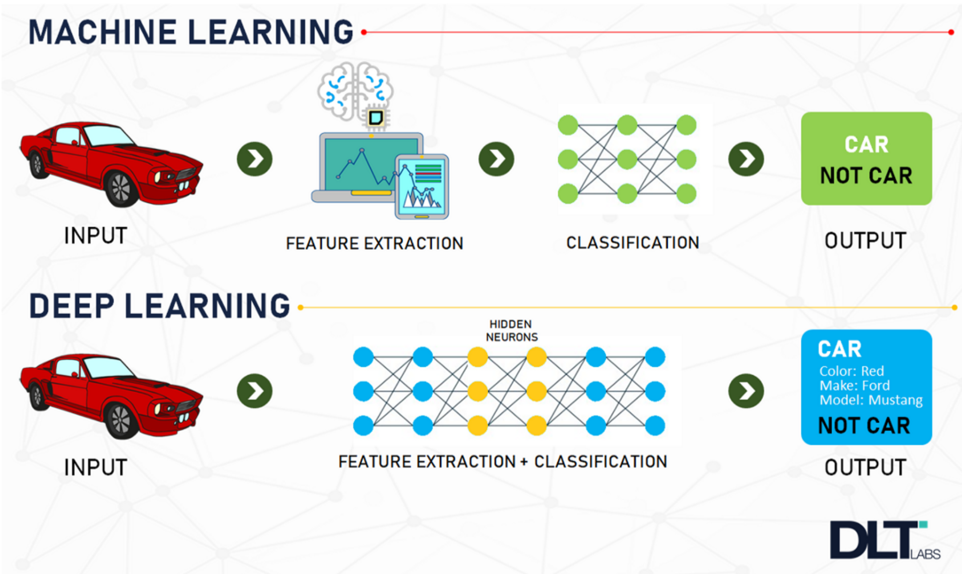
Machine Learning

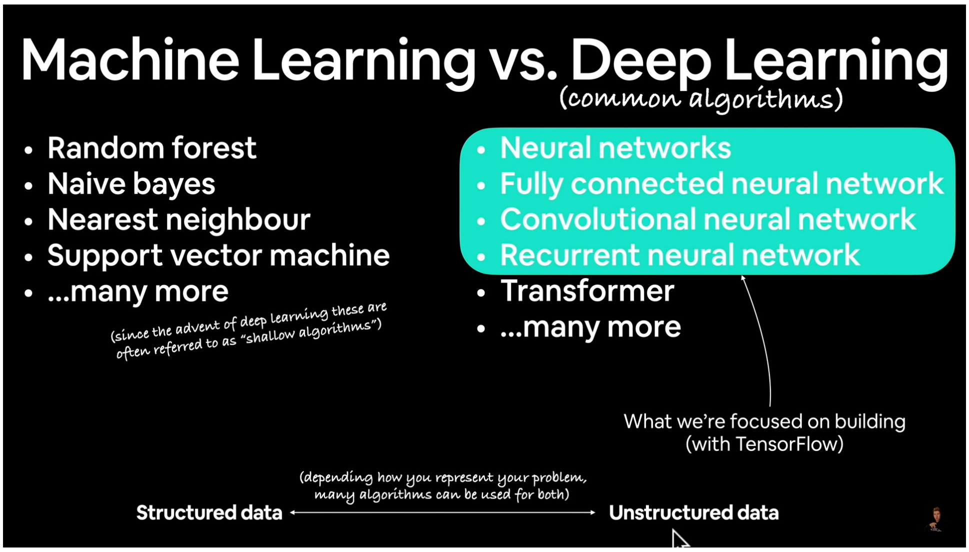
> Machine learning is a game of two parts:
>
> 1. Turn your data, whatever it is, into numbers (a representation).
> 2. Pick or build a model to learn the representation as best as possible.

> machine learning is a game of two parts: 1. turn your data into a representative set of numbers and 2. build or pick a model to learn the representation as best as possible

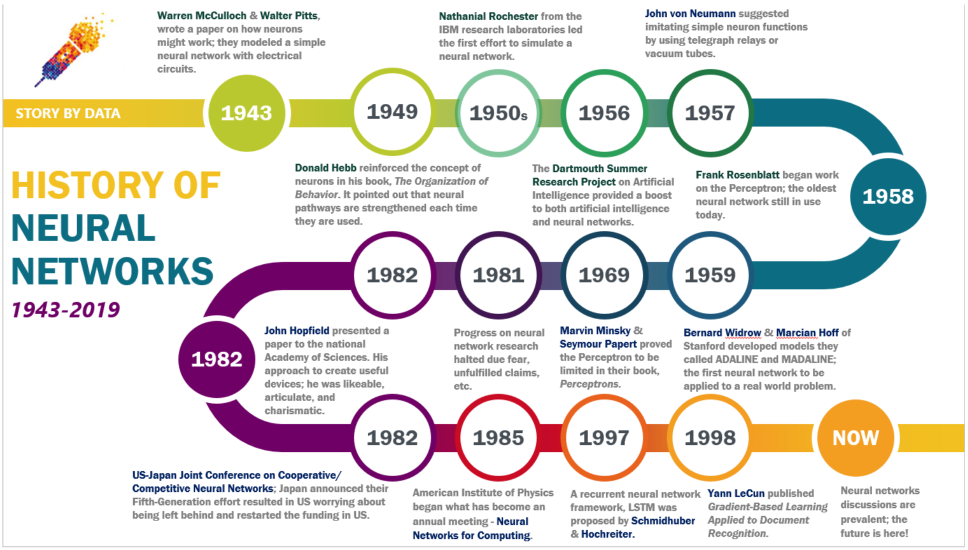


Maching Learning vs Deep Learning

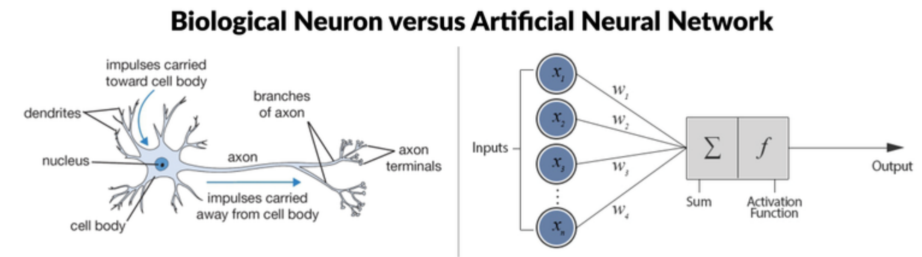




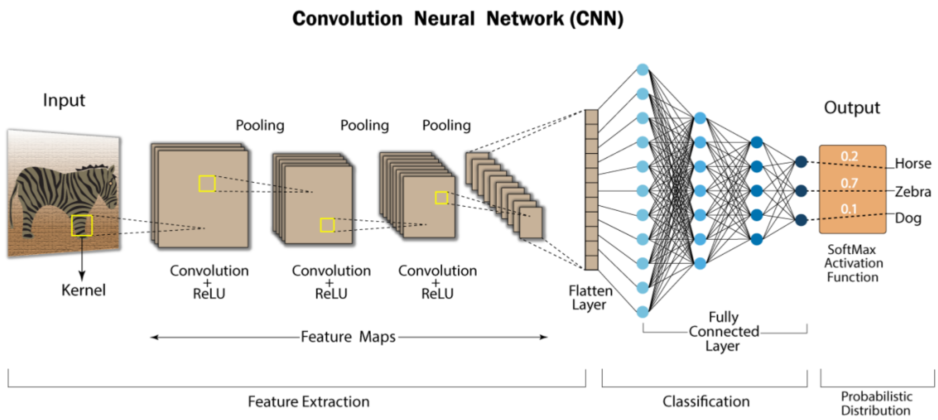
Deep Learning Neural Network

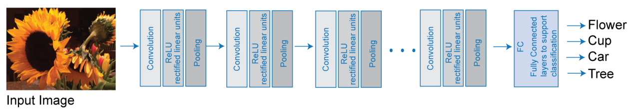

STORY BY DATA
HISTORY OF NEURAL NETWORKS
1943-2019
1943
Warren McCulloch & Walter Pitts, wrote a paper on how neurons might work; they modeled a simple neural network with electrical circuits.
1949
Donald Hebb reinforced the concept of neurons in his book, The Organization of Behavior. It pointed out that neural pathways are strengthened each time they are used.
1950s
Nathanial Rochester from the IBM research laboratories led the first effort to simulate a neural network.
1956
The Dartmouth Summer Research Project on Artificial Intelligence provided a boost to both artificial intelligence and neural networks.
1957
John von Neumann suggested imitating simple neuron functions by using telegraph relays or vacuum tubes.
1958
Frank Rosenblatt began work on the Perceptron; the oldest neural network still in use today.
1959
Bernard Widrow & Marcian Hoff of Stanford developed models they called ADALINE and MADALINE; the first neural network to be applied to a real world problem.
1969
Marvin Minsky & Seymour Papert proved the Perceptron to be limited in their book, Perceptrons.
1981
Progress on neural network research halted due fear, unfulfilled claims, etc.
1982
John Hopfield presented a paper to the national Academy of Sciences. His approach to create useful devices; he was likeable, articulate, and charismatic.
1982
US-Japan Joint Conference on Cooperative/Competitive Neural Networks; Japan announced their Fifth-Generation effort resulted in US worrying about being left behind and restarted the funding in US.
1985
American Institute of Physics began what has become an annual meeting - Neural Networks for Computing.
1997
A recurrent neural network framework, LSTM was proposed by Schmidhuber & Hochreiter.
1998
Yann LeCun published Gradient-Based Learning Applied to Document Recognition.
NOW
Neural networks discussions are prevalent; the future is here!

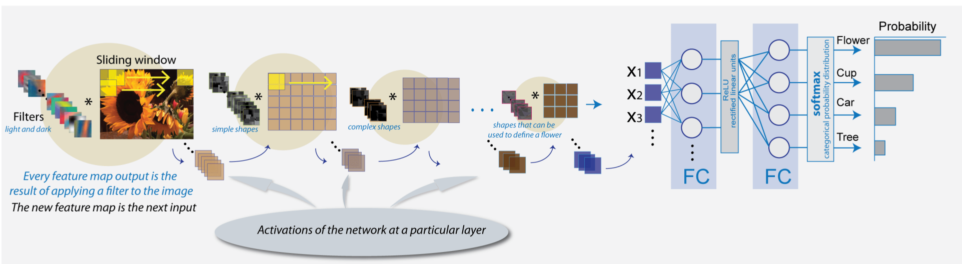
ANN


Biological Neuron versus Artificial Neural Network
impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body
Inputs
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_n
Σ
f
Sum
Activation Function
Output

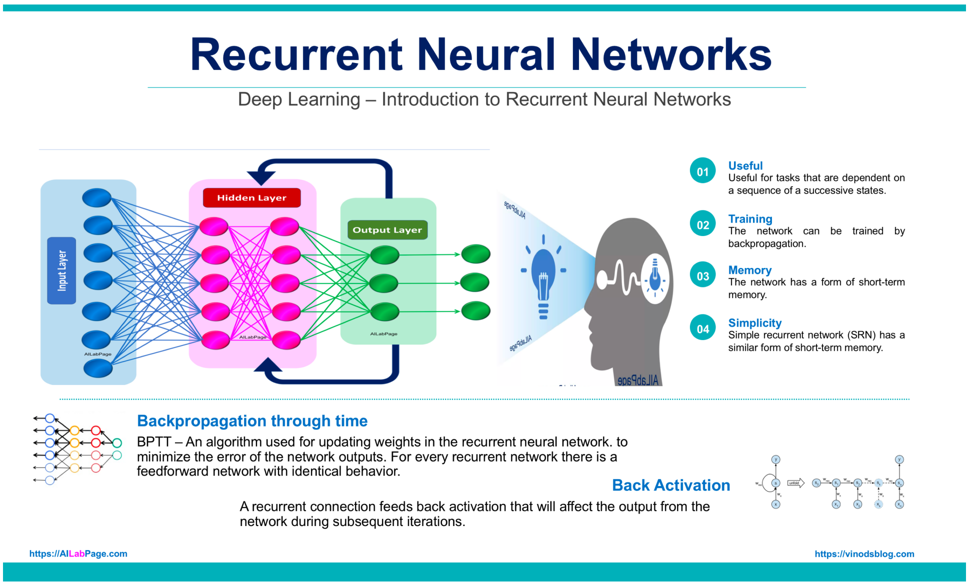
CNN


Convolution Neural Network (CNN)
Input
Pooling
Pooling
Pooling
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
Fully Connected Layer
Feature Maps
Feature Extraction
Classification
Probabilistic Distribution

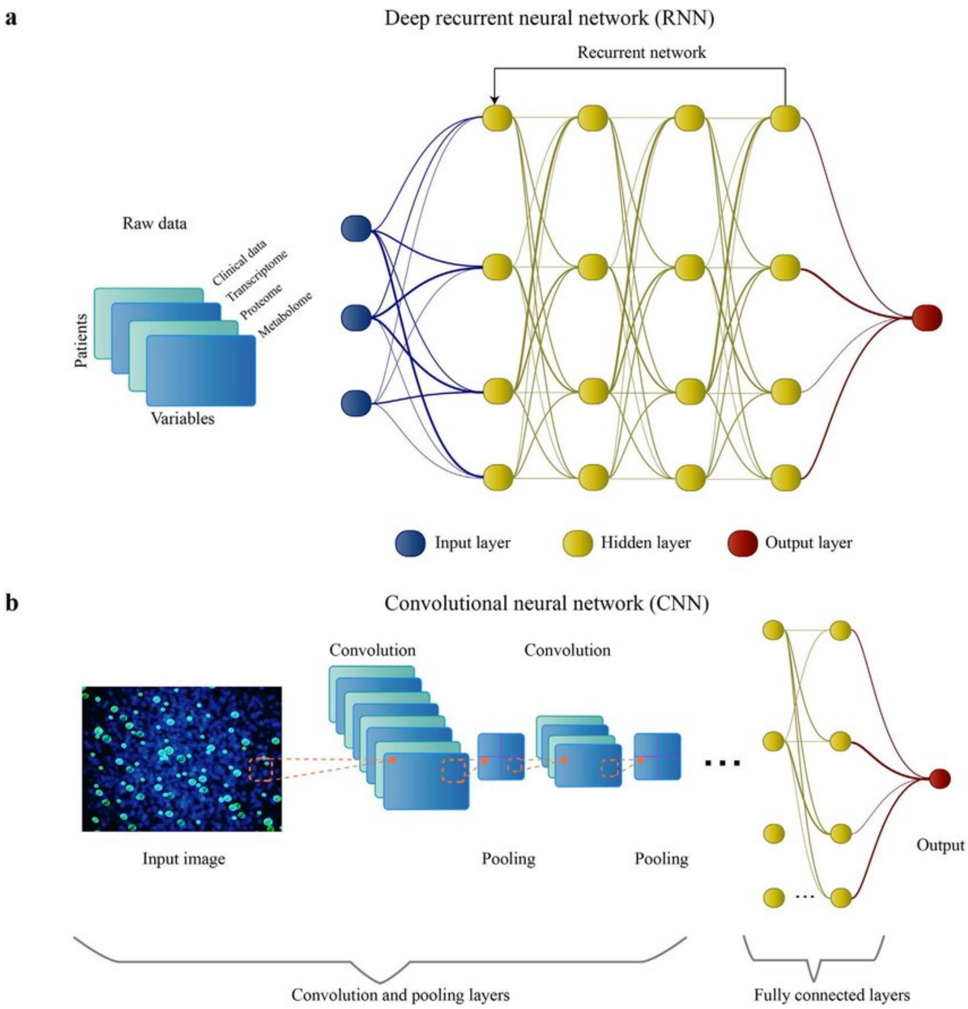

Input Image
Convolution
ReLU rectified linear units
Pooling
FC
Fully Connected layers to support classification
Flower
Cup
Car
Tree


Sliding window
Filters
light and dark
simple shapes
complex shapes
shapes that can be used to define a flower
Every feature map output is the result of applying a filter to the image
The new feature map is the next input
Activations of the network at a particular layer
X1
X2
X3
ReLU rectified linear units
FC
FC
softmax
categorical probability distribution
Probability
Flower
Cup
Car
Tree

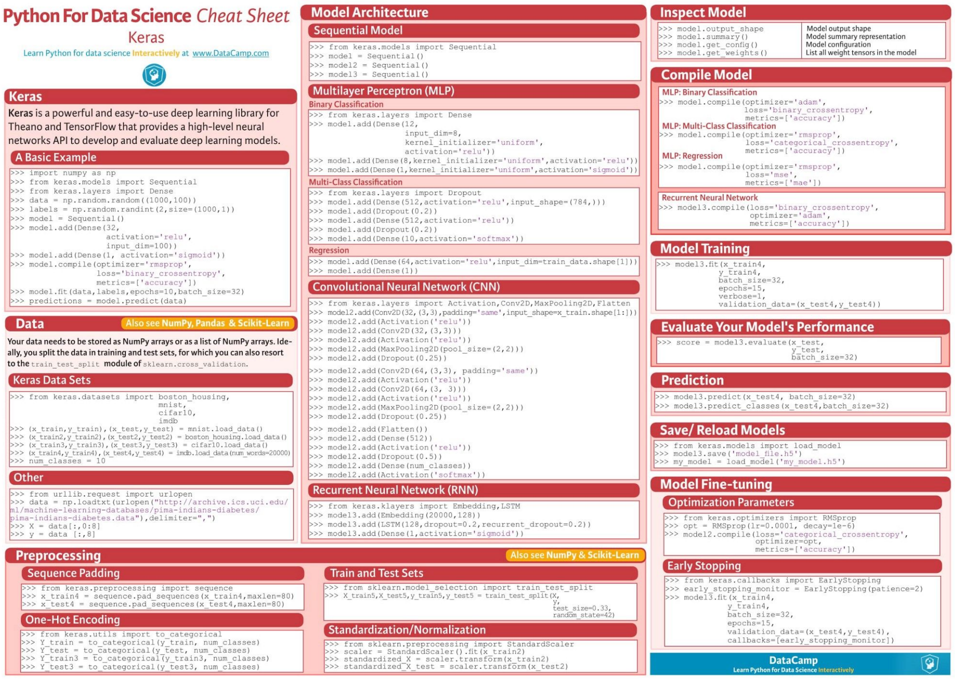
RNN



RNN vs CNN



Keras



Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

DataCamp

Learn Python for Data Science Interactively

Project 1 - Aritifical Neural Network for Handwritten Digits Classification

Preparation

In [3]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
import tensorflow as tf
from tensorflow import keras
```

Read Data

In [4]:
```python
keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
Out[4]: ((array([[[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 ...,

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8),
          array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)),
         (array([[[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]],

                 [[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
```

```
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8),
 array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)))
```

In [5]:
```
type(keras.datasets.mnist.load_data())
```

Out[5]:
```
tuple
```

In [6]:
```
len(keras.datasets.mnist.load_data())
```

Out[6]:
```
2
```

Preprocess the Data -> Design Thinking

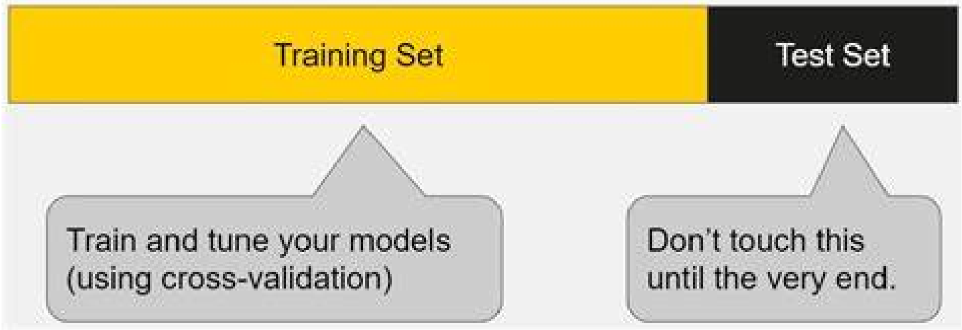
Observing Data

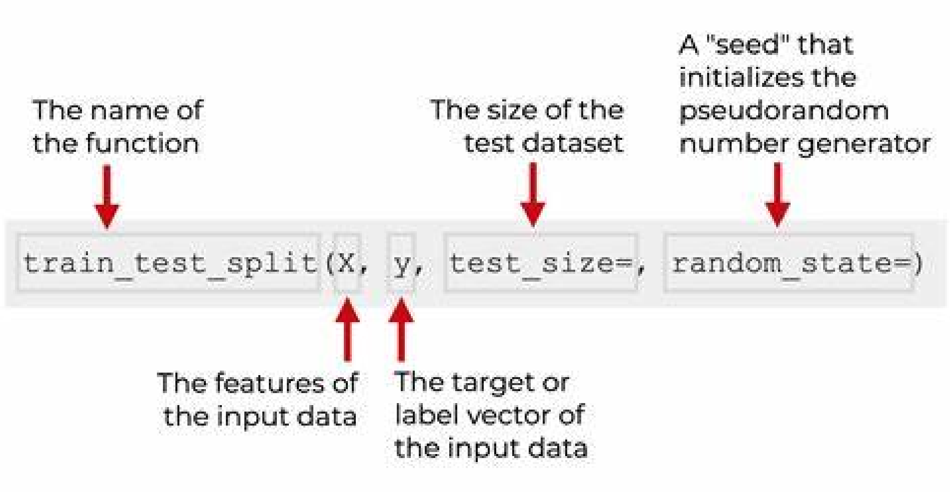
Train-Test Split

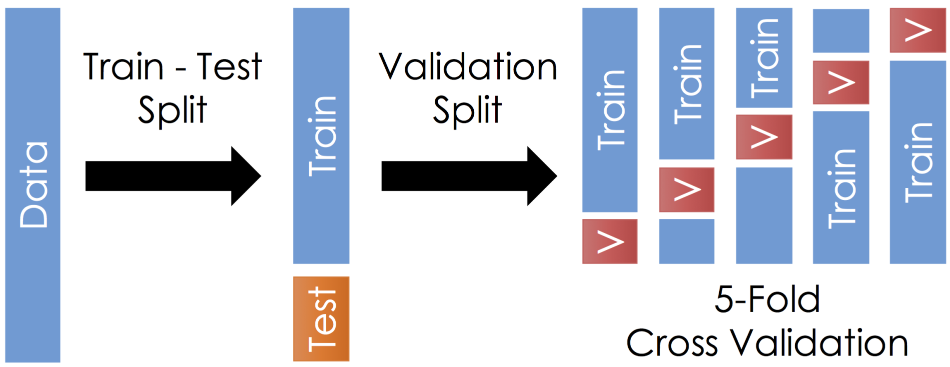
Each split of the dataset serves a specific purpose:

| Split | Purpose | Amount of total data | How often is it used? | | ----- | ----- | -
---- | ----- | | **Training set** | The model learns from this data (like the course

materials you study during the semester). | ~60-80% | Always | | **Validation set** | The model gets tuned on this data (like the practice exam you take before the final exam). | ~10-20% | Often but not always | | **Testing set** | The model gets evaluated on this data to test what it has learned (like the final exam you take at the end of the semester). | ~10-20% | Always |







In [7]:
```
(X_train, y_train), (X_test, y_test)=keras.datasets.mnist.load_data()
```

In [8]:
```
type(X_train), type(y_train)
```

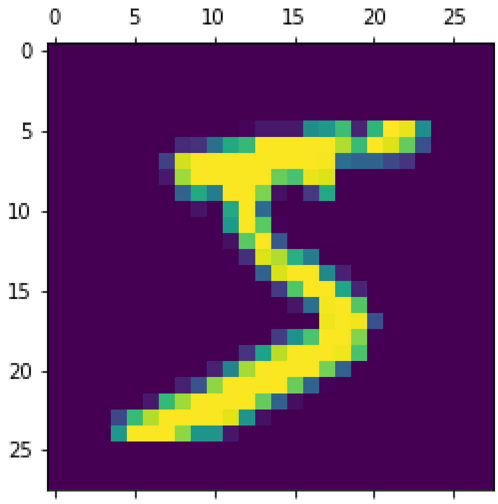
Out[8]: (numpy.ndarray, numpy.ndarray)

In [9]:
```
X_train.shape, y_train.shape
```

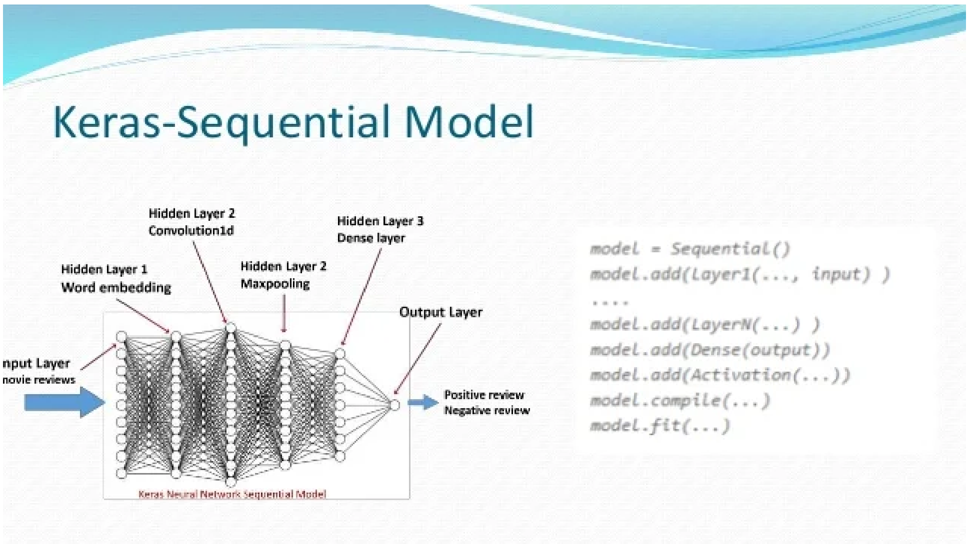
Out[9]: ((60000, 28, 28), (60000,))

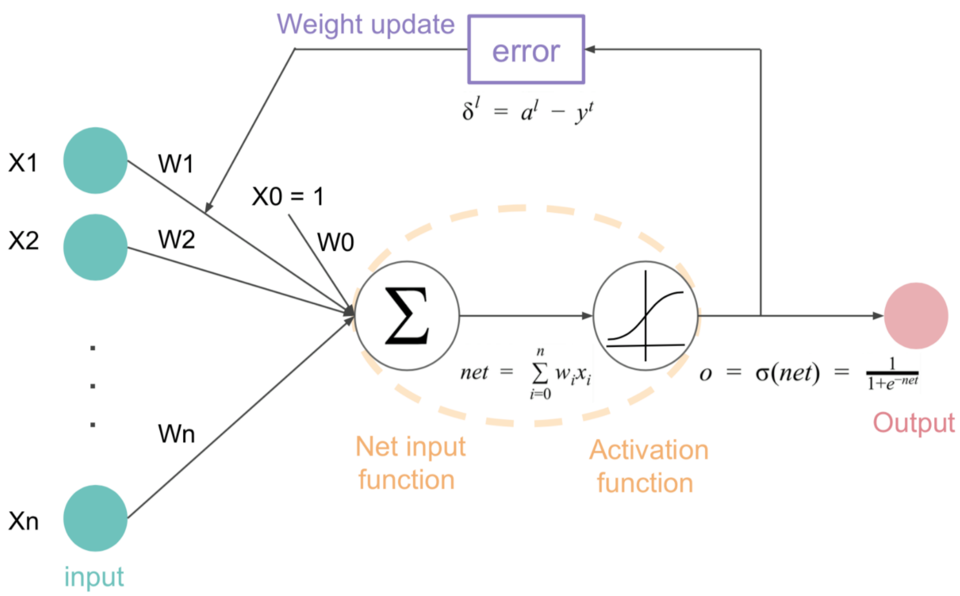
In [10]:
```
plt.matshow(X_train[0])
```

Out[10]: <matplotlib.image.AxesImage at 0x7fc7bf953d10>

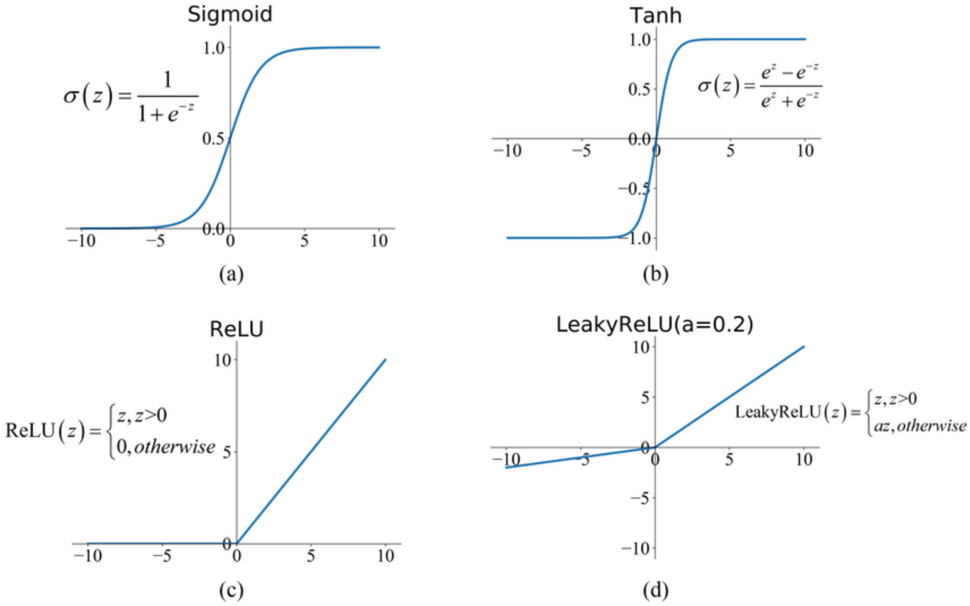


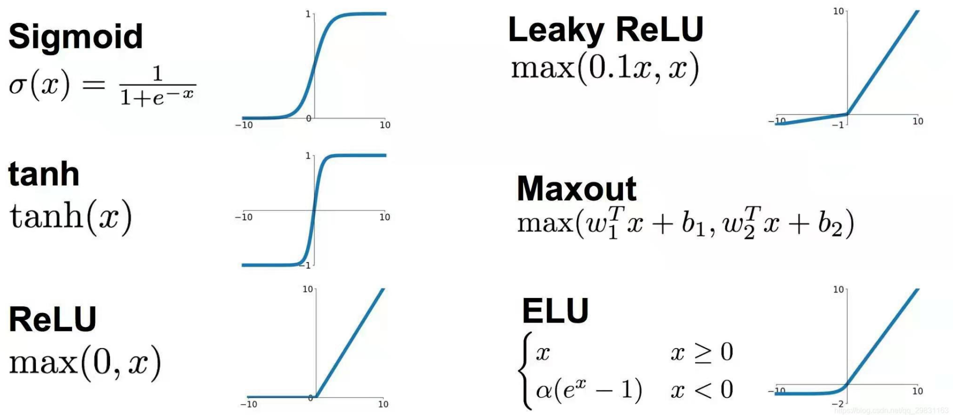
Model Building





Activation





TensorFlow Keras **손실 함수 (Loss Function)**

문제 유형 (Problem types)	마지막층 활성화 함수 (Last-layer activation)	손실 함수(클래스 (Loss function(class)	비고
Binary classfication	sigmoid	binary_crossentropy (tf.keras.losses.BinaryCrossentropy)	
Multiclass, single-label classification	softmax	categorical_crossentropy (tf.keras.losses.CategoricalCrossentropy)	y: one-hot encoded
		sparse_categorical_crossentropy (tf.keras.losses.SparseCategoricalCrossentropy)	y: integer
Multiclass, multilabel classification	softmax	categorical_crossentropy (tf.keras.losses.CategoricalCrossentropy) tf.nn.softmax_cross_entropy_with_logits	y: one-hot encoded
	sigmoid	binary_crossentropy (tf.keras.losses.BinaryCrossentropy) tf.nn.sigmoid_cross_entropy_with_logits	
Regression to arbitrary values	None	mse, mean_squared_error (tf.keras.losses.MeanSquaredError)	
Regression to values between 0 and 1	sigmoid	mse, mean_squared_error (tf.keras.losses.MeanSquaredError) binary_crossentropy (tf.keras.losses.BinaryCrossentropy)	

[R, Python 분석과 프로그래밍의 친구] https://rfriend.tistory.com

Model Traning

- Flattern Data

In [11]:
```
X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_train_flattened.shape
```

Out[11]: (60000, 784)

In [12]:
```
X_test_flattened = X_test.reshape(len(X_test), 28*28)
X_test_flattened.shape
```

Out[12]:
```
(10000, 784)
```

In [13]:
```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 9.7375 - accurac
y: 0.8400
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 6.0182 - accurac
y: 0.8787
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 5.6389 - accurac
y: 0.8832
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 5.5092 - accurac
y: 0.8852
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 5.3579 - accurac
y: 0.8866
```

Out[13]:
```
<keras.callbacks.History at 0x7fc7bb25e550>
```

> ### - Scaling Data

In [14]:
```
X_train_flattened[0]
```

```
Out[14]: array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  18,  18,  18,
                126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
                253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
                253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
                253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,  14,   1, 154, 253,  90,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0, 139, 253, 190,   2,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,  70,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
                241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,  45, 186, 253, 253, 150,  27,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,  16,  93, 252, 253, 187,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 249,
                253, 249,  64,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  46, 130,
                183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
                229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114,
                221, 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  23,  66,
                213, 253, 253, 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 171,
                219, 253, 253, 253, 253, 195,  80,   9,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  55, 172,
                226, 253, 253, 253, 253, 244, 133,  11,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                136, 253, 253, 253, 212, 135, 132,  16,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                  0,   0,   0,   0], dtype=uint8)
```

In [15]:
```
X_train_flattened = X_train_flattened / 255
X_test_flattened = X_test_flattened / 255
```

In [16]:
```
# After Scaling
model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 1.2903 - accurac
y: 0.8065
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8961 - accurac
y: 0.8676
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6193 - accurac
y: 0.8830
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4530 - accurac
y: 0.8957
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3610 - accurac
y: 0.9062
```

Out[16]:
```
<keras.callbacks.History at 0x7fc7b7a20a50>
```

Model Evaluate

In [17]:
```
model.evaluate(X_test_flattened, y_test)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.3215 - accuracy:
0.9143
```

Out[17]:
```
[0.32147109508514404, 0.9143000245094299]
```

- Prdediction

In [18]:
```
y_preds = model.predict(X_test_flattened)
```

```
313/313 [==============================] - 0s 1ms/step
```

In [19]:
```
y_preds[0]
```

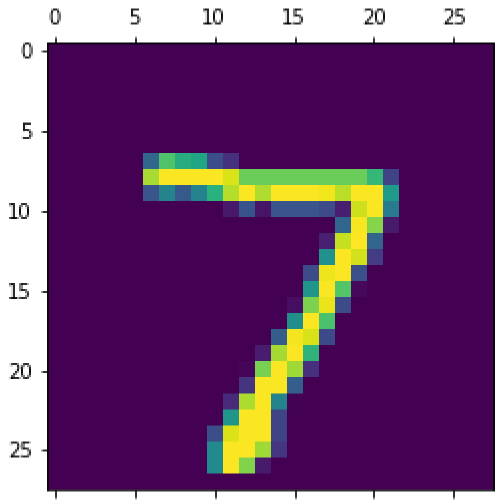
Out[19]:
```
array([1.0158585e-02, 1.7649877e-04, 3.3006098e-02, 7.3862779e-01,
       5.6233473e-02, 1.8736912e-01, 1.5852131e-05, 9.9895340e-01,
       1.3272220e-01, 7.7970177e-01], dtype=float32)
```

In [20]:
```
np.argmax(y_preds[0])
```

Out[20]:
```
7
```

In [21]:
```
plt.matshow(X_test[0])
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x7fc7bb09ab90>
```



In [22]:
```
y_test[0]
```

Out[22]:
```
7
```

In [23]:
```
y_pred_labels = [np.argmax(y_pred) for y_pred in y_preds]
```

In [24]:
```
y_pred_labels[:5]
```

Out[24]:
```
[7, 2, 1, 0, 4]
```

In [25]:
```
y_test[:5]
```

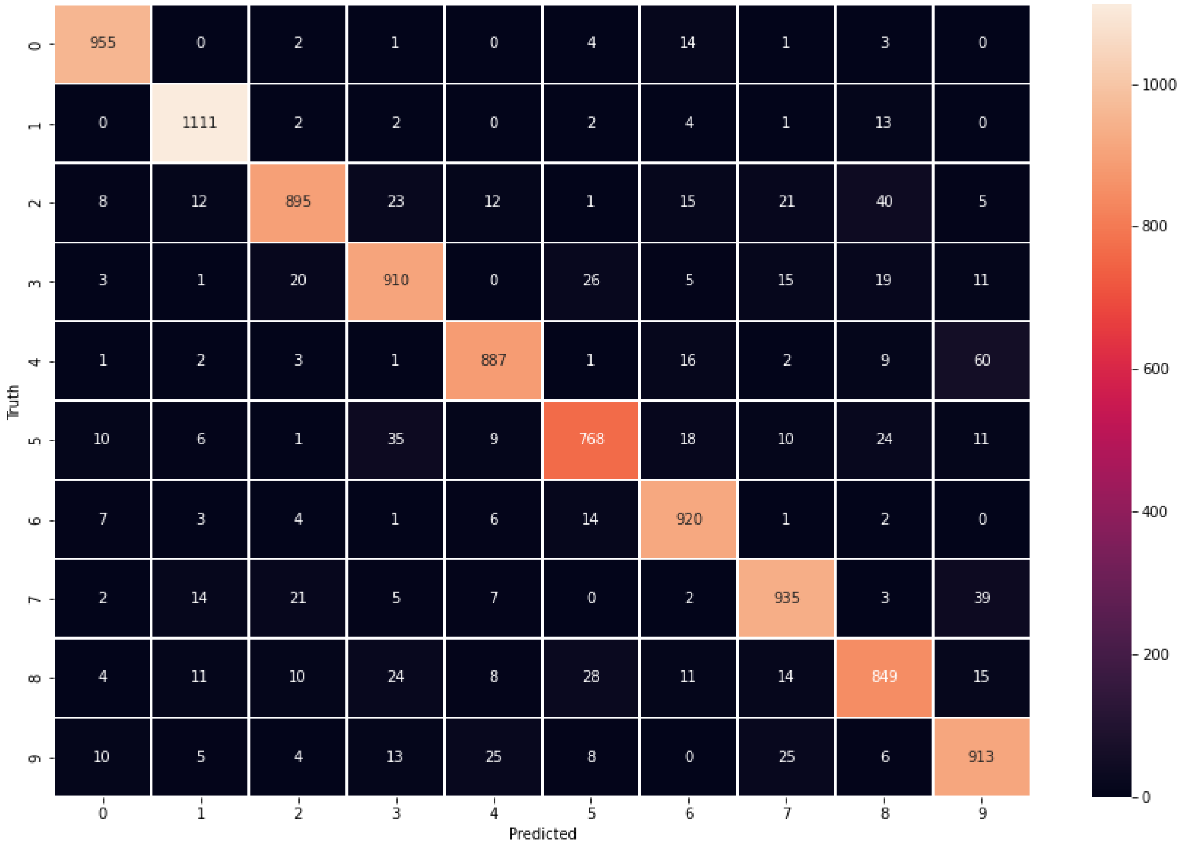
Out[25]:
```
array([7, 2, 1, 0, 4], dtype=uint8)
```

In [26]:
```
cm = tf.math.confusion_matrix(
    labels=y_test,
    predictions=y_pred_labels
)
cm
```

Out[26]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 955,    0,    2,    1,    0,    4,   14,    1,    3,    0],
       [   0, 1111,    2,    2,    0,    2,    4,    1,   13,    0],
       [   8,   12,  895,   23,   12,    1,   15,   21,   40,    5],
       [   3,    1,   20,  910,    0,   26,    5,   15,   19,   11],
       [   1,    2,    3,    1,  887,    1,   16,    2,    9,   60],
       [  10,    6,    1,   35,    9,  768,   18,   10,   24,   11],
       [   7,    3,    4,    1,    6,   14,  920,    1,    2,    0],
       [   2,   14,   21,    5,    7,    0,    2,  935,    3,   39],
       [   4,   11,   10,   24,    8,   28,   11,   14,  849,   15],
       [  10,    5,    4,   13,   25,    8,    0,   25,    6,  913]],
      dtype=int32)>
```

In [27]:
```
plt.figure(figsize=(15, 10))
sn.heatmap(
    cm,
    annot=True,
    fmt="d",
    linewidth=0.5,
)
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

Out[27]: Text(114.0, 0.5, 'Truth')



- Add Hidden Layer to Improve

In [28]:
```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid'),
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2746 - accurac
y: 0.9217
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1225 - accurac
y: 0.9643
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0854 - accurac
y: 0.9742
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0651 - accurac
y: 0.9800
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0517 - accurac
y: 0.9845
```

Out[28]: <keras.callbacks.History at 0x7fc7bade8150>

In [29]:
```
model.evaluate(X_test_flattened, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0875 - accuracy: 0.9740
```

Out[29]:
```
[0.08750998228788376, 0.9739999771118164]
```

In [30]:
```
y_preds = model.predict(X_test_flattened)
```

```
313/313 [==============================] - 1s 2ms/step
```

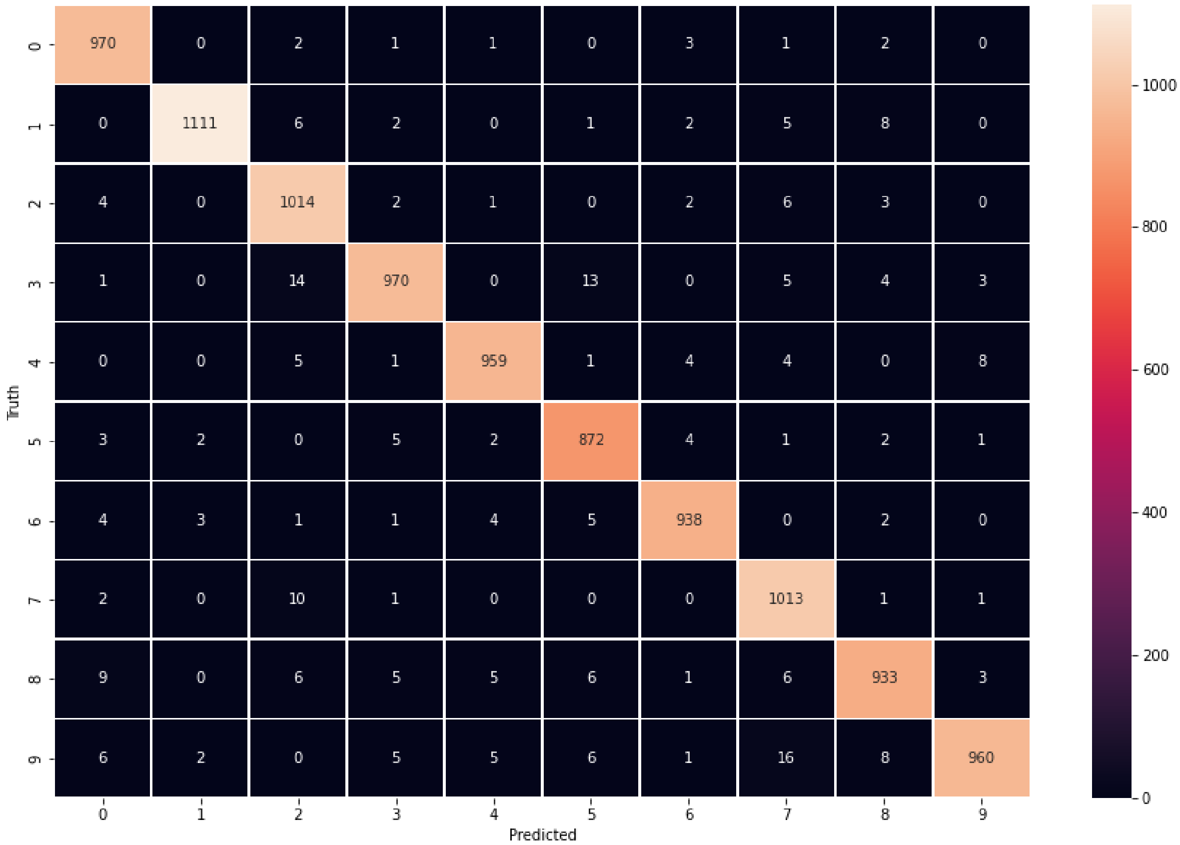
In [31]:
```
y_pred_labels = [np.argmax(y_pred) for y_pred in y_preds]
```

In [32]:
```
cm = tf.math.confusion_matrix(
    labels=y_test,
    predictions=y_pred_labels
)
cm
```

Out[32]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 970,    0,    2,    1,    1,    0,    3,    1,    2,    0],
       [   0, 1111,    6,    2,    0,    1,    2,    5,    8,    0],
       [   4,    0, 1014,    2,    1,    0,    2,    6,    3,    0],
       [   1,    0,   14,  970,    0,   13,    0,    5,    4,    3],
       [   0,    0,    5,    1,  959,    1,    4,    4,    0,    8],
       [   3,    2,    0,    5,    2,  872,    4,    1,    2,    1],
       [   4,    3,    1,    1,    4,    5,  938,    0,    2,    0],
       [   2,    0,   10,    1,    0,    0,    0, 1013,    1,    1],
       [   9,    0,    6,    5,    5,    6,    1,    6,  933,    3],
       [   6,    2,    0,    5,    5,    6,    1,   16,    8,  960]],
      dtype=int32)>
```

In [33]:
```
plt.figure(figsize=(15, 10))
sn.heatmap(
    cm,
    annot=True,
    fmt="d",
    linewidth=0.5,
)
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

Out[33]:
```
Text(114.0, 0.5, 'Truth')
```



- Integrate Faltten with Model

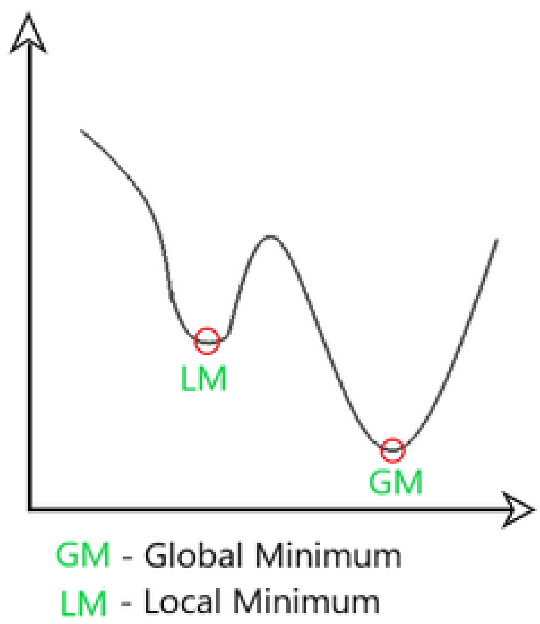
In [34]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid'),
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(X_train, y_train, epochs=5)
```

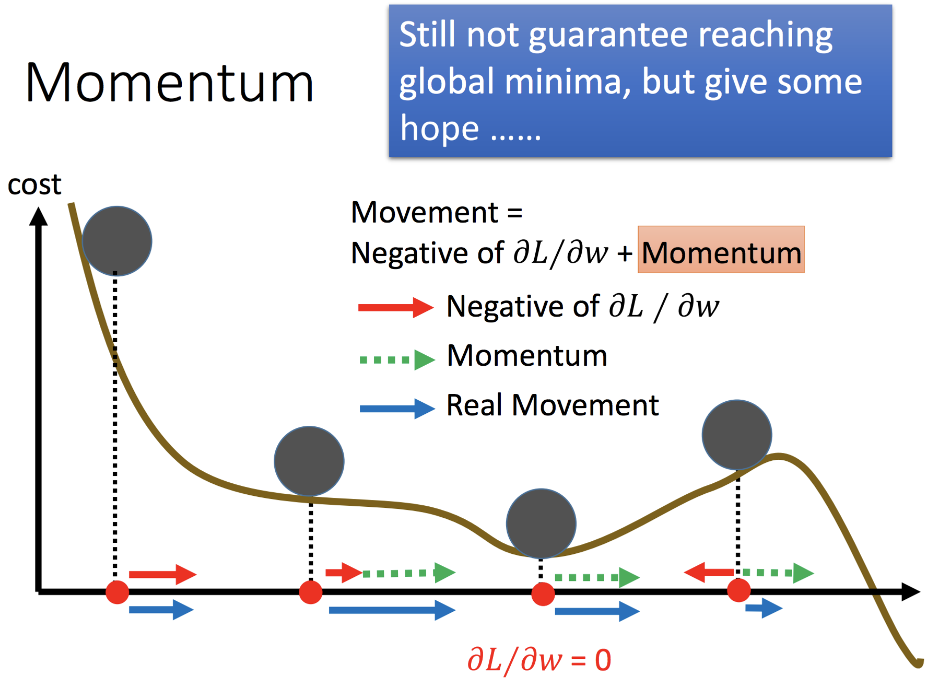
```
Epoch 1/5
1875/1875 [==============================] - 5s 3ms/step - loss: 2.4628 - accurac
y: 0.8286
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3973 - accurac
y: 0.8971
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3051 - accurac
y: 0.9199
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2589 - accurac
y: 0.9318
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2344 - accurac
y: 0.9378
```

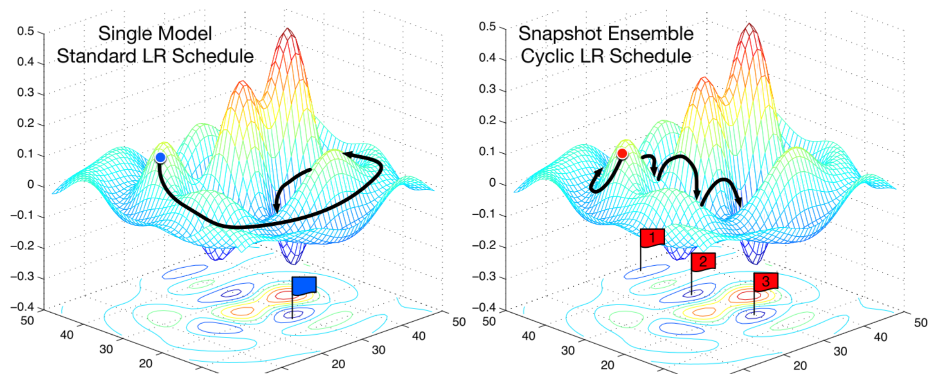
Out[34]:
```
<keras.callbacks.History at 0x7fc7bab34090>
```

Model Improving Thinking

Optimizer







Loss / Cost Function

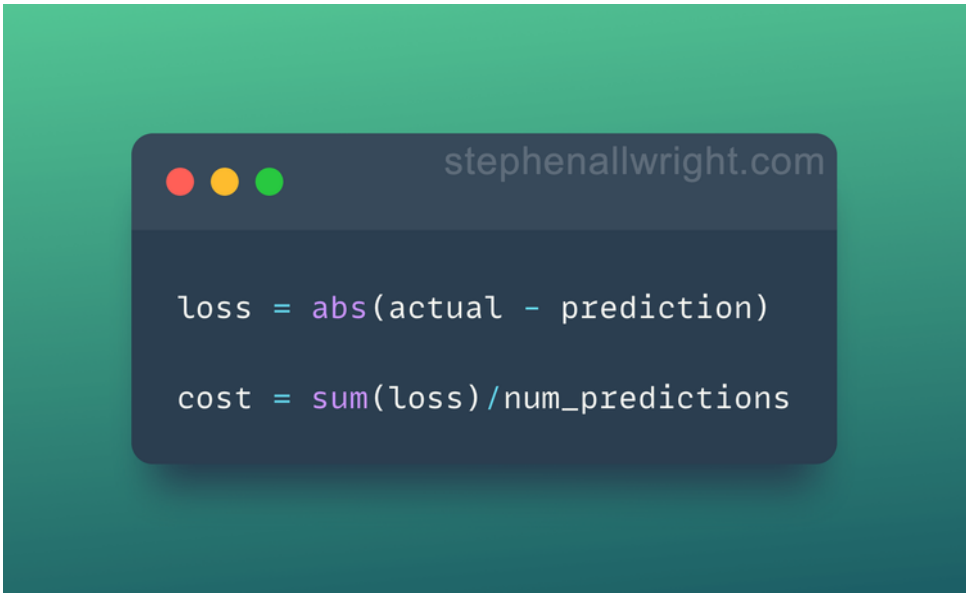
The cost function in logistic regression is given by:

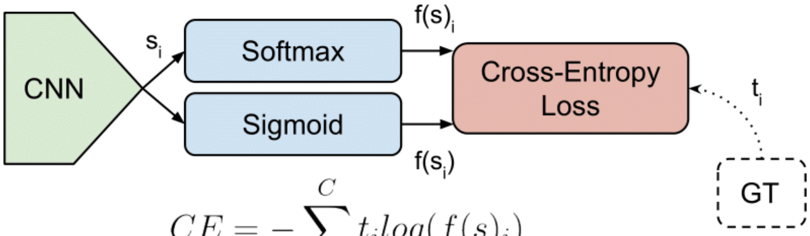
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)}))]$$

and the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0, 1, ..., n) is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Note that while this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of $h_\theta(x)$.





$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1-t_1) log(1-f(s_1))$$

TensorFlow Keras

다중분류 손실함수 (Loss function for multiclass classification) :
sparse_categorical_crossentropy()
vs. categorical_crossentropy()

① **tf.keras.losses.sparse_categorical_crossentropy()**

```
>>> y_true = [1, 2]
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
>>> loss = tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred)
>>> assert loss.shape == (2,)
>>> loss.numpy()
array([0.0513, 2.303], dtype=float32)
```

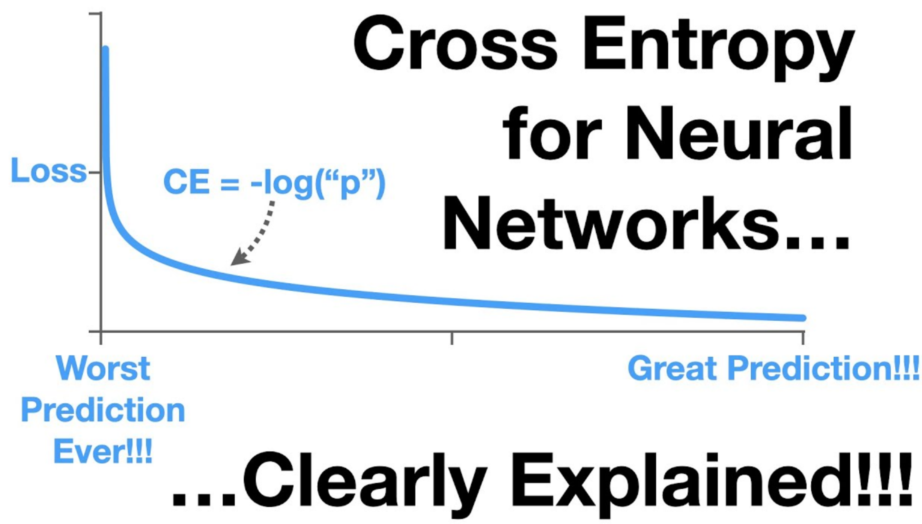
y label: integer
(multiclass)

② **tf.keras.losses.categorical_crossentropy()**

```
>>> y_true = [[0, 1, 0], [0, 0, 1]]
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
>>> loss = tf.keras.losses.categorical_crossentropy(y_true, y_pred)
>>> assert loss.shape == (2,)
>>> loss.numpy()
array([0.0513, 2.303], dtype=float32)
```

y label: one-hot encoded
(multiclass)

[R, Python 분석과 프로그래밍의 친구] https://rfriend.tistory.com

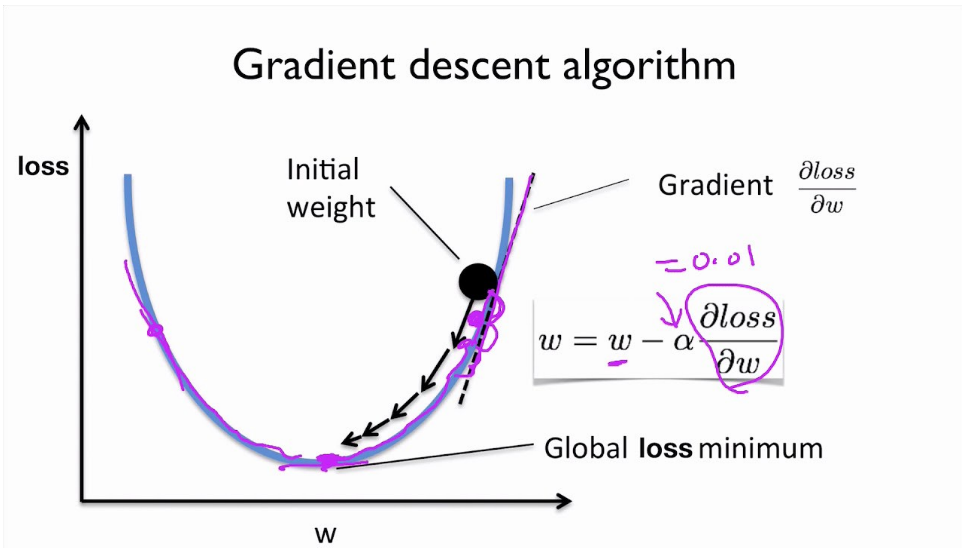


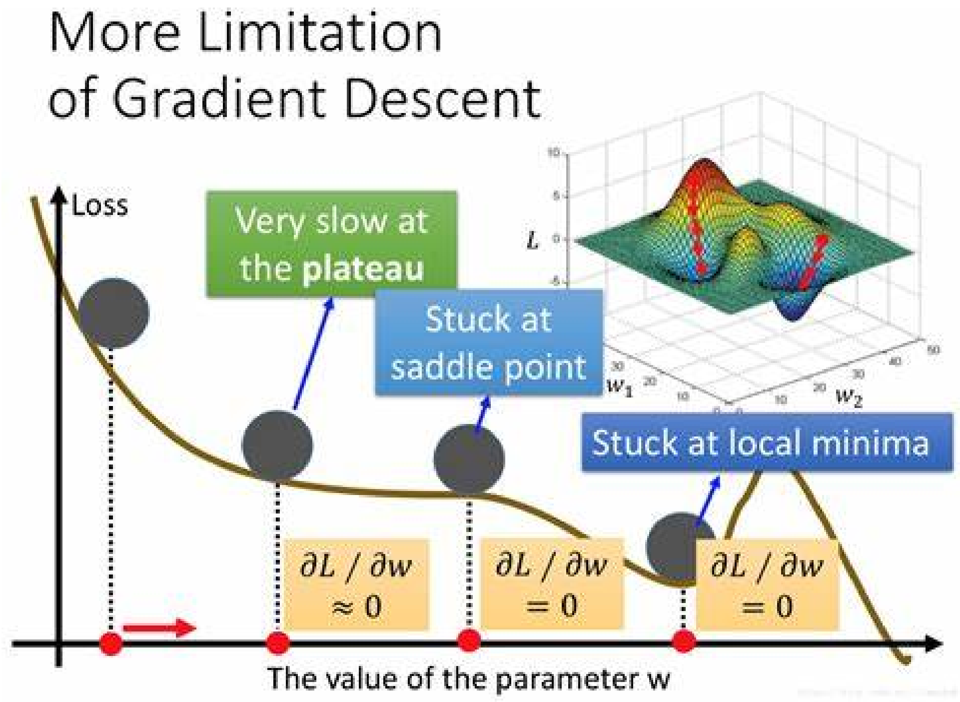
Regression Problems

Linear Regression

- Create the data with known **parameters** (things that can be learned by a model)
- Build model to estimate these parameters using **gradient descent**.

Gradient Descent



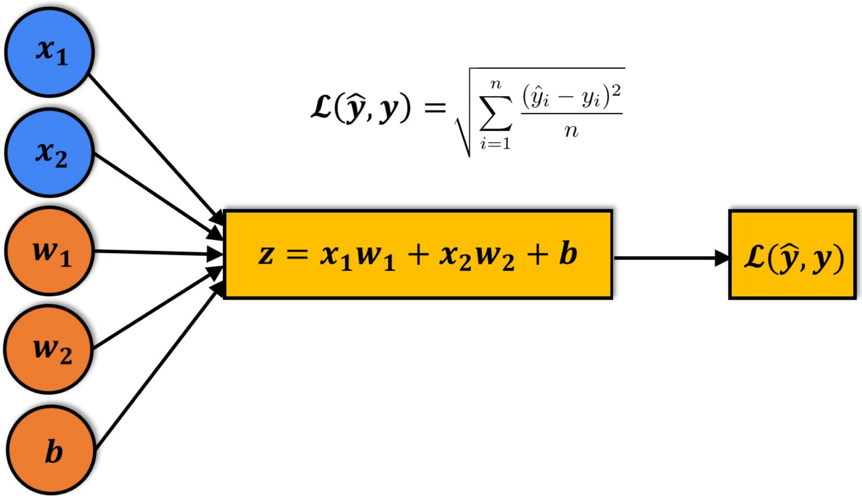


- **RMSE**

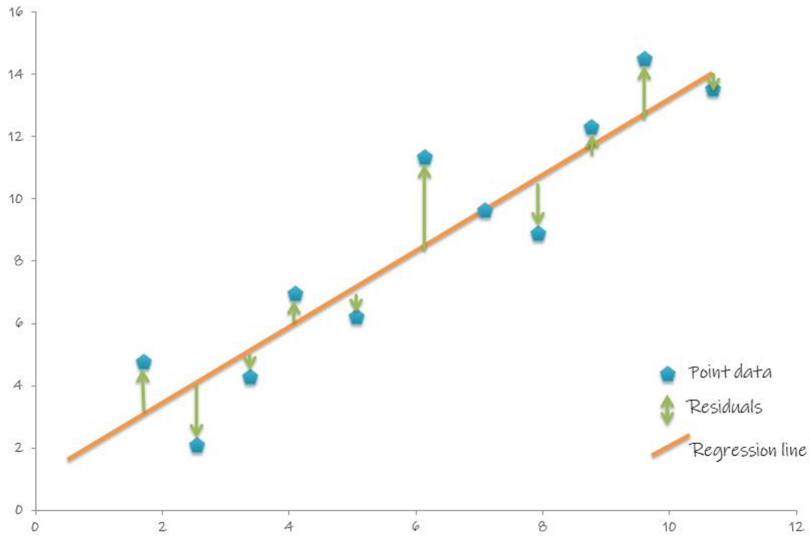
  -> *np.sqrt(np.mean(np.square(targets - predictions)))*

  -> `from sklearn.metrics import mean_squared_error`

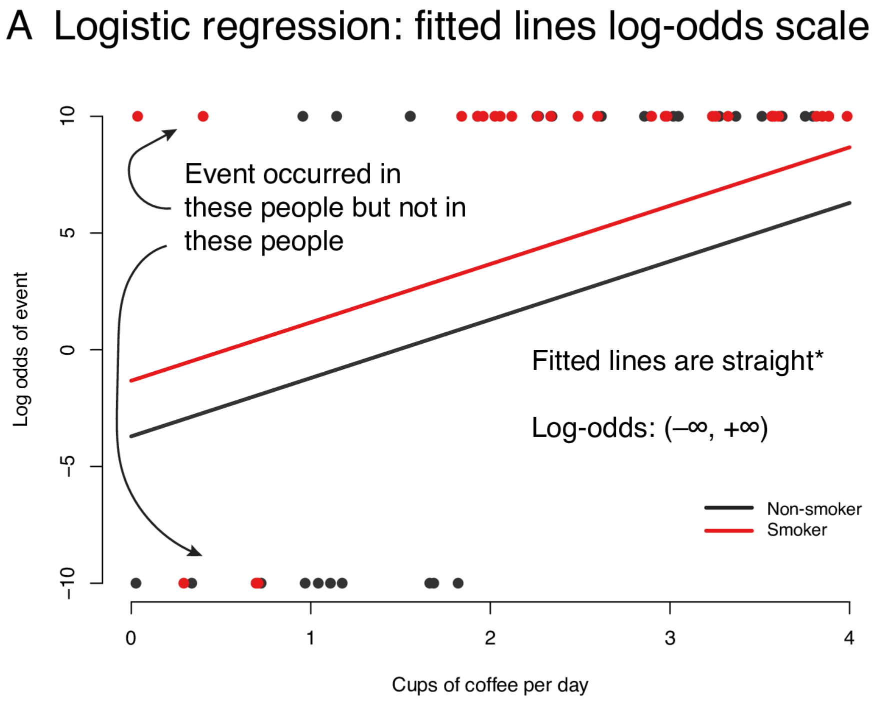
  -> *mean_squared_error(targets, predictions, squared=False)*`

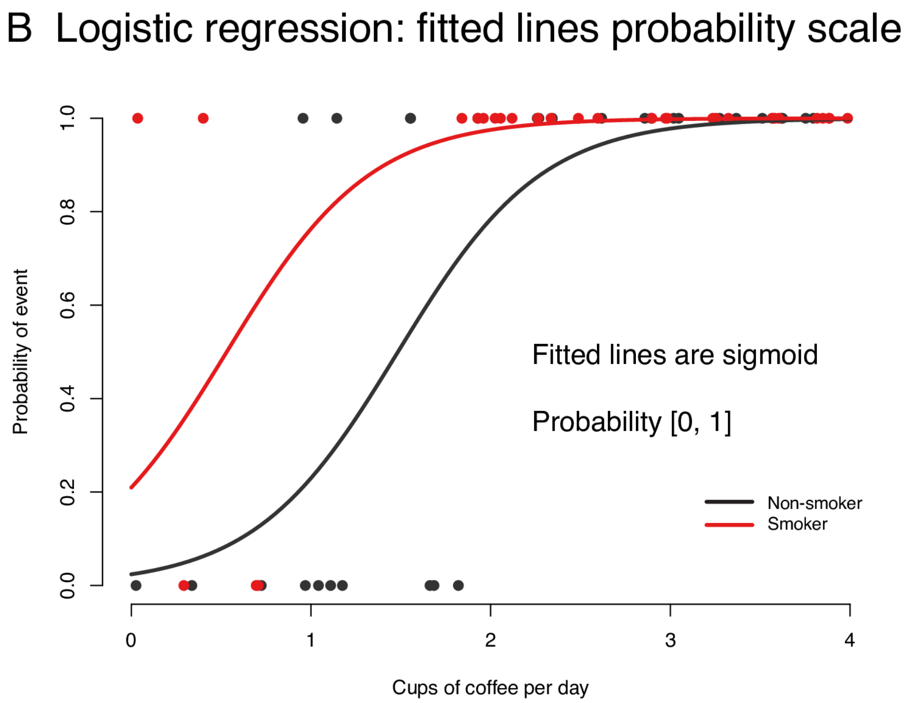


Geometrically, the residuals can be visualized as follows:



Logistic Regression

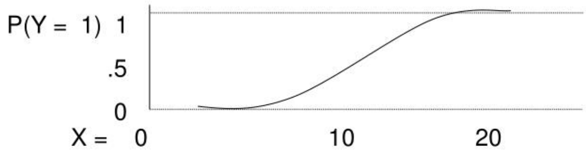

A Logistic regression: fitted lines log-odds scale
Event occurred in these people but not in these people
Fitted lines are straight*
Log-odds: (−∞, +∞)
Non-smoker
Smoker
Log odds of event
10
5
0
−5
−10
0
1
2
3
4
Cups of coffee per day


B Logistic regression: fitted lines probability scale
Fitted lines are sigmoid
Probability [0, 1]
Non-smoker
Smoker
Probability of event
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
Cups of coffee per day

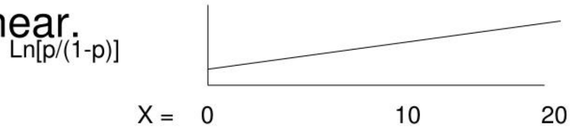
Binary Logistic Regression

- Logistic Distribution

With the logistic transformation, we're fitting the "model" to the data better.



- Transformed, however, the "log odds" are linear.

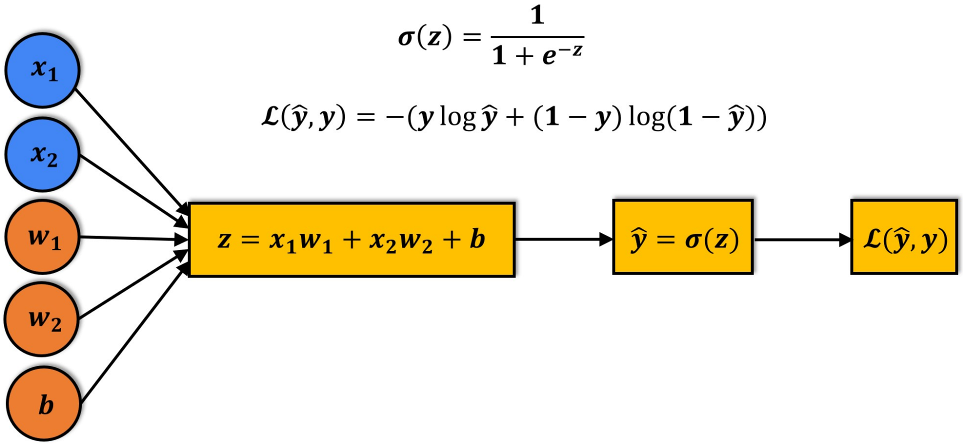


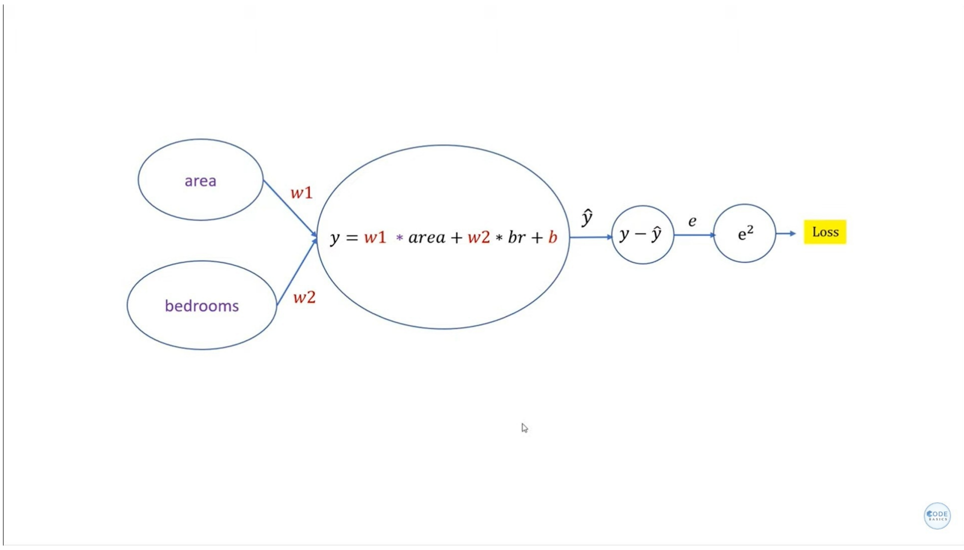
Classification Problems

`cross entropy loss function`

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$





$$\text{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n} abs(y_i - \hat{y}_i)$$

```
model.compile(optimizer='adam',
              loss='mean_absolute_error',
              metrics=['accuracy'])
```

$$\text{Mean Squared Error (MSE)} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

```
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

$$\text{Log loss or binary cross entropy} = -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

- MAE

In [35]:
```
y_predicted = np.array([1, 1, 0, 0, 1])
y_true = np.array([0.3, 0.7, 1, 0, 0.5])
```

In [36]:
```
def mae(y_true, y_predicted):
    total_error = 0
    for yt, yp in zip(y_true, y_predicted):
        total_error += abs(yt - yp)
    print("Total Error: ", total_error)
    mae = total_error / len(y_true)
    print("MAE: ", mae)
    return mae
```

In [37]:
```
mae(y_true, y_predicted)
```

```
Total Error:  2.5
MAE:  0.5
```

Out[37]: `0.5`

- Binary Cross Entropy

In [38]:
```
epsilon = 1e-15
```

In [39]:
```
np.log(epsilon)
```

Out[39]: `-34.538776394910684`

In [40]:
```
y_predicted_new = [max(i, epsilon) for i in y_predicted]
y_predicted_new
```

Out[40]: `[1, 1, 1e-15, 1e-15, 1]`

In [41]:
```
y_predicted_new = [min(i,1-epsilon) for i in y_predicted_new]
y_predicted_new
```

Out[41]: `[0.999999999999999, 0.999999999999999, 1e-15, 1e-15, 0.999999999999999]`

In [42]:
```
np.log(y_predicted_new)
```

Out[42]:
```
array([-9.99200722e-16, -9.99200722e-16, -3.45387764e+01, -3.45387764e+01,
       -9.99200722e-16])
```

In [43]:
```
np.log(y_predicted)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: di
vide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

Out[43]:
```
array([  0.,   0., -inf, -inf,   0.])
```

In [44]:
```
def log_loss(y_true, y_predicted):
    epsilon = 1e-15
    y_predicted_new = [max(i,epsilon) for i in y_predicted]
    y_predicted_new = [min(i,1-epsilon) for i in y_predicted_new]
    y_predicted_new = np.array(y_predicted_new)
    return -np.mean(y_true*np.log(y_predicted_new)+(1-y_true)*np.log(1-y_predicted_
```

In [45]:
```
def sigmoid_numpy(X):
   return 1/(1+np.exp(-X))

sigmoid_numpy(np.array([12,0,1]))
```

Out[45]:
```
array([0.99999386, 0.5       , 0.73105858])
```

- Neural Network

In [46]:
```
class myNN:
    def __init__(self):
        self.w1 = 1
        self.w2 = 1
        self.bias = 0

    def fit(self, X, y, epochs, loss_thresold):
        self.w1, self.w2, self.bias = self.gradient_descent(X ,y, epochs, loss_thre
        print(f"Final weights and bias: w1: {self.w1}, w2: {self.w2}, bias: {self.b

    def predict(self, X):
        weighted_sum = self.w1*X['age'] + self.w2*X['affordability'] + self.bias
        return 1/(1+np.exp(-weighted_sum)) # sigmoid

    def gradient_descent(self, X, y, epochs, loss_thresold):
        w1=w2=1
        bias=0
        rate=0.5
        n = len(X['age'])
        for i in range(epochs):
            weighted_sum = w1 * X['age'] + w2 * X['affordability'] + bias
            y_predicted = 1/(1+np.exp(-weighted_sum)) # sigmoid
            loss = log_loss(y, y_predicted)

            w1d = (1/n)*np.dot(np.transpose(X['age']),(y_predicted-y))
            w2d = (1/n)*np.dot(np.transpose(X['affordability']),(y_predicted-y))
            bias_d = np.mean(y_predicted-y)

            w1 = w1 - rate * w1d
            w2 = w2 - rate * w2d
```

```
            bias = bias - rate * bias_d

            if i%50==0:
                print (f'Epoch:{i}, w1:{w1}, w2:{w2}, bias:{bias}, loss:{loss}')

            if loss<=loss_thresold:
                print (f'Epoch:{i}, w1:{w1}, w2:{w2}, bias:{bias}, loss:{loss}')
                break

        return self.w1, self.w2, self.bias
```

In [47]:
```
def gradient_descent(age, affordability, y_true, epochs):
    w1 = 1,
    w2 = 1,
    bias = 0,
    learning_rate = 0.5,
    n = len(age)

    for i in range(epochs):
        weight_sum = w1 * age + w2 * affordability + bias
        y_predicted = sigmoid_numpy(weight_sum)

        loss = log_loss(y_true, y_predicted)

        w1_d = (1/n)*np.dot(np.transpose(age), (y_predicted - y_true))
        w2_d = (1/n)*np.dot(np.transpose(affordability), (y_predicted - y_true))
        bias_d = np.mean(y_predicted - y_true)

        w1 = w1 - learning_rate * w1_d
        w2 = w2 - learning_rate * w2_d
        bias = bias - learning_rate * bias_d

        print(f"Epoch-{i}: , w1: {w1}, w2: {w2}, bias: {bias}, loss: {loss}")

    return w1, w2, bias
```

In [48]:
```
def gradient_descent(age, affordability, y_true, epochs, loss_thresold):
    w1 = w2 = 1
    bias = 0
    learning_rate = 0.5
    n = len(age)
    for i in range(epochs):
        weighted_sum = w1 * age + w2 * affordability + bias
        y_predicted = sigmoid_numpy(weighted_sum)
        loss = log_loss(y_true, y_predicted)

        w1d = (1/n)*np.dot(np.transpose(age),(y_predicted-y_true))
        w2d = (1/n)*np.dot(np.transpose(affordability),(y_predicted-y_true))
        bias_d = np.mean(y_predicted-y_true)

        w1 = w1 - learning_rate * w1d
        w2 = w2 - learning_rate * w2d
        bias = bias - learning_rate * bias_d

        print (f'Epoch:{i}, w1:{w1}, w2:{w2}, bias:{bias}, loss:{loss}')

        if loss<=loss_thresold:
            break

    return w1, w2, bias
```

> - Batch Gradient Descent

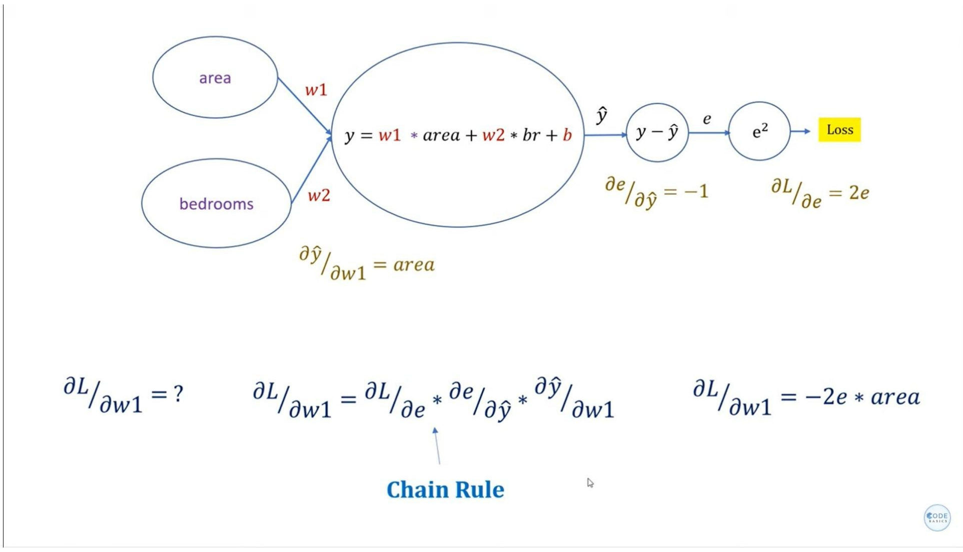
In [49]:
```
def batch_gradient_descent(X, y, epochs, learning_rate=0.01):
    number_of_features = X.shape[1]
    w = np.ones(shape=number_of_features)
    bias = 0
    total_sample = X.shape[0]

    epoch_list = []
    cost_list = []

    for i in range(epochs):
        y_predicted = np.dot(w, X.T) + bias

    w_grad = -(2/total_sample)*X.T.dot(y-y_predicted)
    b_grad = -(2/total_sample)*np.sum(y-y_predicted)

    w = w - learning_rate * w_grad
    bias = bias - learning_rate * b_grad

    cost = np.mean(np.square(y-y_predicted))

    if i%10==0:
        epoch_list.append(i)
        cost_list.append(cost)

    return w, bias, cost, cost_list, epoch_list
```

> - Stochastic Gradient Descent

In [50]:
```
def stochastic_gradient_descent(X, y_true, epochs, learning_rate = 0.01):

    number_of_features = X.shape[1]
    # numpy array with 1 row and columns equal to number of features. In
    # our case number_of_features = 3 (area, bedroom and age)
    w = np.ones(shape=(number_of_features))
    b = 0
    total_samples = X.shape[0]

    cost_list = []
    epoch_list = []

    for i in range(epochs):
        random_index = random.randint(0,total_samples-1) # random index from total
        sample_x = X[random_index]
        sample_y = y_true[random_index]

        y_predicted = np.dot(w, sample_x.T) + b

        w_grad = -(2/total_samples)*(sample_x.T.dot(sample_y-y_predicted))
        b_grad = -(2/total_samples)*(sample_y-y_predicted)

        w = w - learning_rate * w_grad
        b = b - learning_rate * b_grad

        cost = np.square(sample_y-y_predicted)
```

```
        if i%100==0: # at every 100th iteration record the cost and epoch value
            cost_list.append(cost)
            epoch_list.append(i)

    return w, b, cost, cost_list, epoch_list
```

- Chain Rule



Matrix Basic

In [51]:
```
revenue = np.array([[180, 200, 220], [24, 36, 40], [12, 18, 20]])
expense = np.array([[80, 90, 100], [10, 16, 20], [8, 10, 10]])

profit = revenue - expense

profit
```

Out[51]:
```
array([[100, 110, 120],
       [ 14,  20,  20],
       [  4,   8,  10]])
```

In [52]:
```
price_per_unit = np.array([1000, 400, 1200])
unit = np.array([[30, 40, 50], [5, 10, 15], [2, 5, 7]])
```

In [53]:
```
price_per_unit
```

Out[53]:
```
array([1000,  400, 1200])
```

In [54]:
```
unit
```

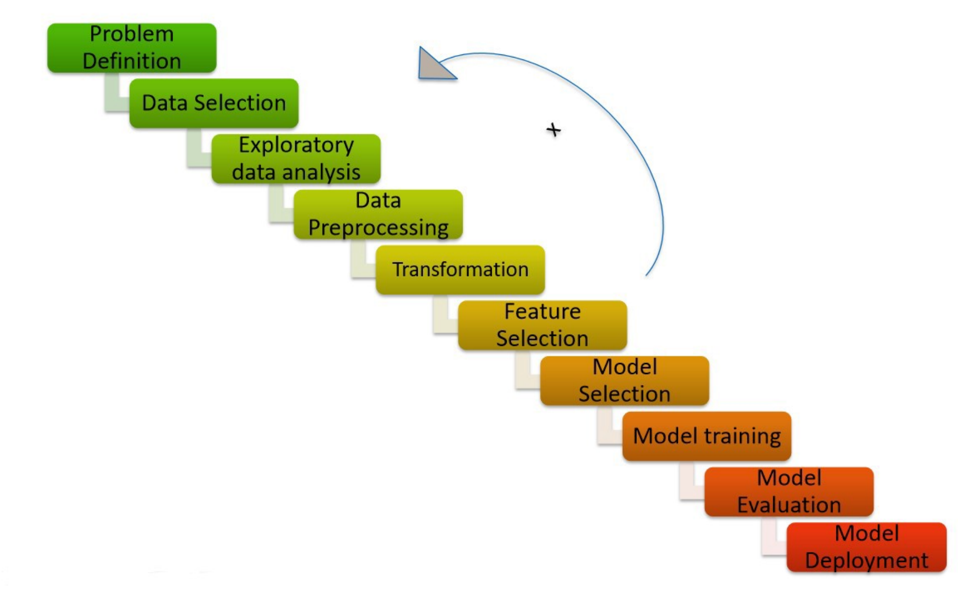
Out[54]:
```
array([[30, 40, 50],
       [ 5, 10, 15],
       [ 2,  5,  7]])
```

In [55]:
```
np.dot(price_per_unit, unit)
```

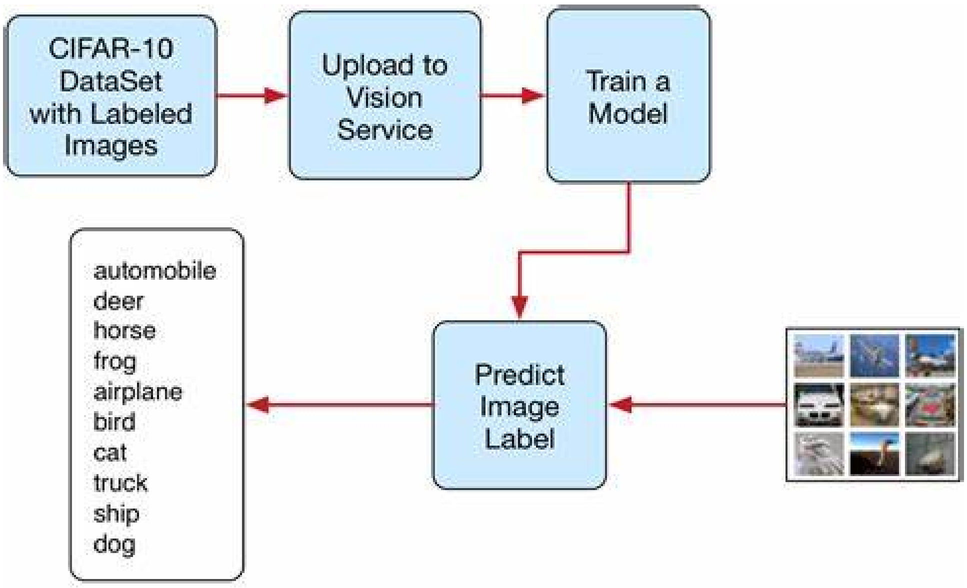
Out[55]:
```
array([34400, 50000, 64400])
```

Project 2 - Aritifical Neural Network for Image Classifieration

> The CIFAR-10 dataset



> ## Get Data
>
> The CIFAR-10 dataset



In [56]:
```
(X_train, y_train), (X_test, y_test) =  keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

Exploratory Data

In [57]:
```
X_train.shape
```
Out[57]:
```
(50000, 32, 32, 3)
```

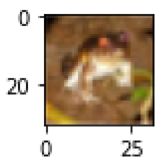
In [58]:
```
y_train.shape
```
Out[58]:
```
(50000, 1)
```

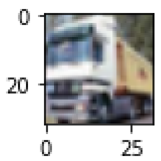
In [59]:
```
X_test.shape
```
Out[59]:
```
(10000, 32, 32, 3)
```

In [60]:
```
y_test.shape
```
Out[60]:
```
(10000, 1)
```

In [61]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship",

def plot_sample(index):
    plt.figure(figsize = (1,1))
    plt.imshow(X_train[index])
```

In [62]:
```
y_train[:10]
```
Out[62]:
```
array([[6],
       [9],
       [9],
       [4],
       [1],
       [1],
       [2],
       [7],
       [8],
       [3]], dtype=uint8)
```

In [63]:
```
plot_sample(0)
```


In [64]:
```
plot_sample(1)
```


Preprocess Data

- Scaling

In [65]:
```
X_train_scaled = X_train / 255
X_test_scaled = X_test / 255
```

In [66]:
```
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32,32,3)),
        keras.layers.Dense(3000, activation='relu'),
        keras.layers.Dense(1000, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])

model.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scaled, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 100s 64ms/step - loss: 1.8130 - accur
acy: 0.3531
Epoch 2/10
1563/1563 [==============================] - 94s 60ms/step - loss: 1.6219 - accura
cy: 0.4298
Epoch 3/10
1563/1563 [==============================] - 93s 60ms/step - loss: 1.5411 - accura
cy: 0.4572
Epoch 4/10
1563/1563 [==============================] - 96s 61ms/step - loss: 1.4811 - accura
cy: 0.4794
Epoch 5/10
1563/1563 [==============================] - 98s 63ms/step - loss: 1.4325 - accura
cy: 0.4953
Epoch 6/10
1563/1563 [==============================] - 96s 62ms/step - loss: 1.3927 - accura
cy: 0.5120
Epoch 7/10
1563/1563 [==============================] - 96s 62ms/step - loss: 1.3508 - accura
cy: 0.5244
Epoch 8/10
1563/1563 [==============================] - 98s 62ms/step - loss: 1.3174 - accura
cy: 0.5365
Epoch 9/10
1563/1563 [==============================] - 96s 61ms/step - loss: 1.2827 - accura
cy: 0.5498
Epoch 10/10
1563/1563 [==============================] - 96s 61ms/step - loss: 1.2498 - accura
cy: 0.5613
```

Out[66]:
```
<keras.callbacks.History at 0x7fc7b1bbeed0>
```

- One Hot Encoding

In [97]:
```
y_train_categorical = keras.utils.to_categorical(
    y_train, num_classes=10, dtype='float32'
)
y_test_categorical = keras.utils.to_categorical(
    y_test, num_classes=10, dtype='float32'
)
```

In [68]:
```
y_train[:5]
```

Out[68]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

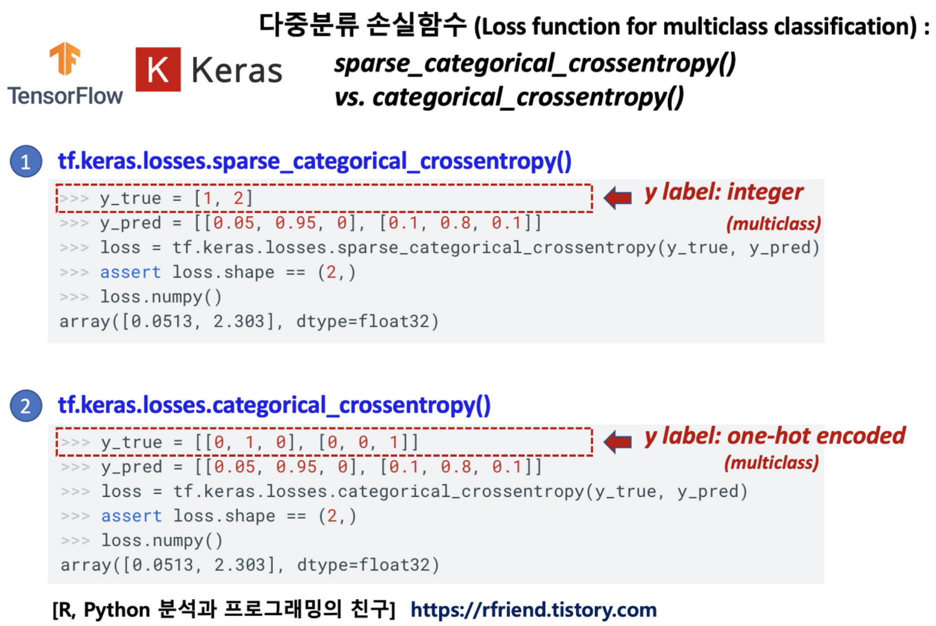
In [69]:
```
y_train_categorical[:5]
```

Out[69]:
```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

In [70]:
```
y_test_categorical[:5]
```

Out[70]:
```
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]], dtype=float32)
```

Model Building and Traning



In [71]:
```
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32,32,3)),
        keras.layers.Dense(3000, activation='relu'),
        keras.layers.Dense(1000, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])

model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scaled, y_train_categorical, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 94s 60ms/step - loss: 1.8097 - accura
cy: 0.3534
Epoch 2/10
1563/1563 [==============================] - 96s 62ms/step - loss: 1.6217 - accura
cy: 0.4297
Epoch 3/10
1563/1563 [==============================] - 95s 61ms/step - loss: 1.5397 - accura
cy: 0.4581
Epoch 4/10
1563/1563 [==============================] - 100s 64ms/step - loss: 1.4827 - accur
acy: 0.4762
Epoch 5/10
1563/1563 [==============================] - 98s 63ms/step - loss: 1.4319 - accura
cy: 0.4958
Epoch 6/10
1563/1563 [==============================] - 97s 62ms/step - loss: 1.3916 - accura
cy: 0.5112
Epoch 7/10
1563/1563 [==============================] - 99s 63ms/step - loss: 1.3549 - accura
cy: 0.5233
Epoch 8/10
1563/1563 [==============================] - 97s 62ms/step - loss: 1.3169 - accura
cy: 0.5378
Epoch 9/10
1563/1563 [==============================] - 97s 62ms/step - loss: 1.2836 - accura
cy: 0.5486
Epoch 10/10
1563/1563 [==============================] - 98s 63ms/step - loss: 1.2561 - accura
cy: 0.5607
```

Out[71]:

```
<keras.callbacks.History at 0x7fc7b1b44150>
```

In [72]:

```python
y_preds = model.predict(X_test_scaled)
```

```
313/313 [==============================] - 7s 23ms/step
```

In [73]:

```python
y_preds[:5]
```

Out[73]:

```
array([[0.48503116, 0.7009101 , 0.7252337 , 0.8752738 , 0.7465099 ,
        0.6314355 , 0.66172177, 0.02758165, 0.7579216 , 0.05951715],
       [0.83940506, 0.9579895 , 0.4146872 , 0.16014317, 0.3224557 ,
        0.06487381, 0.05274982, 0.07824815, 0.98045266, 0.9349547 ],
       [0.98708326, 0.9589655 , 0.37635937, 0.16620606, 0.3204554 ,
        0.08185194, 0.00592267, 0.16839802, 0.97674906, 0.8228345 ],
       [0.94194686, 0.52145004, 0.7490967 , 0.30454218, 0.8145238 ,
        0.18877065, 0.02294317, 0.53637576, 0.92732984, 0.1422694 ],
       [0.16433033, 0.07534318, 0.8490903 , 0.6963421 , 0.9894589 ,
        0.6498679 , 0.9476556 , 0.17545243, 0.22228247, 0.02557657]],
      dtype=float32)
```

In [74]:

```python
y_pred_labels = [np.argmax(y_pred) for y_pred in y_preds]
y_pred_labels[:5]
```

Out[74]:

```
[3, 8, 0, 0, 4]
```

In [75]:

```python
classes[y_pred_labels[0]]
```

Out[75]:

```
'cat'
```

In [76]:

```python
classes[y_test[0][0]]
```

Out[76]: 'cat'

In [77]:
```
classes[y_pred_labels[1]]
```

Out[77]: 'ship'

In [78]:
```
classes[y_test[1][0]]
```

Out[78]: 'ship'

Model Evaluation

In [79]:
```
model.evaluate(X_test_scaled, y_test_categorical)
```

```
313/313 [==============================] - 7s 23ms/step - loss: 1.3652 - accuracy:
0.5127
```

Out[79]: [1.365151047706604, 0.5127000212669373]

In [ ]:
```
from sklearn.metrics import confusion_matrix , classification_report
```

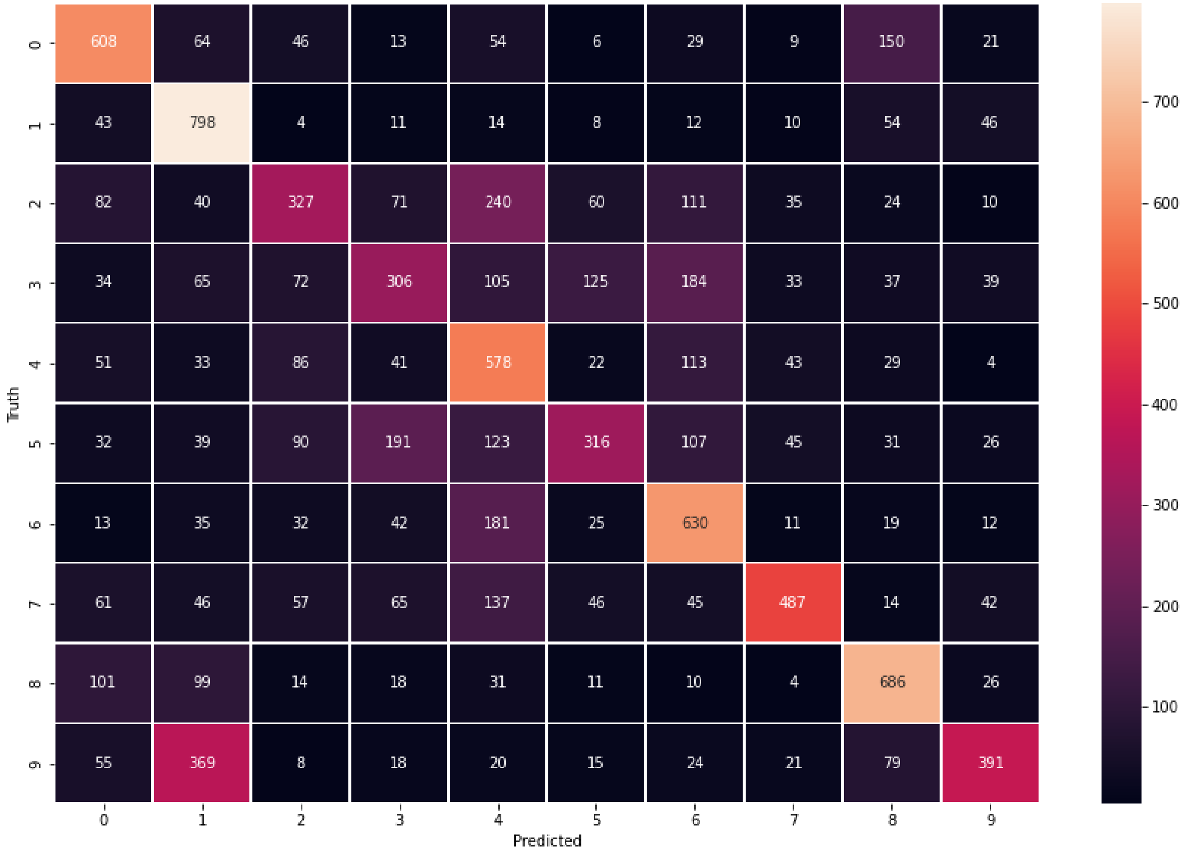
In [81]:
```
y_test_categorical_labels = [np.argmax(y_test_categorical_label) for y_test_categor
```

In [82]:
```
cm = tf.math.confusion_matrix(
    labels=y_test_categorical_labels,
    predictions=y_pred_labels
)
cm
```

Out[82]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[608,  64,  46,  13,  54,   6,  29,   9, 150,  21],
       [ 43, 798,   4,  11,  14,   8,  12,  10,  54,  46],
       [ 82,  40, 327,  71, 240,  60, 111,  35,  24,  10],
       [ 34,  65,  72, 306, 105, 125, 184,  33,  37,  39],
       [ 51,  33,  86,  41, 578,  22, 113,  43,  29,   4],
       [ 32,  39,  90, 191, 123, 316, 107,  45,  31,  26],
       [ 13,  35,  32,  42, 181,  25, 630,  11,  19,  12],
       [ 61,  46,  57,  65, 137,  46,  45, 487,  14,  42],
       [101,  99,  14,  18,  31,  11,  10,   4, 686,  26],
       [ 55, 369,   8,  18,  20,  15,  24,  21,  79, 391]], dtype=int32)>
```

In [83]:
```
plt.figure(figsize=(15, 10))
sn.heatmap(
    cm,
    annot=True,
    fmt="d",
    linewidth=0.5,
)
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

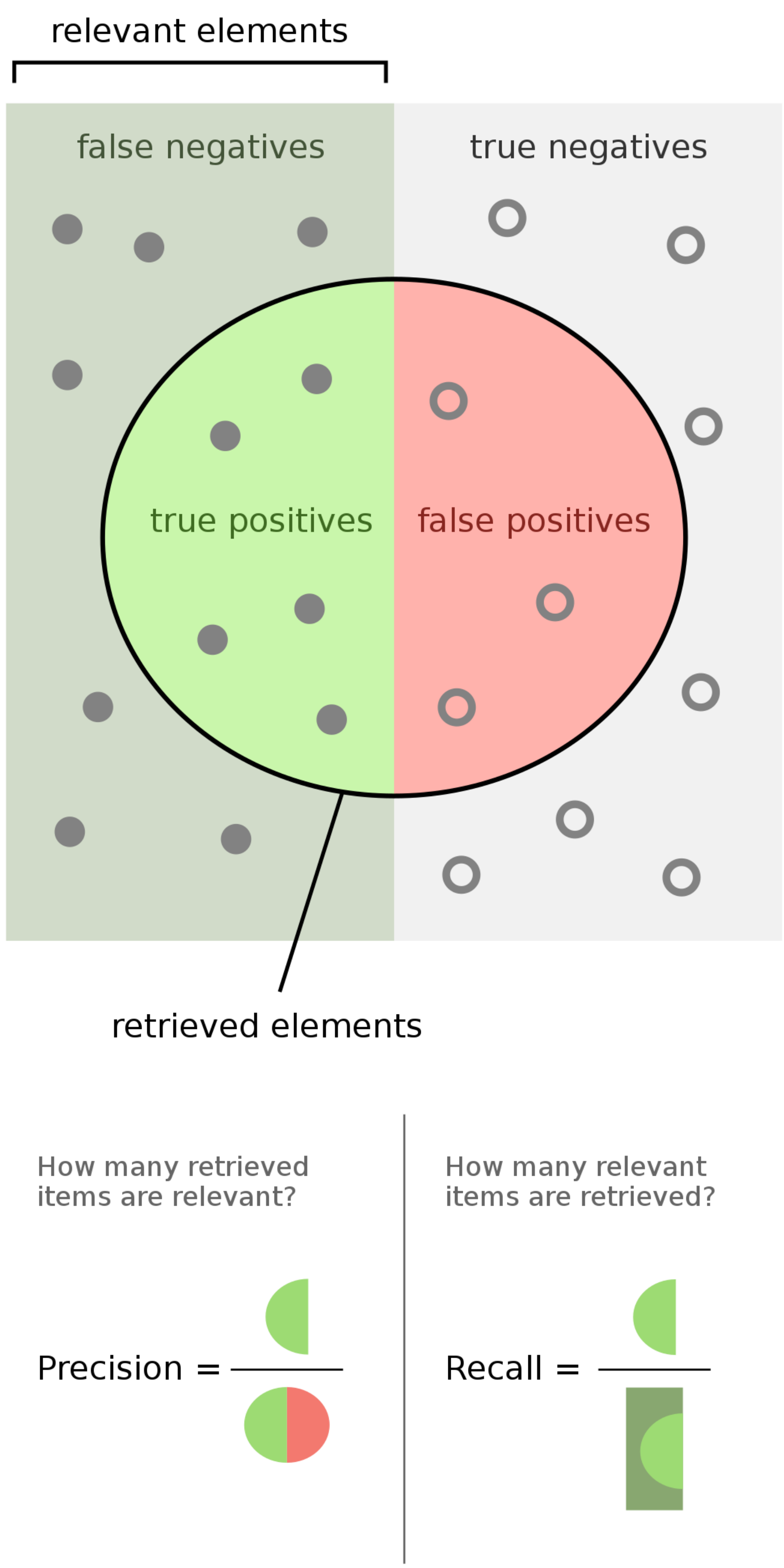
Out[83]: Text(114.0, 0.5, 'Truth')



In [84]:
```
print(classification_report(y_test_categorical_labels, y_pred_labels))
```

```
              precision    recall  f1-score   support

           0       0.56      0.61      0.58      1000
           1       0.50      0.80      0.62      1000
           2       0.44      0.33      0.38      1000
           3       0.39      0.31      0.34      1000
           4       0.39      0.58      0.47      1000
           5       0.50      0.32      0.39      1000
           6       0.50      0.63      0.56      1000
           7       0.70      0.49      0.57      1000
           8       0.61      0.69      0.65      1000
           9       0.63      0.39      0.48      1000

    accuracy                           0.51     10000
   macro avg       0.52      0.51      0.50     10000
weighted avg       0.52      0.51      0.50     10000
```

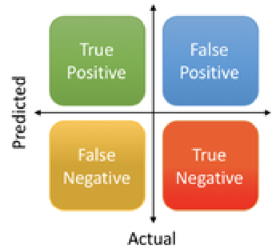
Precision / Recall

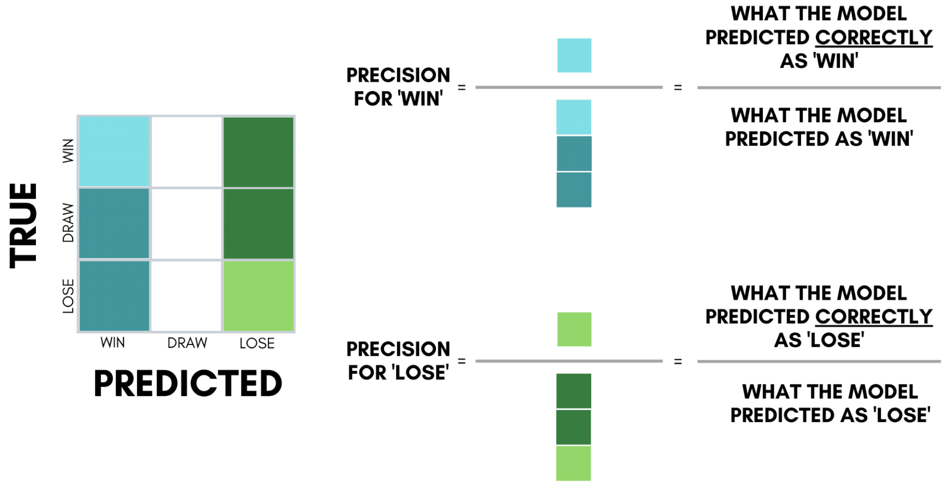

relevant elements
false negatives
true negatives
true positives
false positives
retrieved elements
How many retrieved items are relevant?
How many relevant items are retrieved?
Precision =
Recall =

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$

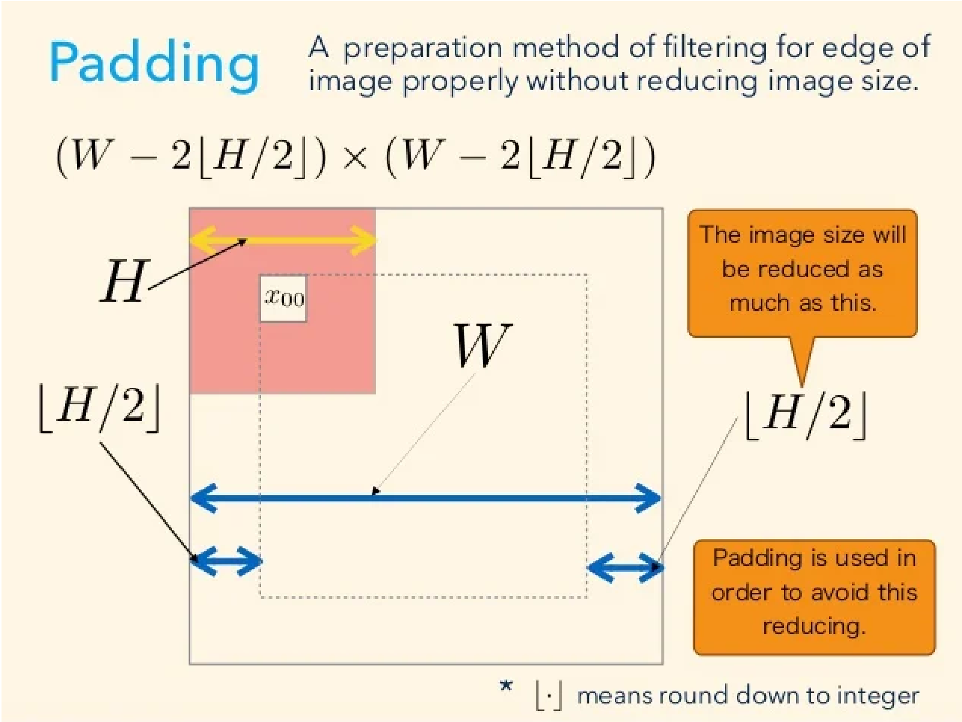


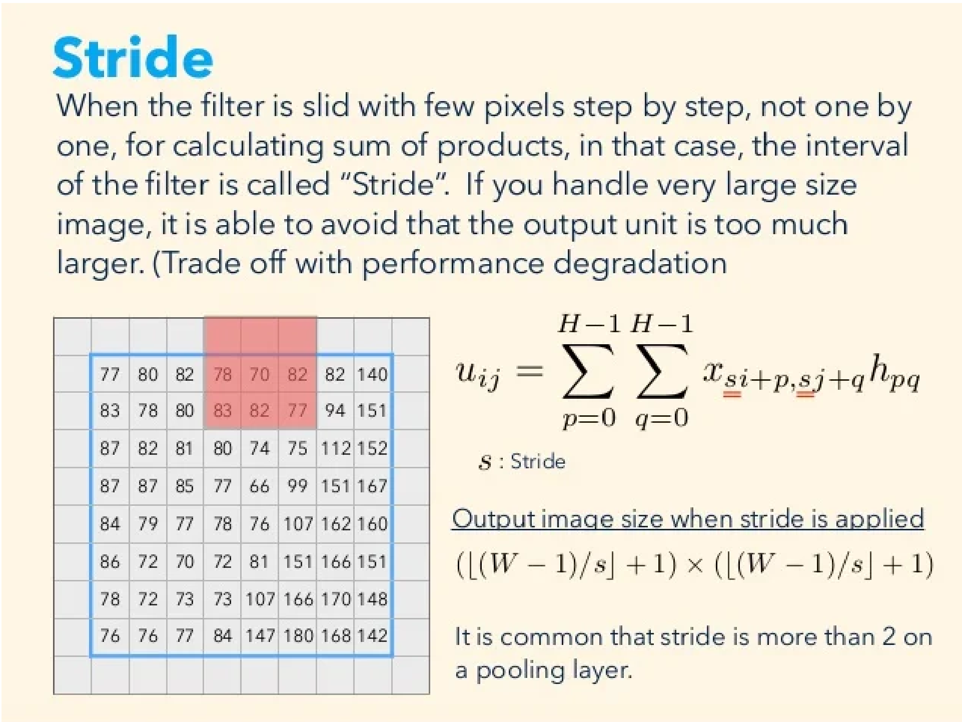


F1 Score

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision + Recall}}$$

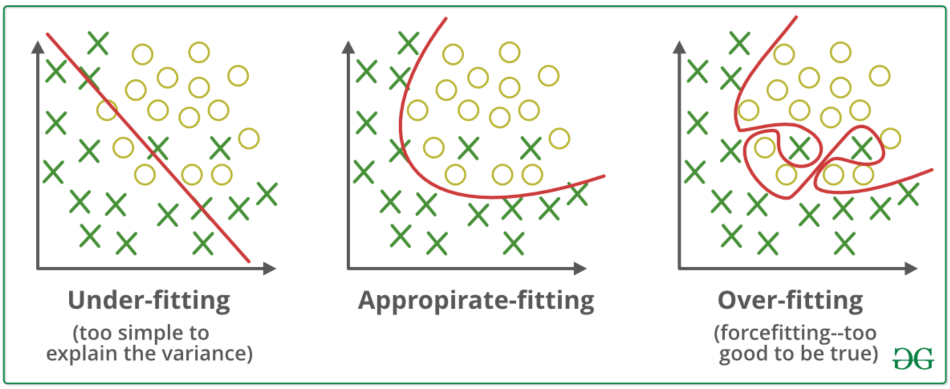
Padding and Stride

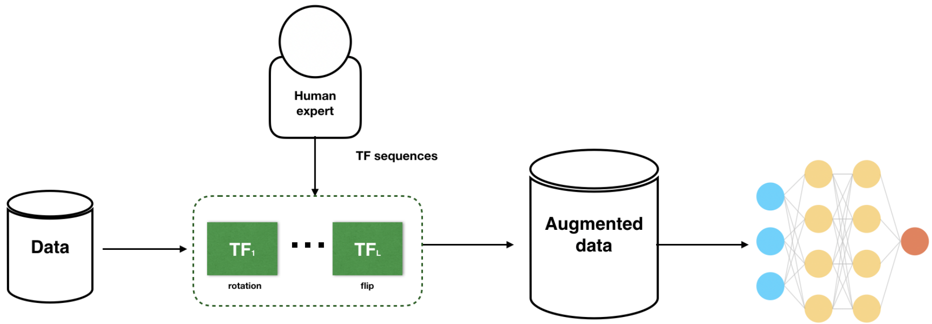


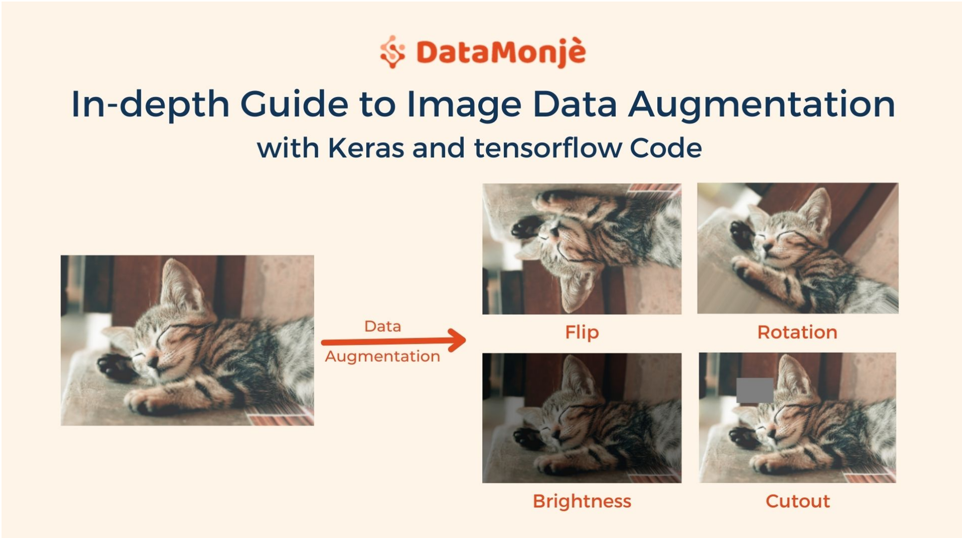


77	80	82	78	70	82	82	140
83	78	80	83	82	77	94	151
87	82	81	80	74	75	112	152
87	87	85	77	66	99	151	167
84	79	77	78	76	107	162	160
86	72	70	72	81	151	166	151
78	72	73	73	107	166	170	148
76	76	77	84	147	180	168	142

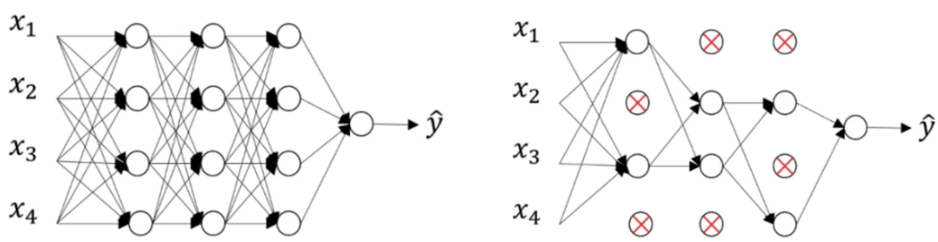
Data Augmentation to Address Overfitting

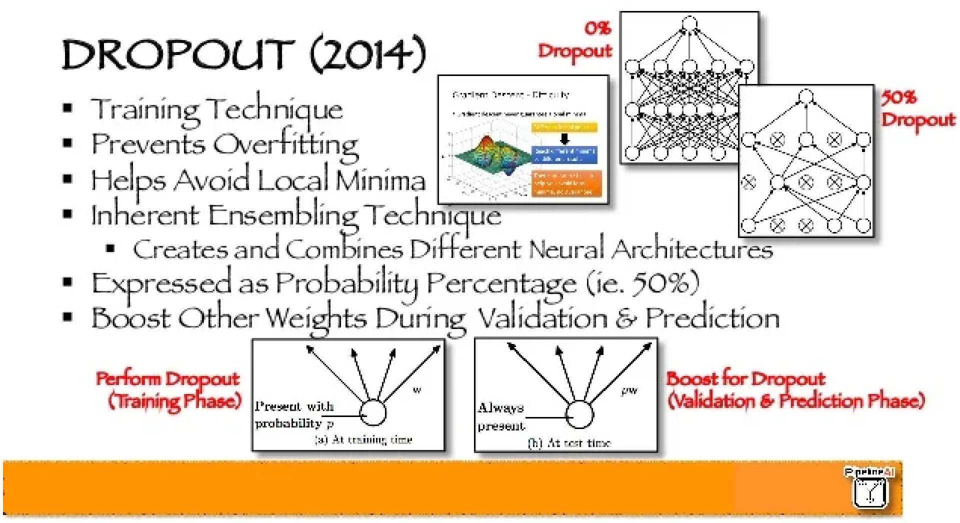




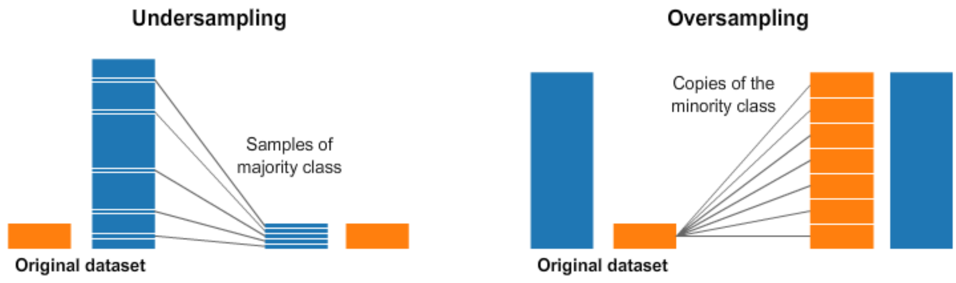


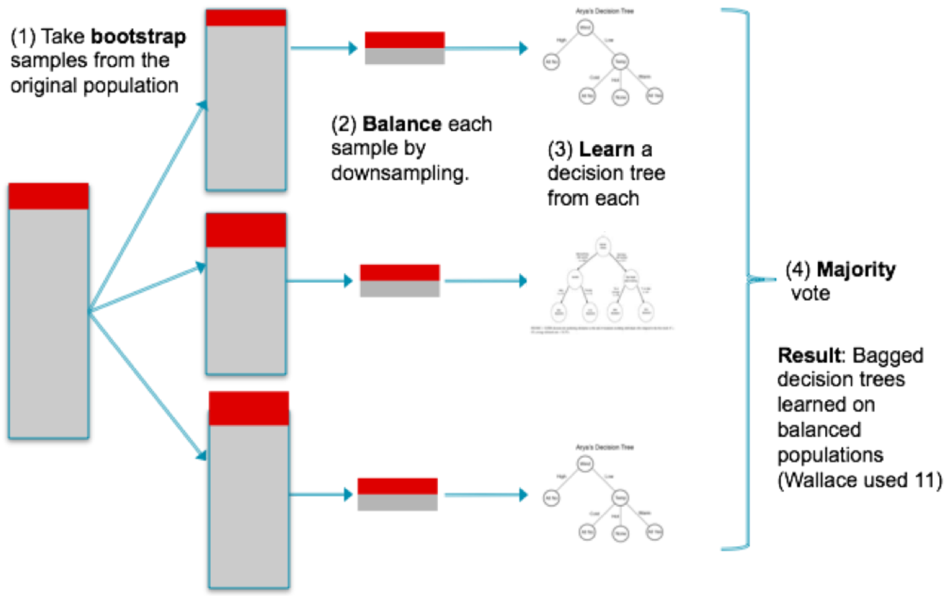
Dropout to Address Overfitting





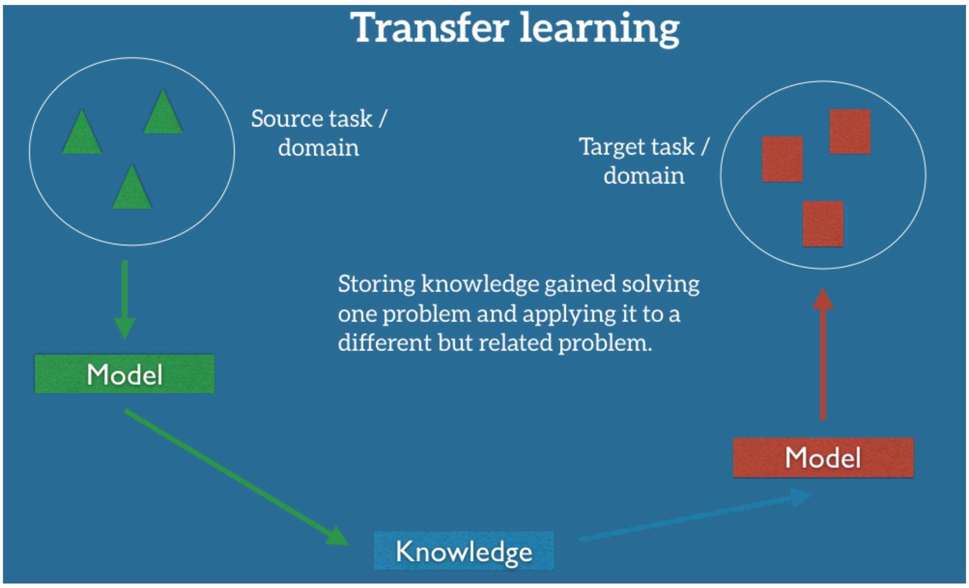
Dealing with Imbalanced Datasets

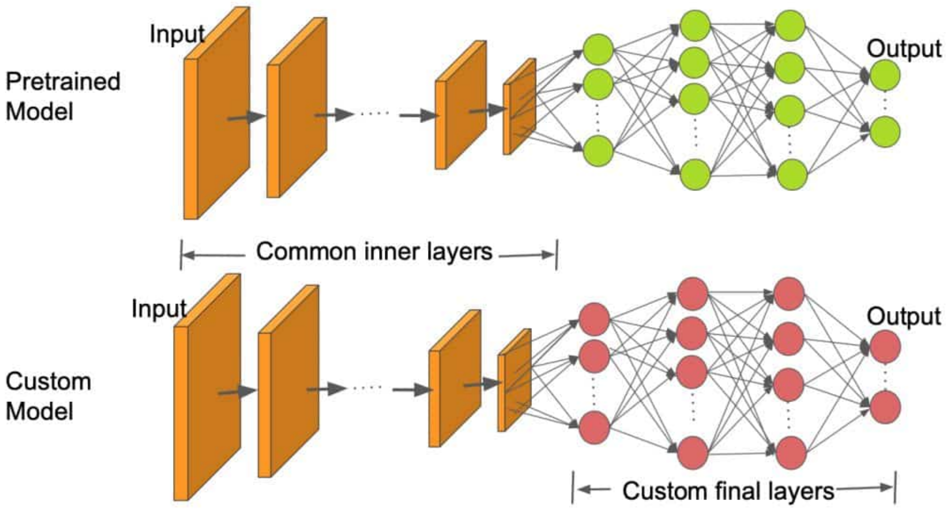




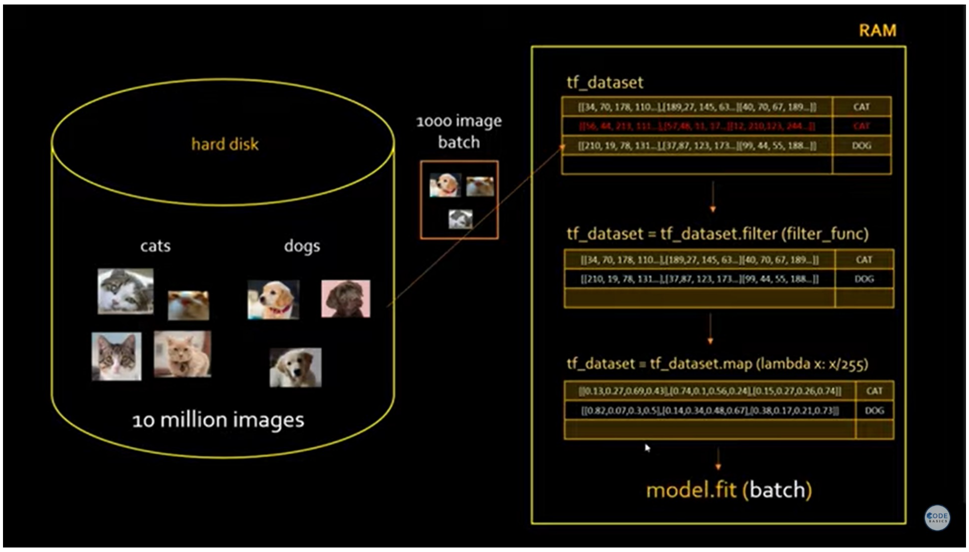
Transfer Learning

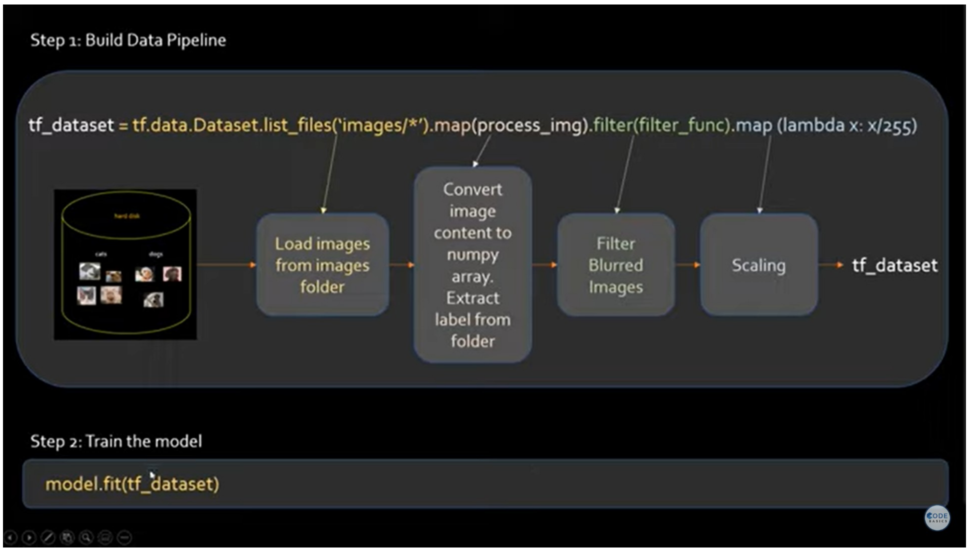
> [TensorFlow Hub](#)
> [tf2-preview/mobilenet_v2/classification](#)

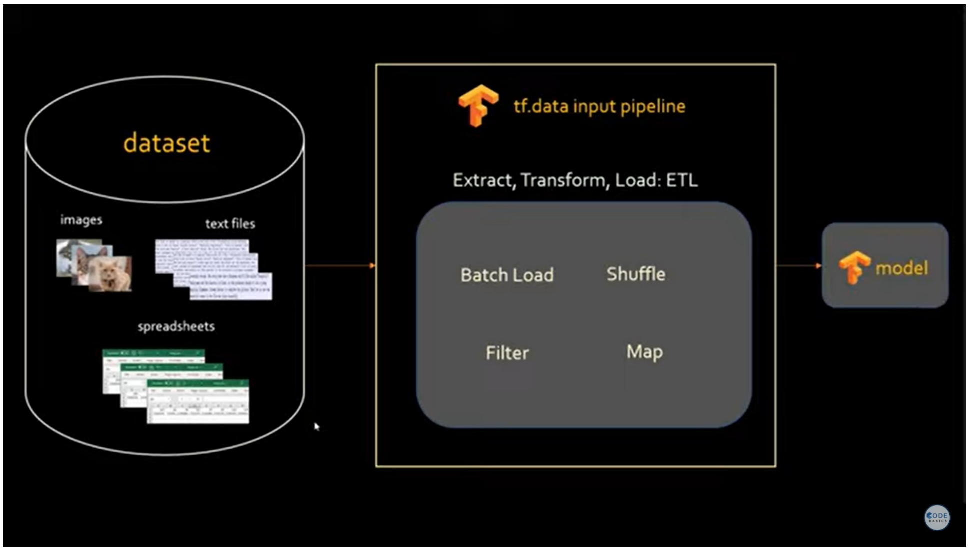


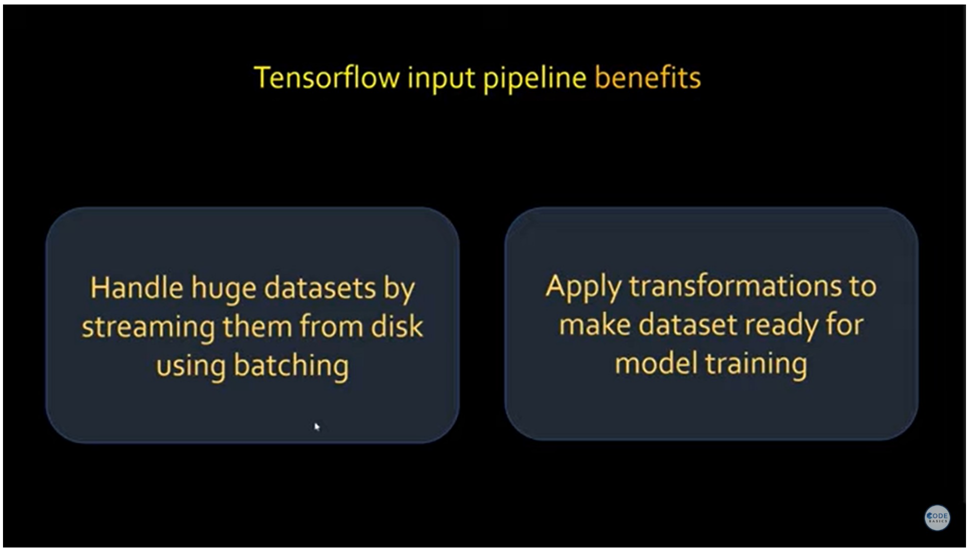


tf Dataset - Tensorflow Input Pipeline









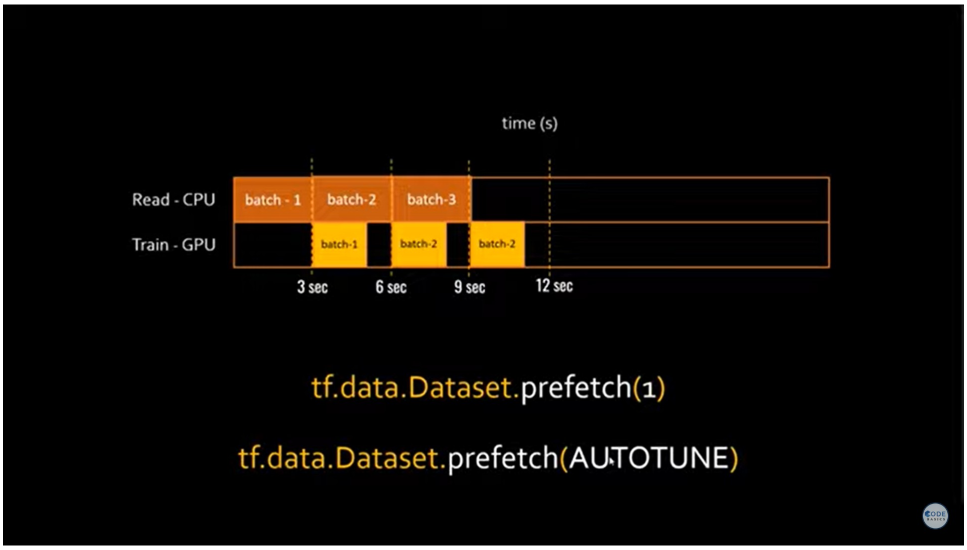
In [85]:
```
daily_sales_numbers = [21, 22, -108, 31, -1, 32, 34,31]

tf_dataset = tf.data.Dataset.from_tensor_slices(daily_sales_numbers)
tf_dataset
```

Out[85]:
```
<TensorSliceDataset element_spec=TensorSpec(shape=(), dtype=tf.int32, name=None)>
```

In [86]:
```
tf_dataset = tf_dataset.filter(lambda x: x>0).map(lambda y: y*72).shuffle(2).batch(

for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```

```
[1512 2232]
[2304 1584]
[2232 2448]
```

Prefetch & Cache


time (s)
Read - CPU
Train - GPU
batch - 1
batch-2
batch-3
batch-1
batch-2
batch-2
3 sec
6 sec
9 sec
12 sec
tf.data.Dataset.prefetch(1)
tf.data.Dataset.prefetch(AUTOTUNE)

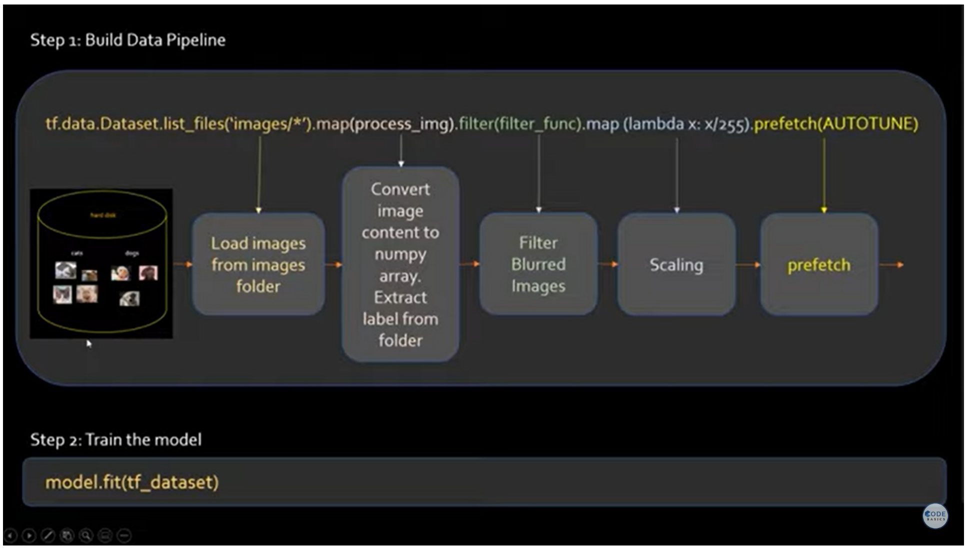
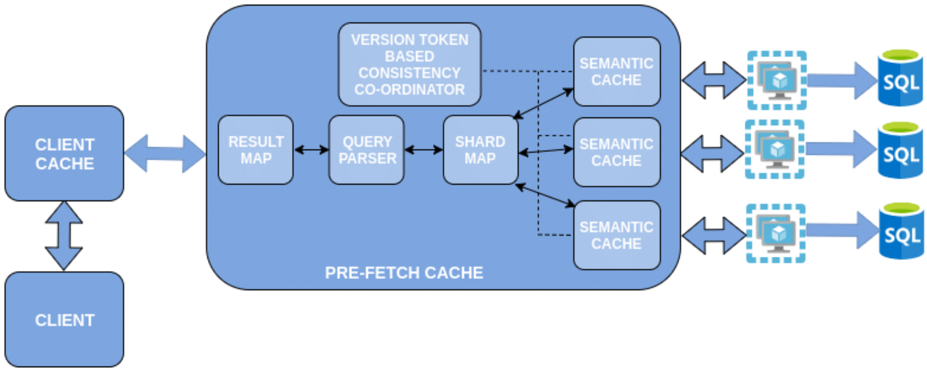
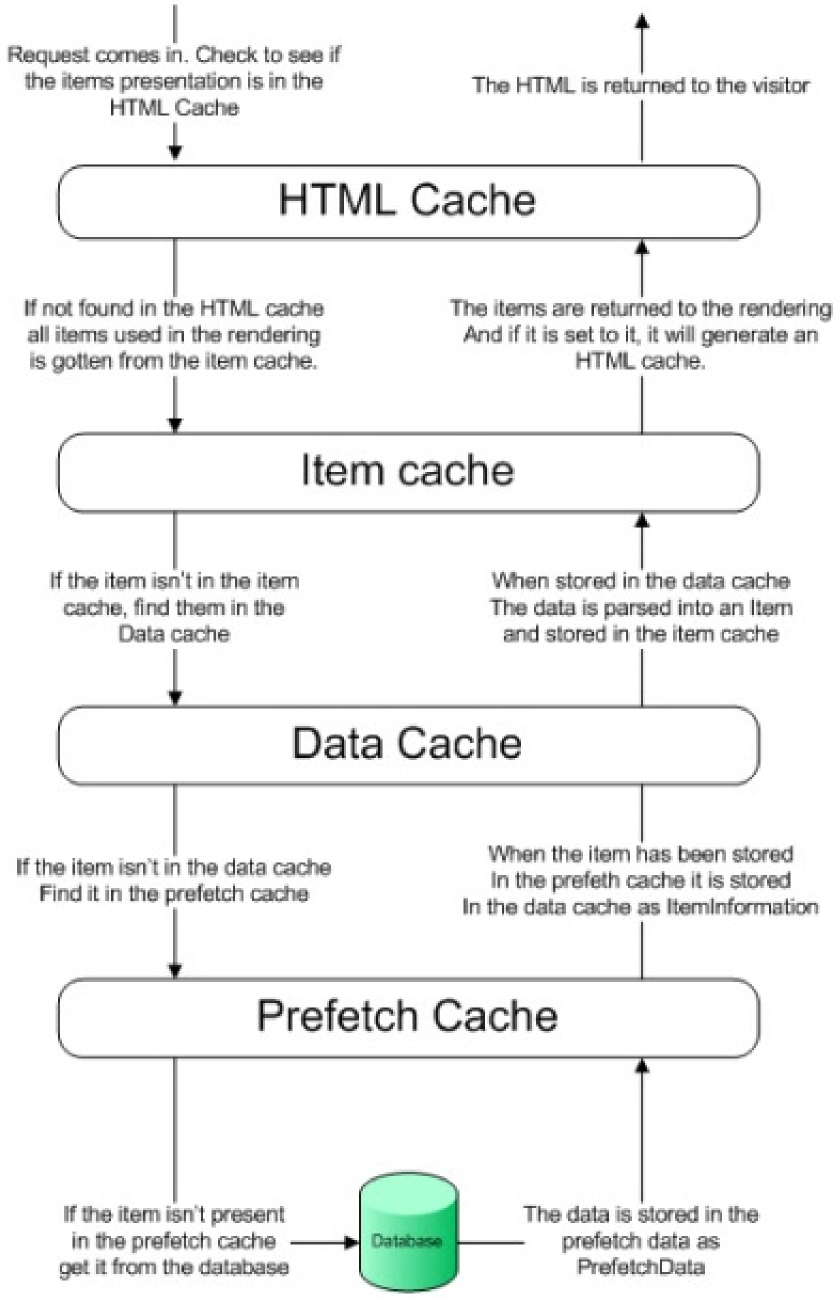

Step 1: Build Data Pipeline
tf.data.Dataset.list_files('images/*').map(process_img).filter(filter_func).map (lambda x: x/255).prefetch(AUTOTUNE)
Load images from images folder
Convert image content to numpy array. Extract label from folder
Filter Blurred Images
Scaling
prefetch
Step 2: Train the model
model.fit(tf_dataset)


Image credit: tensorflow official documentation
Open
Read
Map
Train
Epoch
time (s)





In [87]:
```
import time
import tensorflow as tf
```

In [88]:
```
class FileDataset(tf.data.Dataset):
    def read_file_in_batches(num_samples):
        # Opening the file
        time.sleep(0.03)
```

```
        for sample_idx in range(num_samples):
            # Reading data (line, record) from the file
            time.sleep(0.015)

            yield (sample_idx,)

    def __new__(cls, num_samples=3):
        return tf.data.Dataset.from_generator(
            cls.read_file_in_batches,
            output_signature = tf.TensorSpec(shape = (1,), dtype = tf.int64),
            args=(num_samples,)
        )

def benchmark(dataset, num_epochs=2):
    for epoch_num in range(num_epochs):
        for sample in dataset:
            # Performing a training step
            time.sleep(0.01)
```

In [89]:
```
%%timeit -n1 -r1
benchmark(FileDataset())
```

```
314 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

In [90]:
```
%%timeit
benchmark(FileDataset().prefetch(1))
```

```
284 ms ± 40.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

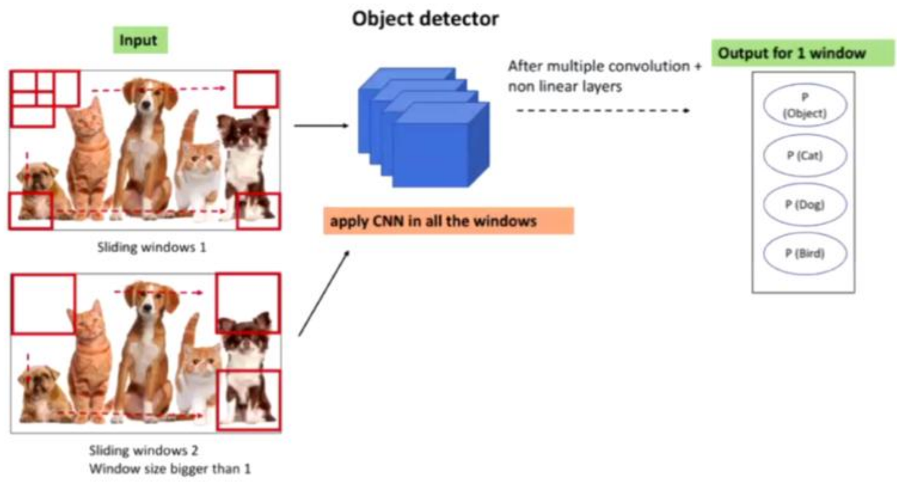
In [91]:
```
%%timeit
benchmark(FileDataset().prefetch(tf.data.AUTOTUNE))
```

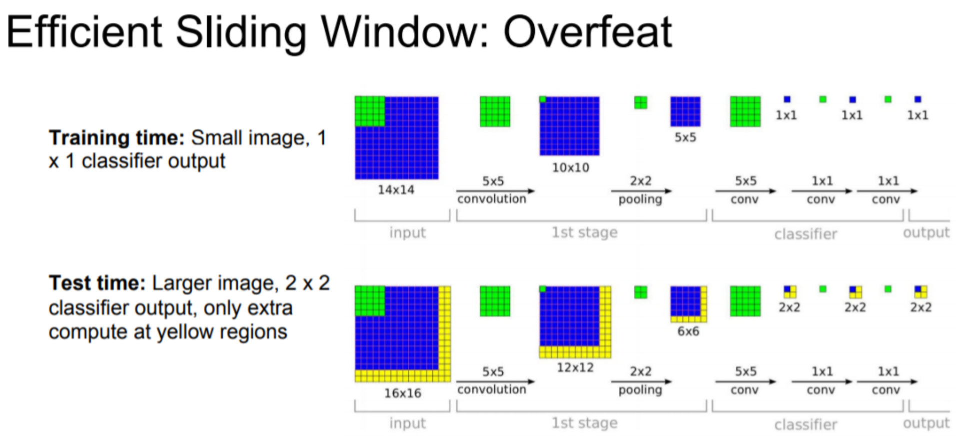
```
259 ms ± 31.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

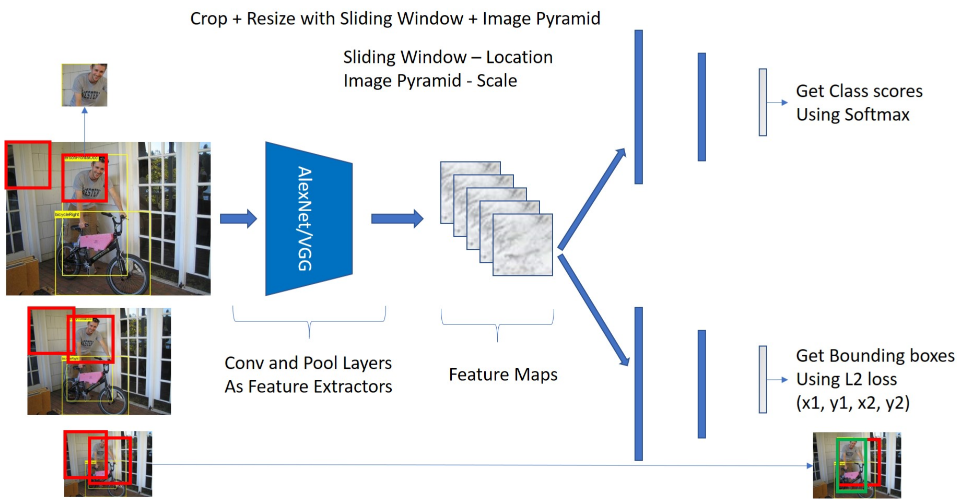
In [ ]:
```
dataset = tf.data.Dataset.range(5)
dataset = dataset.map(lambda x: x**2)
dataset = dataset.cache("mycache.txt")
# The first time reading through the data will generate the data using
# `range` and `map`.
list(dataset.as_numpy_iterator())
```

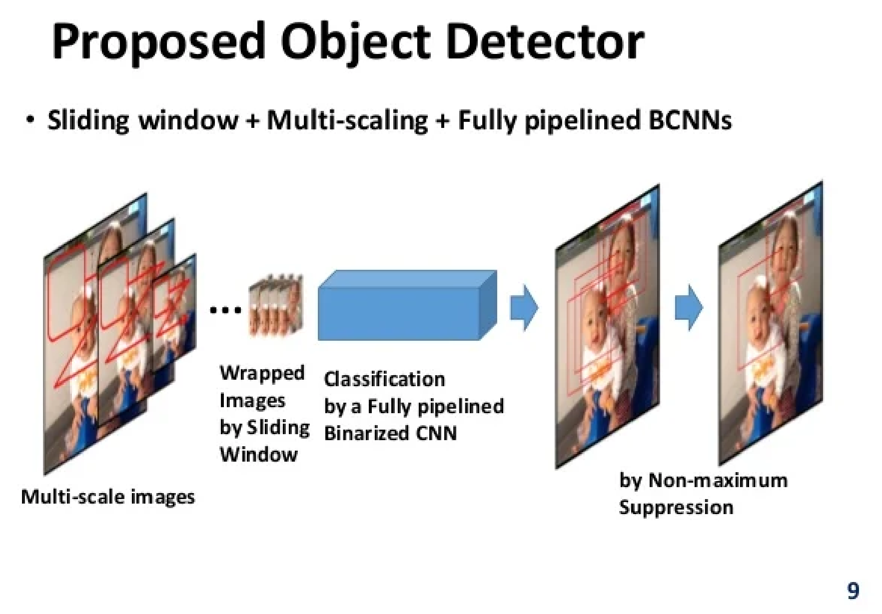
In [93]:
```
def mapped_function(s):
    # Do some hard pre-processing
    tf.py_function(lambda: time.sleep(0.03), [], ())
    return s
```

Sliding Window Object Detection

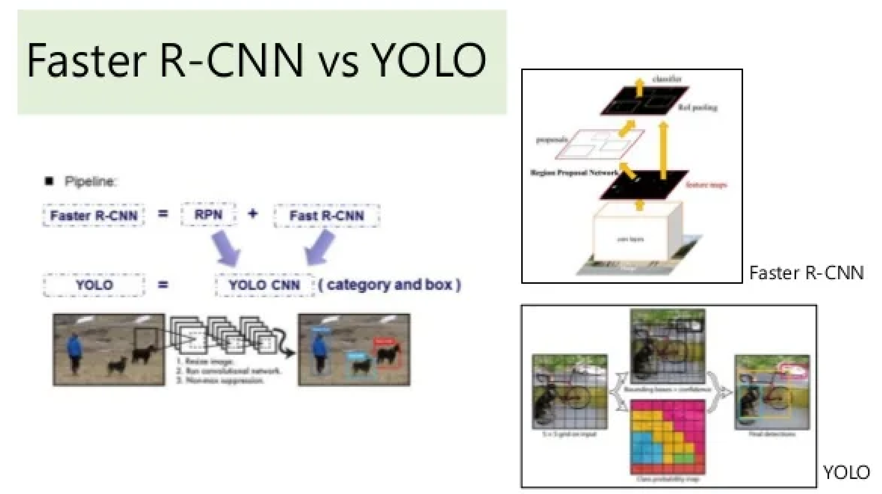

Object detector
Input
Sliding windows 1
Sliding windows 2
Window size bigger than 1
After multiple convolution + non linear layers
apply CNN in all the windows
Output for 1 window
P (Object)
P (Cat)
P (Dog)
P (Bird)

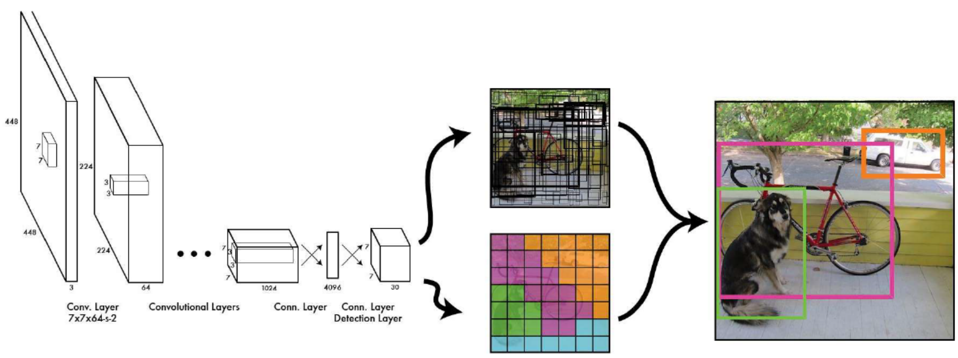

Efficient Sliding Window: Overfeat
Training time: Small image, 1 x 1 classifier output
Test time: Larger image, 2 x 2 classifier output, only extra compute at yellow regions
14x14
5x5 convolution
10x10
2x2 pooling
5x5
5x5 conv
1x1 conv
1x1 conv
1x1
input
1st stage
classifier
output
16x16
12x12
6x6
2x2

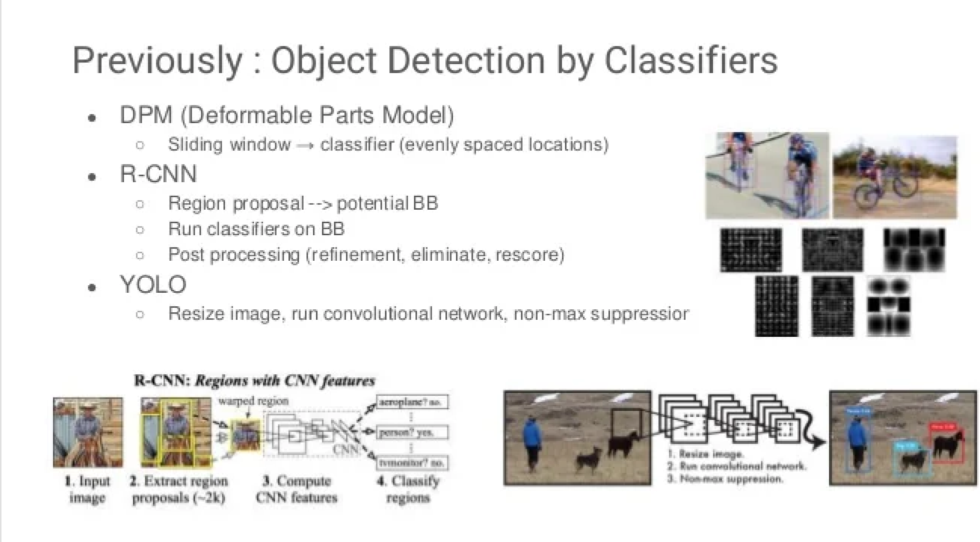

Crop + Resize with Sliding Window + Image Pyramid
Sliding Window – Location
Image Pyramid - Scale
AlexNet/VGG
Conv and Pool Layers
As Feature Extractors
Feature Maps
Get Class scores
Using Softmax
Get Bounding boxes
Using L2 loss
(x1, y1, x2, y2)

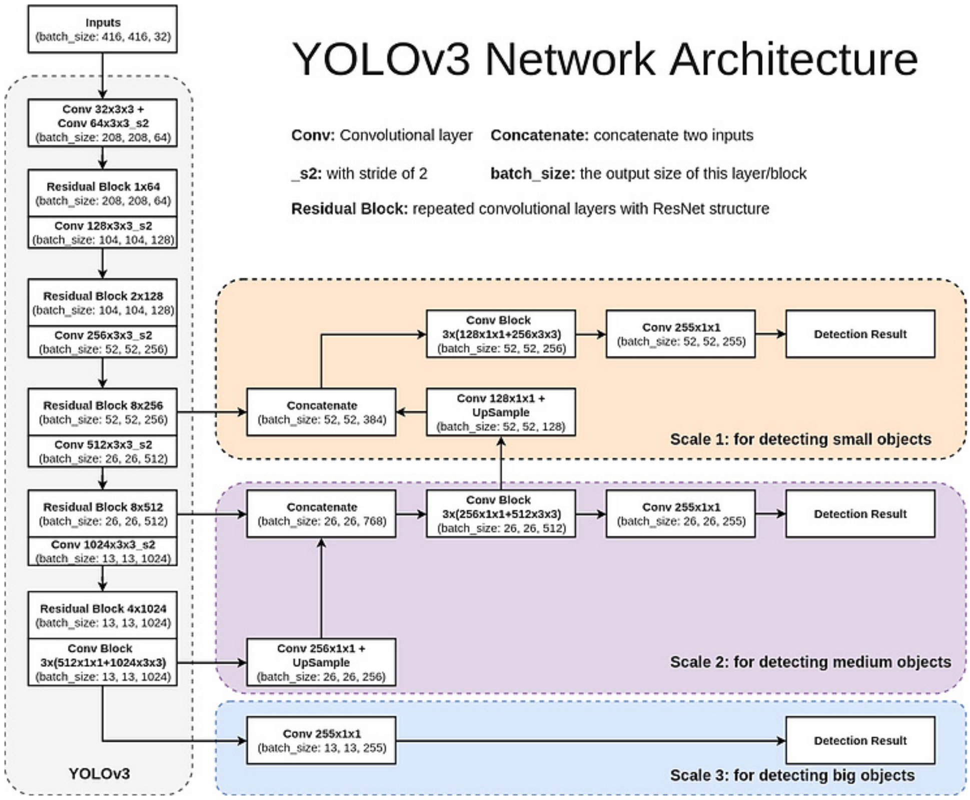


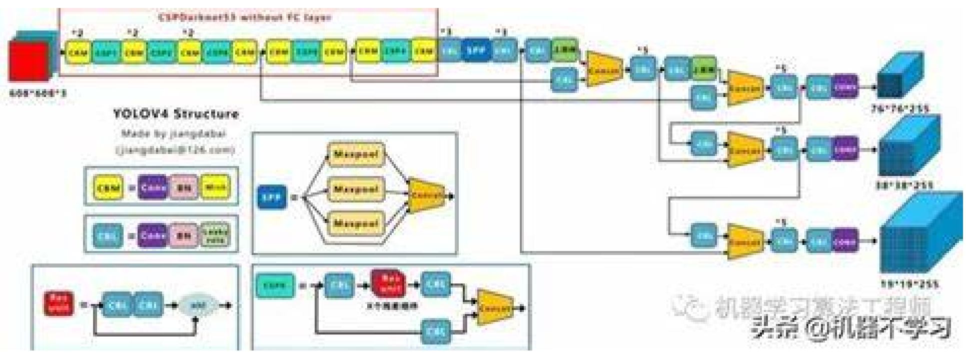
YOLO(You Only Look One)



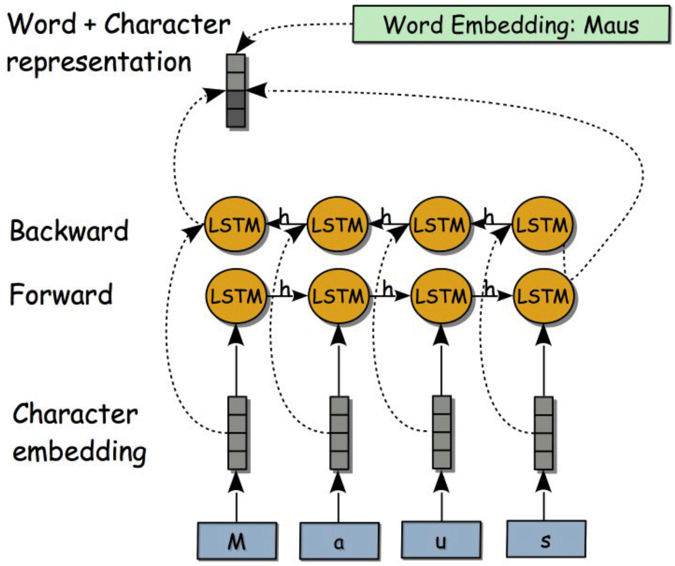




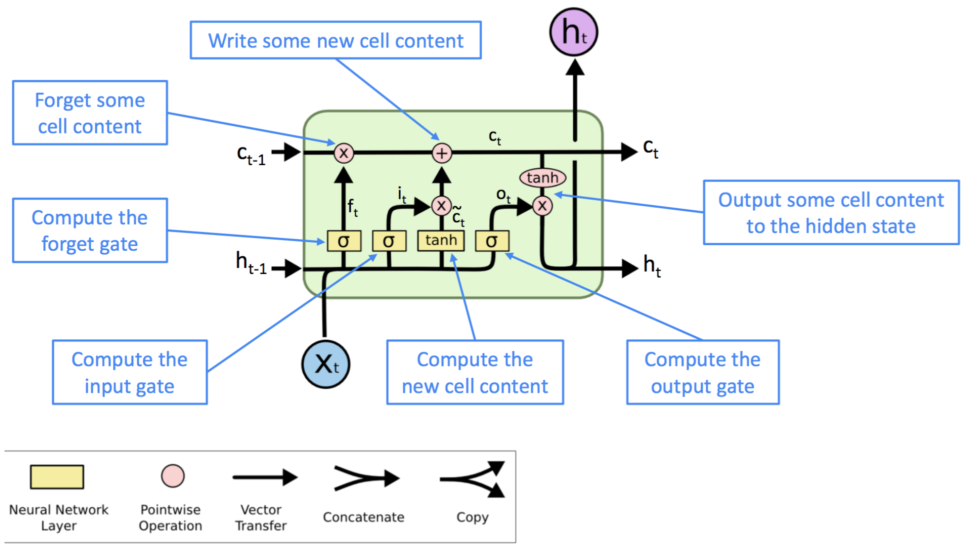


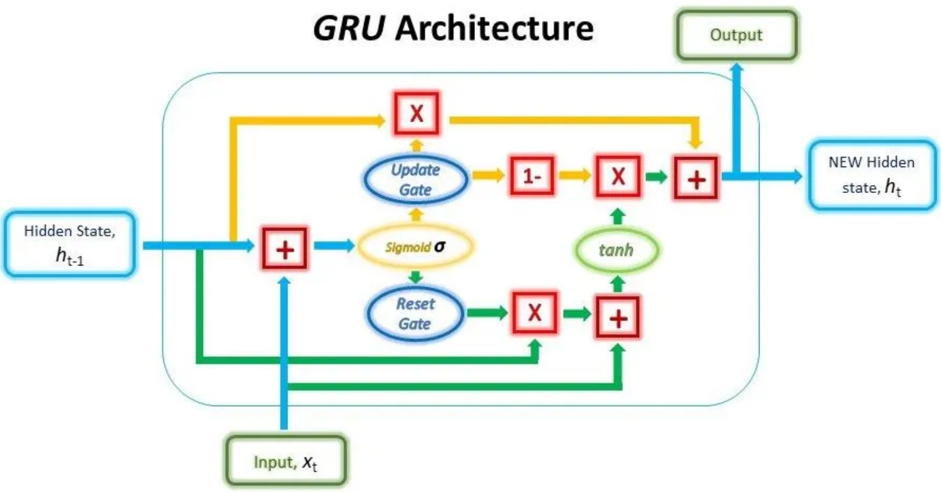


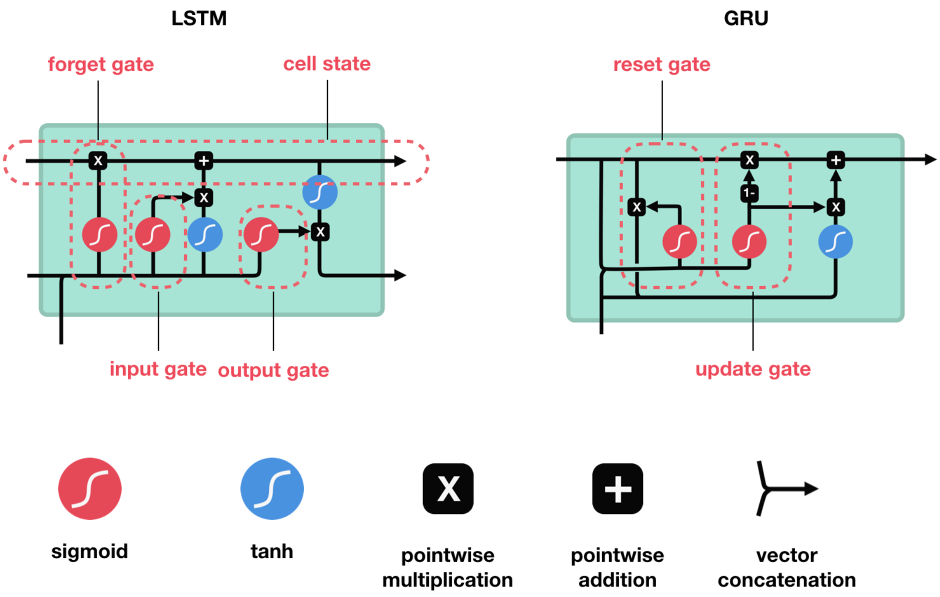
LTSM (Long Short Term Memory) and GRU (Gated Recurrent Units)



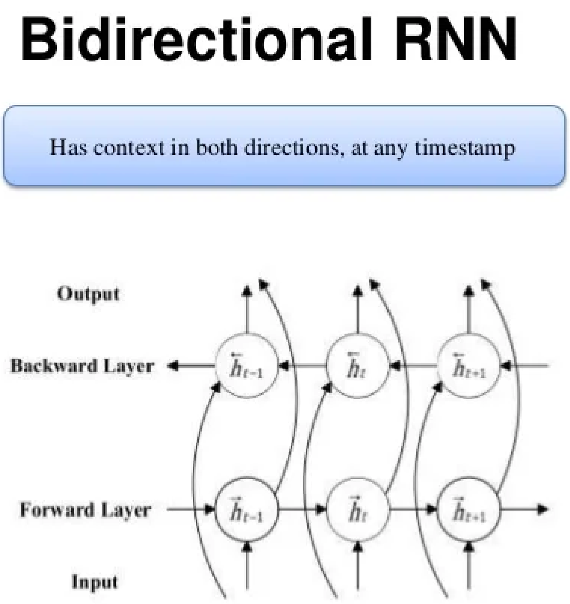
You can think of the LSTM equations visually like this:

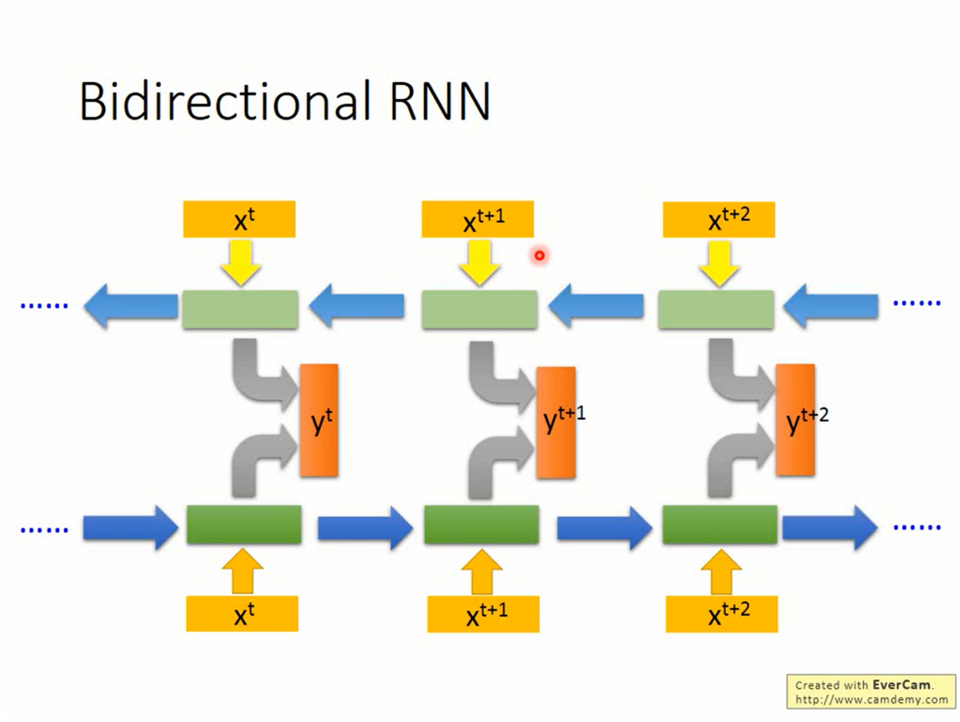


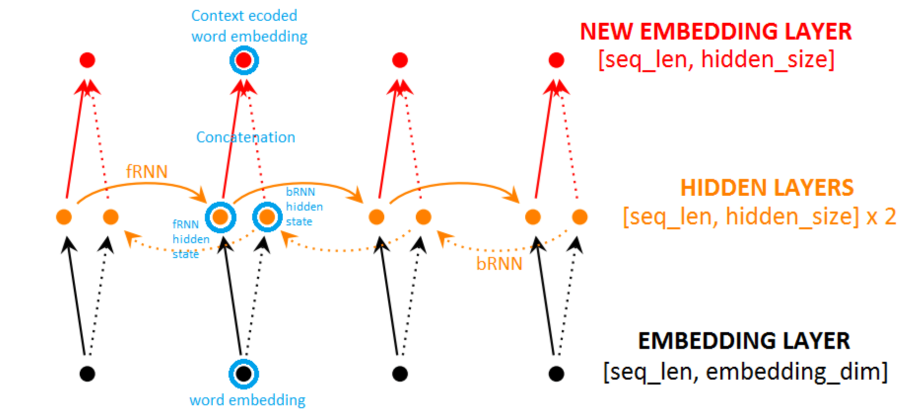




Bidirectional RNN

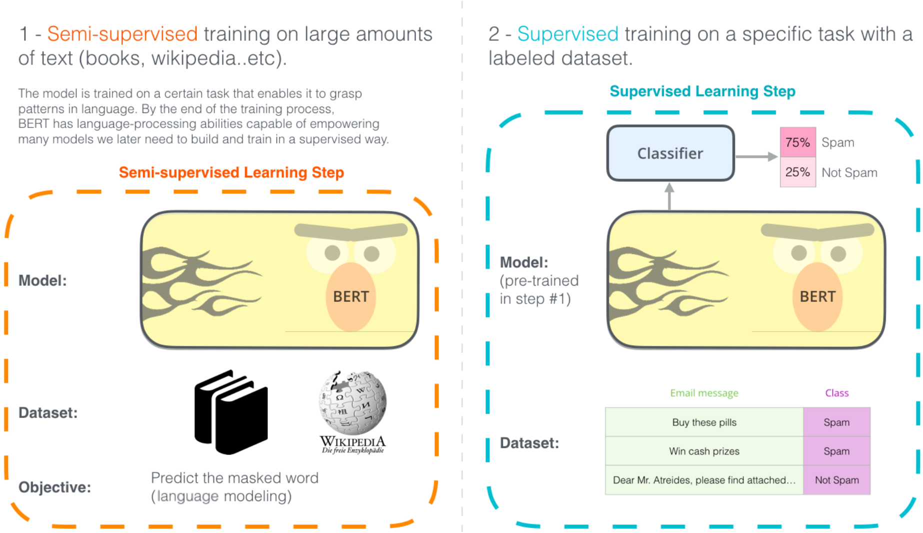






BERT (Bidirectional Encoder Representations From Transformers)

TF2.0 Saved Model (v4)



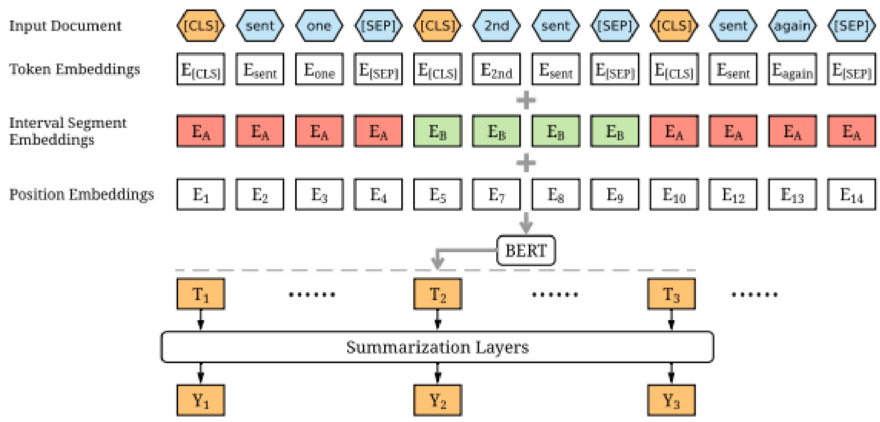
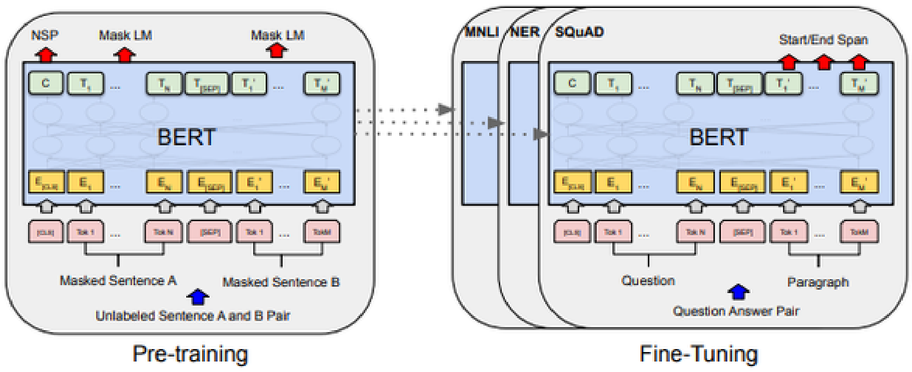
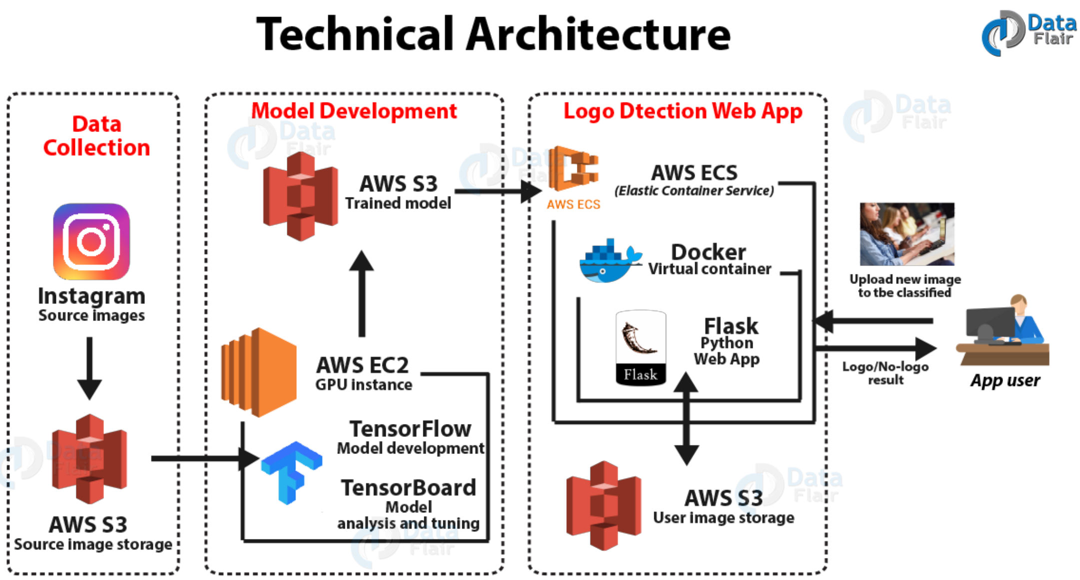
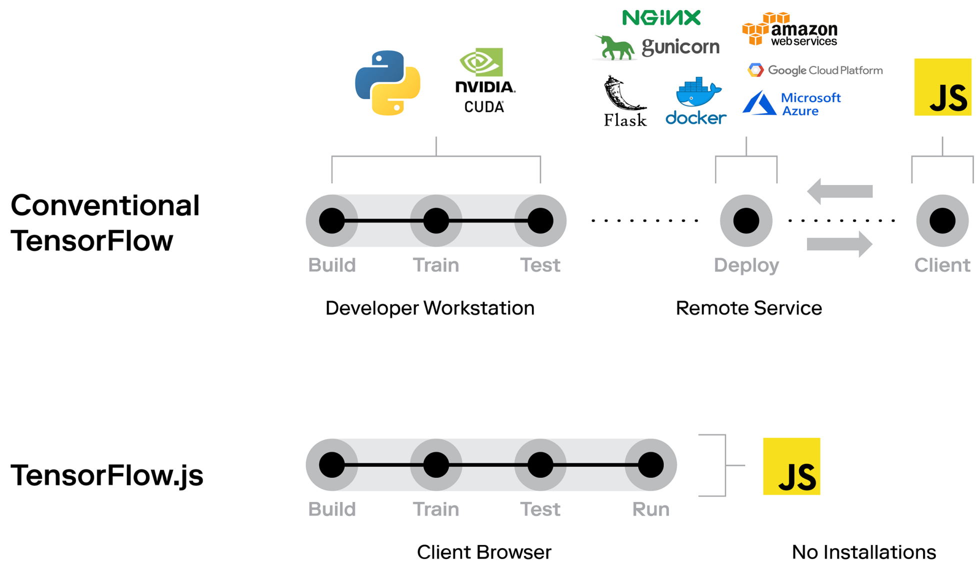


Figure 1: The overview architecture of the BERTSUM model.



Tensorflow Deploymeny




Conventional TensorFlow
CUDA
Flask
docker
Google Cloud Platform
Microsoft Azure
JS
Build
Train
Test
Deploy
Client
Developer Workstation
Remote Service
TensorFlow.js
JS
Build
Train
Test
Run
Client Browser
No Installations

-- Memo End --