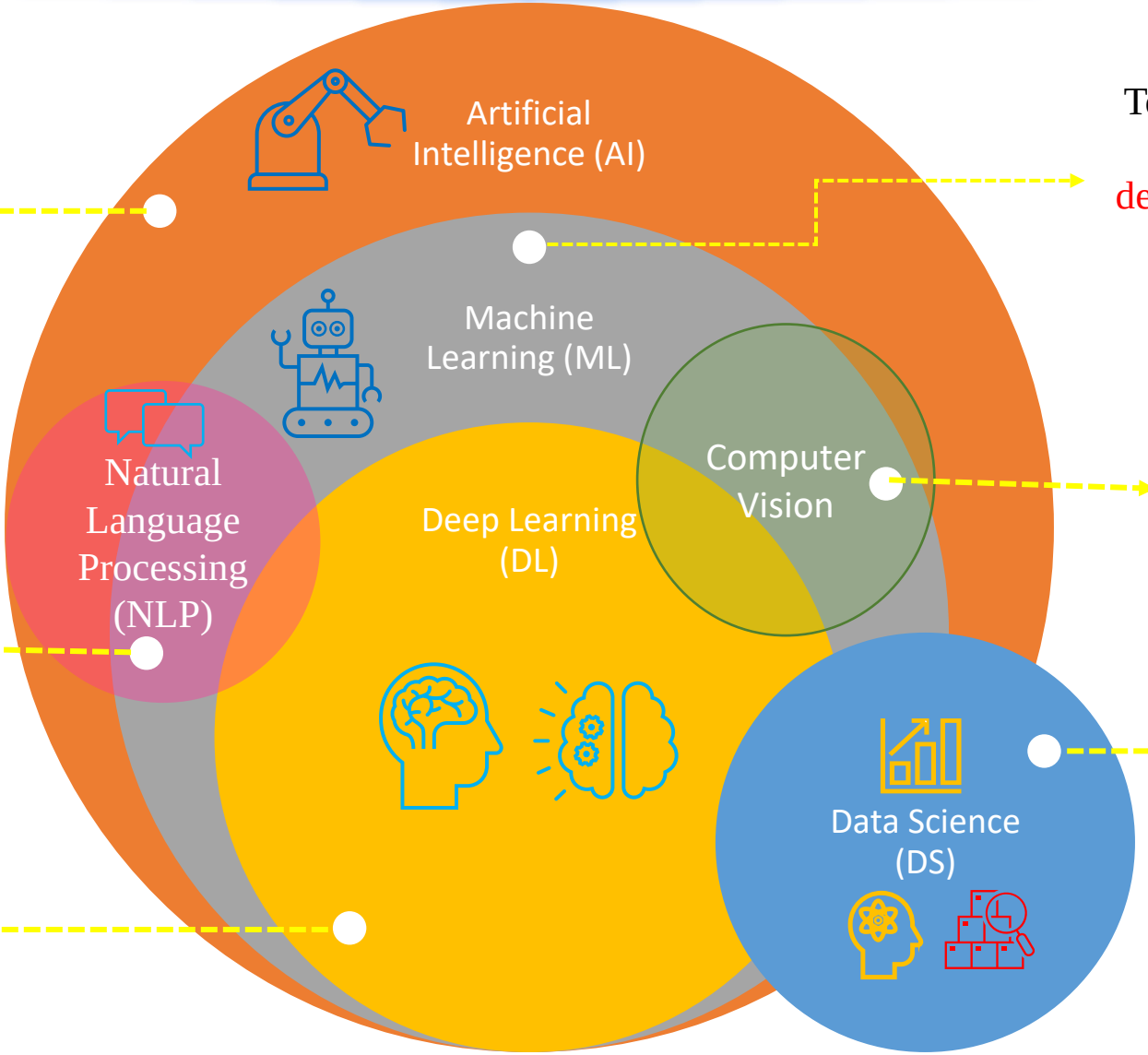
COMPUTER VISION
&
DEEP LEARNING

Presented by
M.Mohana
Research Scholar

AI VS ML VS DL VS DS – ARE ALL SAME ?

Creation of an Intelligence machine that works and reacts like human

To analyze, identify patterns from data, and make decisions given to machine is called ML



The concept which is used to listening & replying, analyzing, speaking, chatting capability given to the machine is called NL

Computer to understand digital images and videos

Data Science is not exactly a subset of ML but it uses ML and DL to gain insights from both structured and unstructured data

Human visualization & Recognition capability that can be given to the machine is called DL. NN learns from vast amounts of data.

What is Computer Vision?

- It focuses on developing techniques to make computers process, analyse and understand digital images, video, or other digital inputs.
- It allows computers to extract meaningful information from images and videos the same way humans do.
- The idea is to mimic how the human eye can capture light and color in the real world and extract information from images.

Example:

- Humans can immediately recognize a flower as they see it because they have had a million years of head start to help identify what a flower looks like, what kind is it, where it grows, and how to tell the different flowers apart.
- But computers do not have that same advantage; to computers, it may look like a massive array of numbers with no context here, but data.

Computer Vision	Machine Learning	Deep Learning
It helps systems understand and interpret the visual world in the way humans do.	It provides machines the ability to automatically learn and improve from their past experiences.	It is a subset of AI that seeks to mimic the functioning of the human brain based on artificial neural networks.
Focuses on developing techniques to make computers process, analyze and understand digital images, video or other digital inputs.	Focuses on building machines that can automatically learn from their experiences without being explicitly programmed.	The goal is to enable machines to gain some level of understanding and knowledge just the way human brain filters information.
Applications include image recognition, driverless car testing, medical diagnostics, livestock monitoring, movement analysis, etc.	Applications include speech recognition, traffic prediction, product recommendations, virtual assistants, self driving cars, email filtering, etc.	Applications include self driving cars, natural language processing, visual recognition, image and speech recognition, virtual assistants, etc.

What is OpenCV?

- It is a real-time optimized computer vision library.
- It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more.

Before starting Computer Vision

- Requirements for OpenCV
  - 32bit or 64 bit
  - Disk space
  - OS platform
  - Programming Language and version (C, C++, Java, Python, R)
- Choose a platform to work
  - Anaconda
  - Pycharm
  - Python IDE
  - Tabnine
  - Sublime Text 3
  - Visual Studio
  - MatLab
  - Google colab
  - Kaggle
  - Azure Notebooks
  - Google Cloud Platform
  - Amazon SageMaker
- How to create an environment
- How to install packages

Image processing:

- Read, write (Save with specified format), display
- Color Spaces (RGB, HSV, CMYK, etc.,)
- Image enhancing
- Gray scaling of images
- Image Resizing
- Image transformation
- Analyze image using histogram
- Histogram Equalization
- Convert image one color space to another one.
- Visualization of the image in a different color
- Image Inpainting
- Filter color
- Segmentation using Thresholding
- Image Pyramid

Video processing:

- Play a video
- Create a video using images
- Extract image (frame) from video

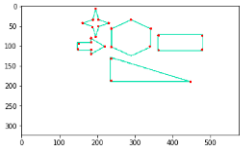
https://docs.opencv.org/3.4/d9/df8/tutorial_root.html

Feature Detection and Description:

- Line detection using Houghline method
- Circle Detection
- Detect corner of an image
- Corner Detection with Shi-Tomasi method
- Corner detection with Harris Corner Detection
- Find Circles and Ellipses in an Image
- Document field detection
- Smile detection

Drawing a Function:

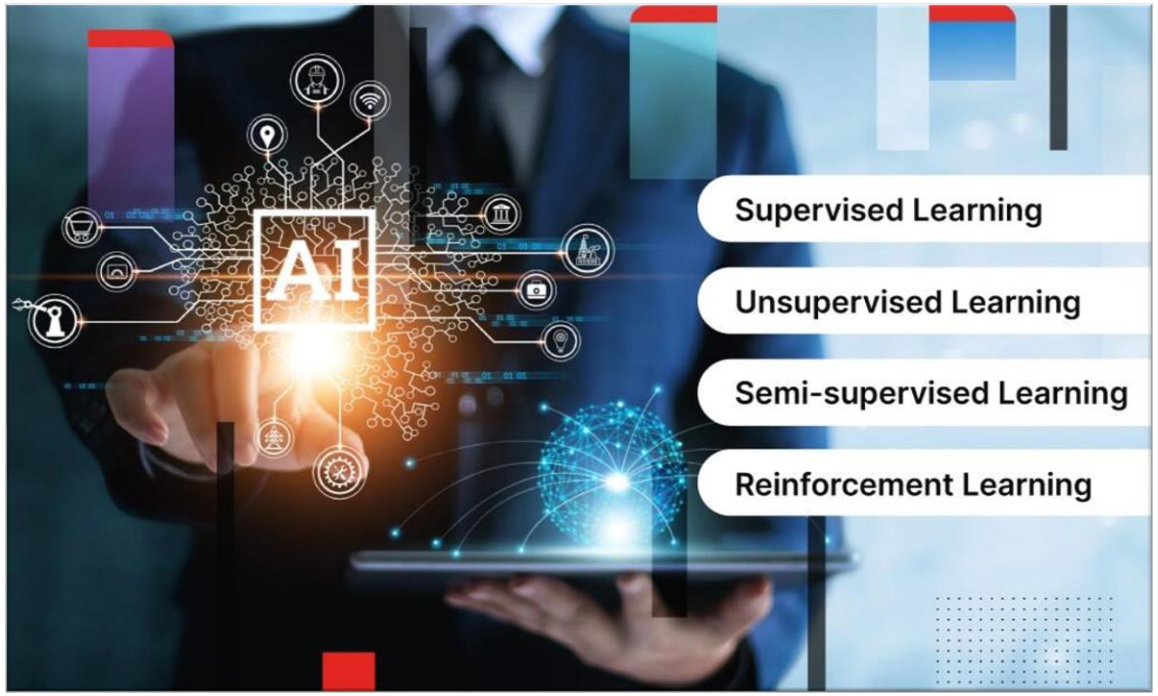
- Draw a line, ellipse, circle, rectangle
- Draw arrow segment
- Draw a text string
- Find and Draw Contours
- Draw a triangle with centroid



Application:

- White and block dot detection
- Face detection
- Count number of face
- Detect and Recognize car License Plate from real-time video
- Cartooning an image

Type of ML Learning

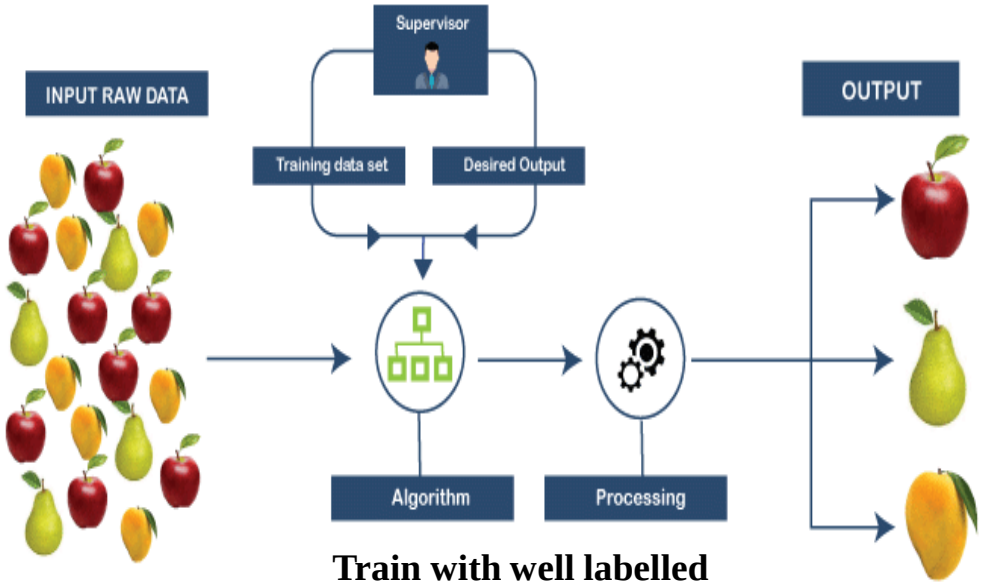


SUPERVISED LEARNING

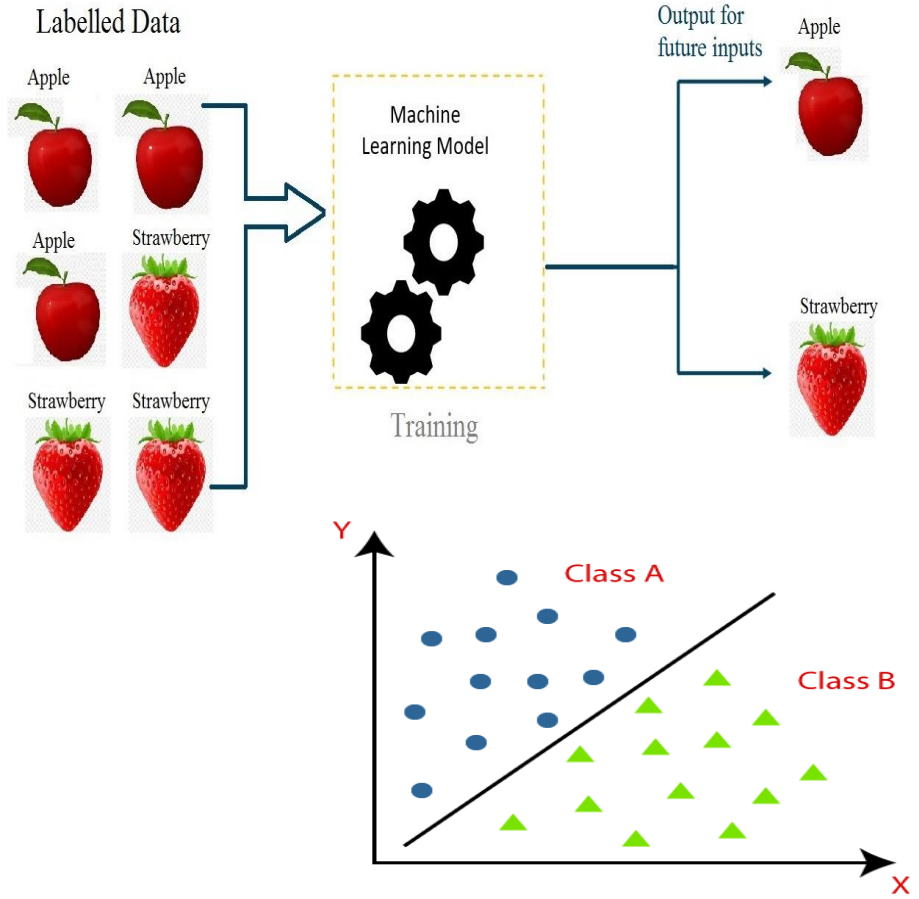
UNSUPERVISED LEARNING

REINFORCEMENT LEARNING

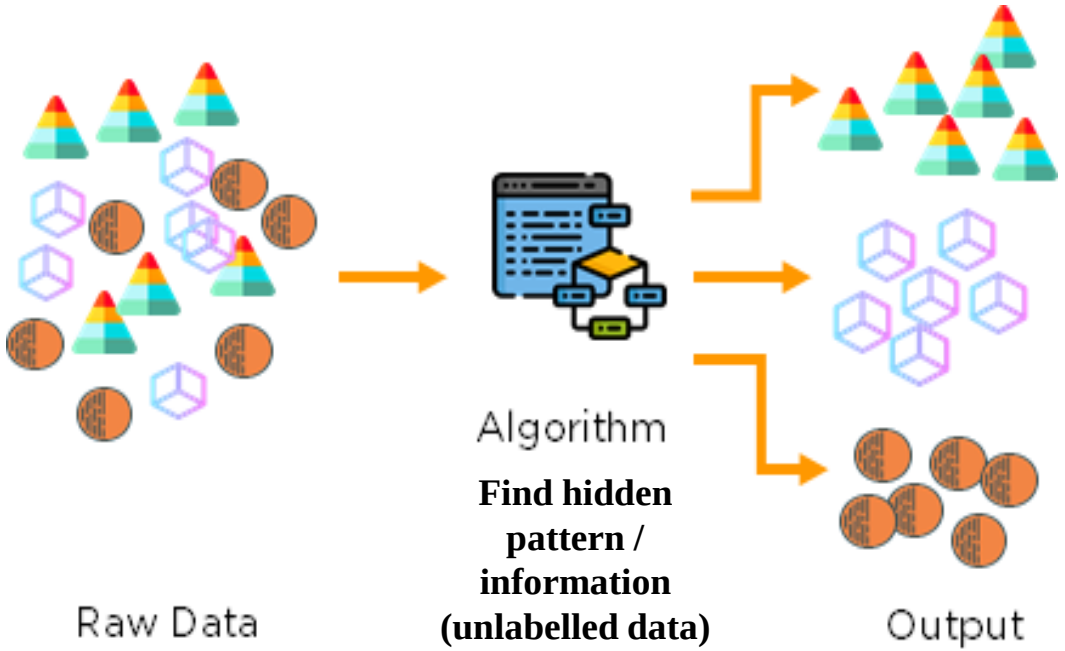
Supervised Learning

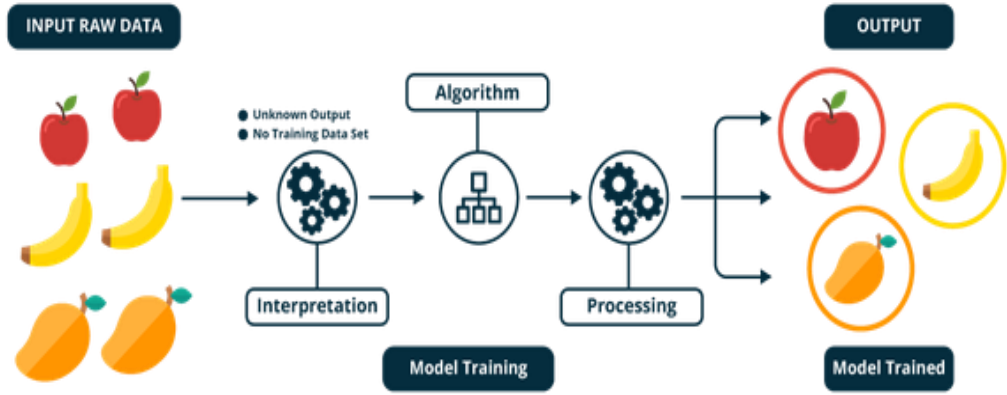


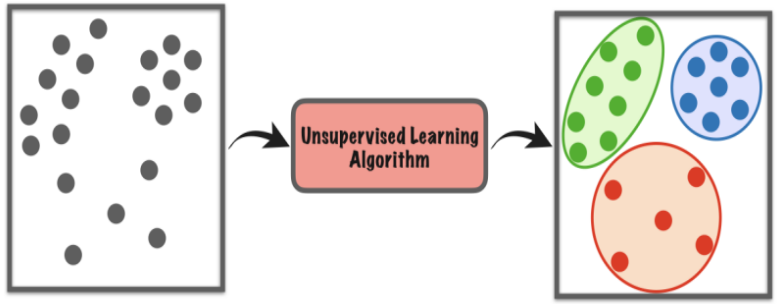
Train with well labelled data



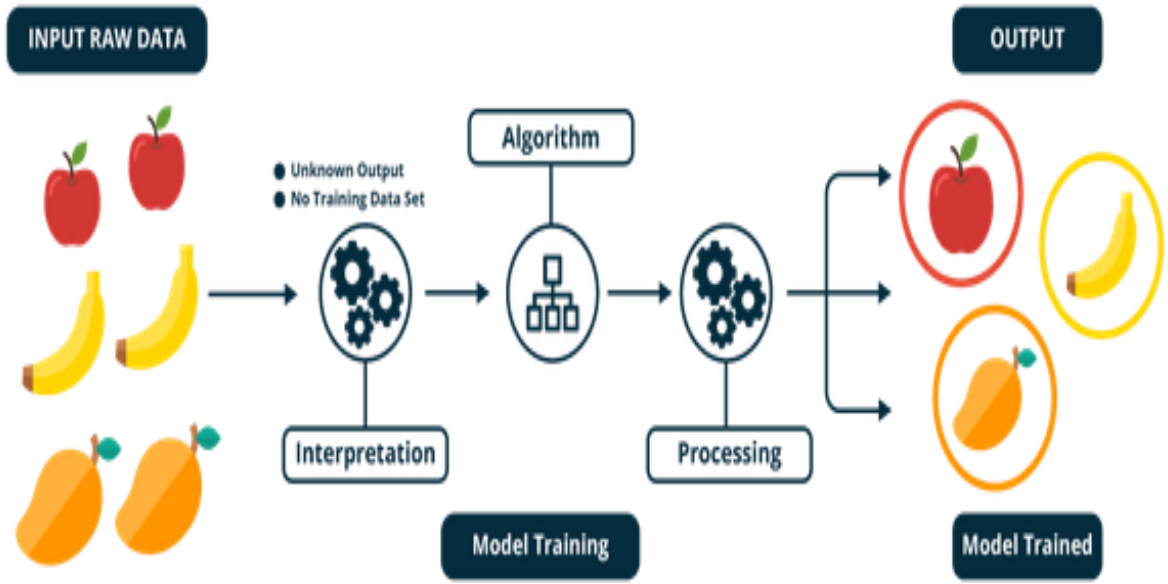
Unsupervised Learning







Semi-supervised learning



Learns from the labelled data to the unlabelled data and draw conclusions from the set as a whole

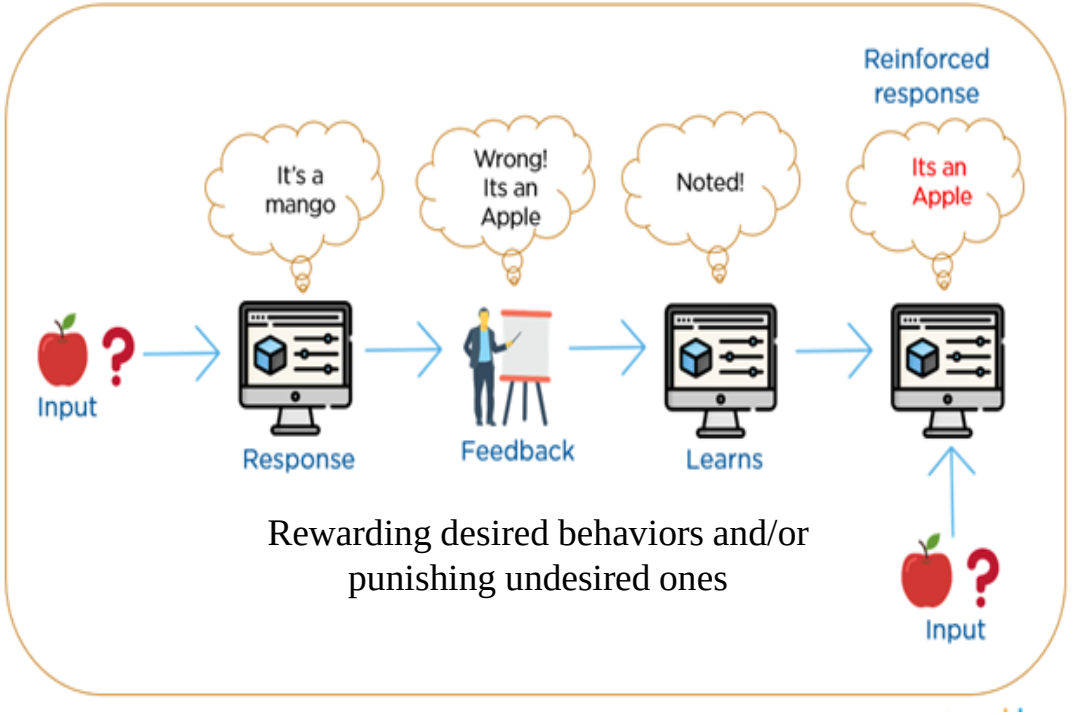
Supervised Learning

data	label
	Dog
	Bird
	Airplane
	Deer
	Cat
	Truck
	Ship

Semi-Supervised Learning

data	label
	Dog
	Bird
	No label
	No label
	No label
	No label
	No label

Reinforcement Learning



Rewarding desired behaviors and/or punishing undesired ones

Setup:
Mouse tries to maximize rewards by learning through trial and error.

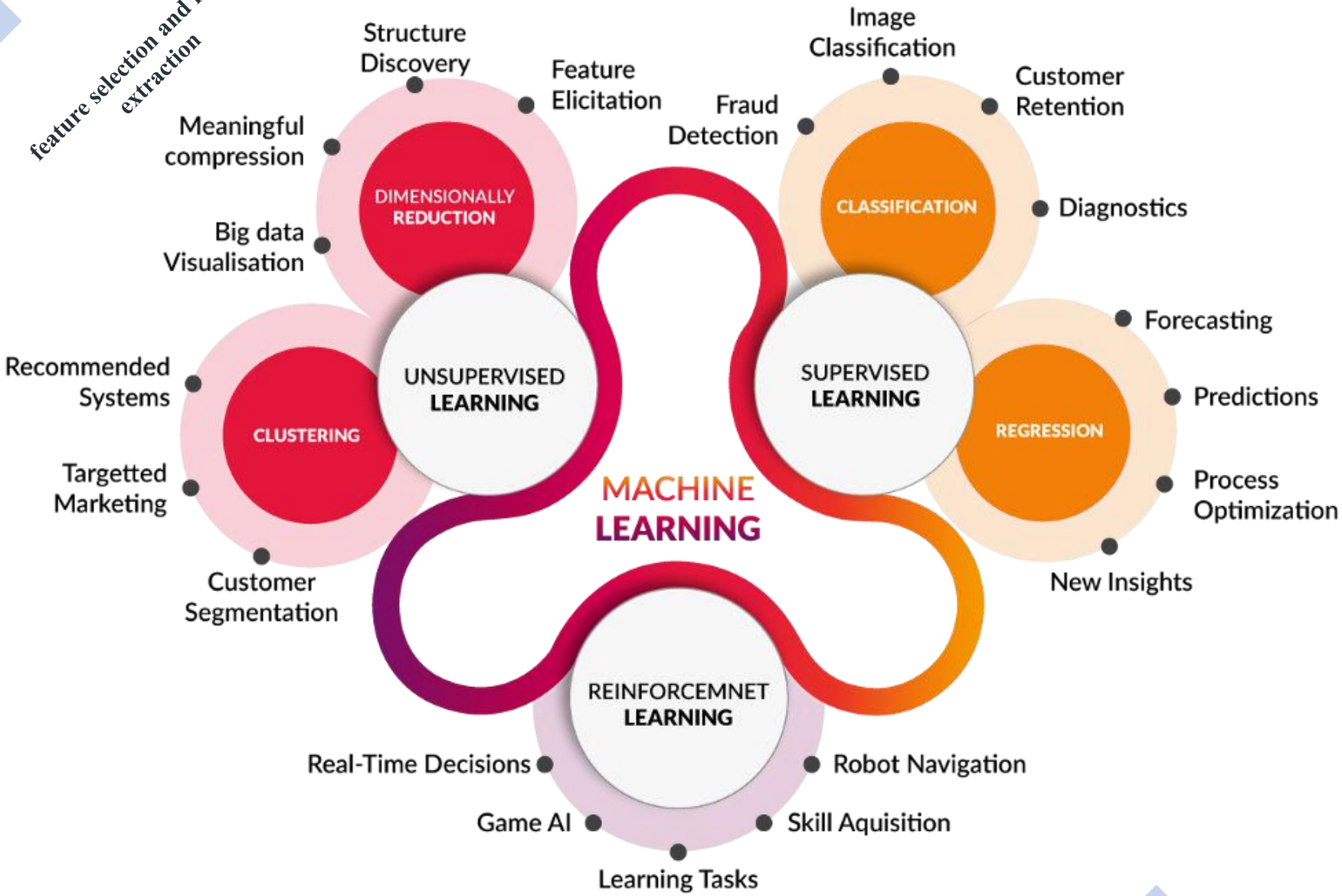
Point system

+1000

+50

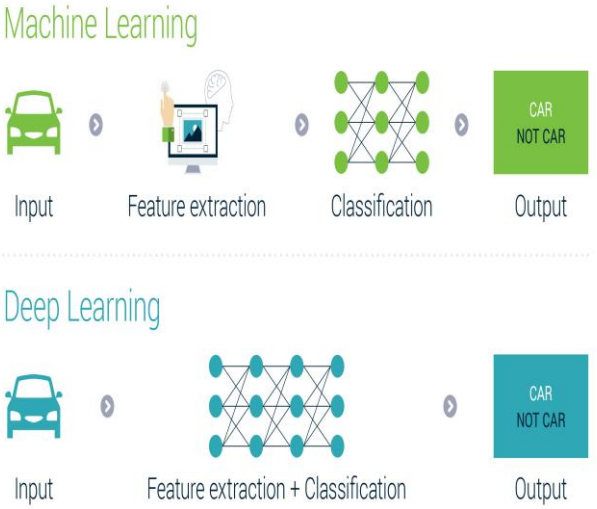
-100

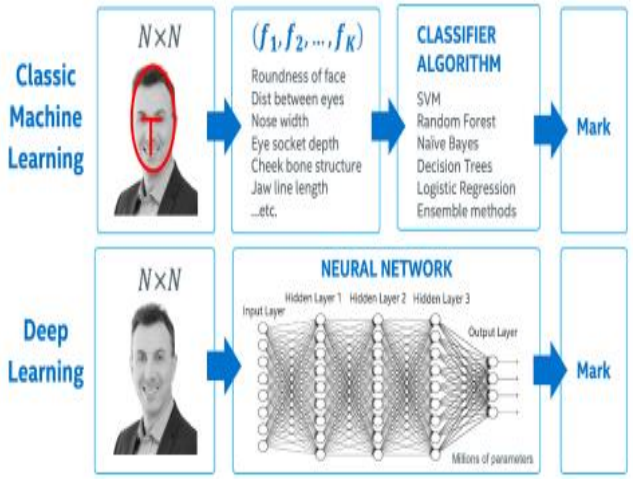
feature selection and feature extraction

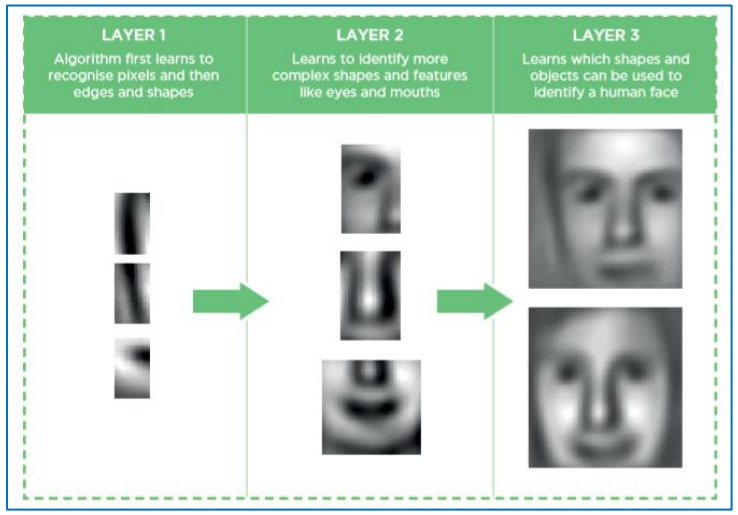


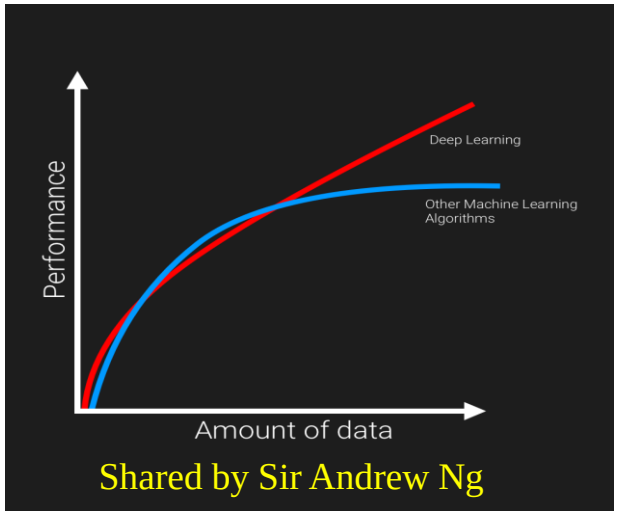
WHY THE NEED FOR DEEP LEARNING?

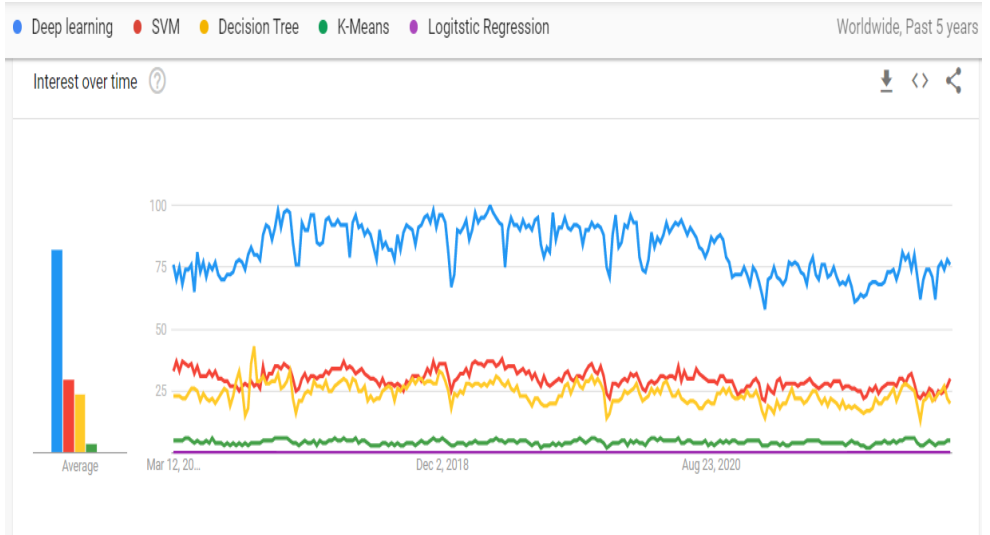
- The true field of Deep Learning began in 2012 bey before 2012, most of the experts believed that Neural Network was useless.
- 2013 most of the people are aware of smartphones and start using smartphones
- Data is the main specific reason for the Deep Learning
- Each day approx. 2.5 quintillion bytes of data generated





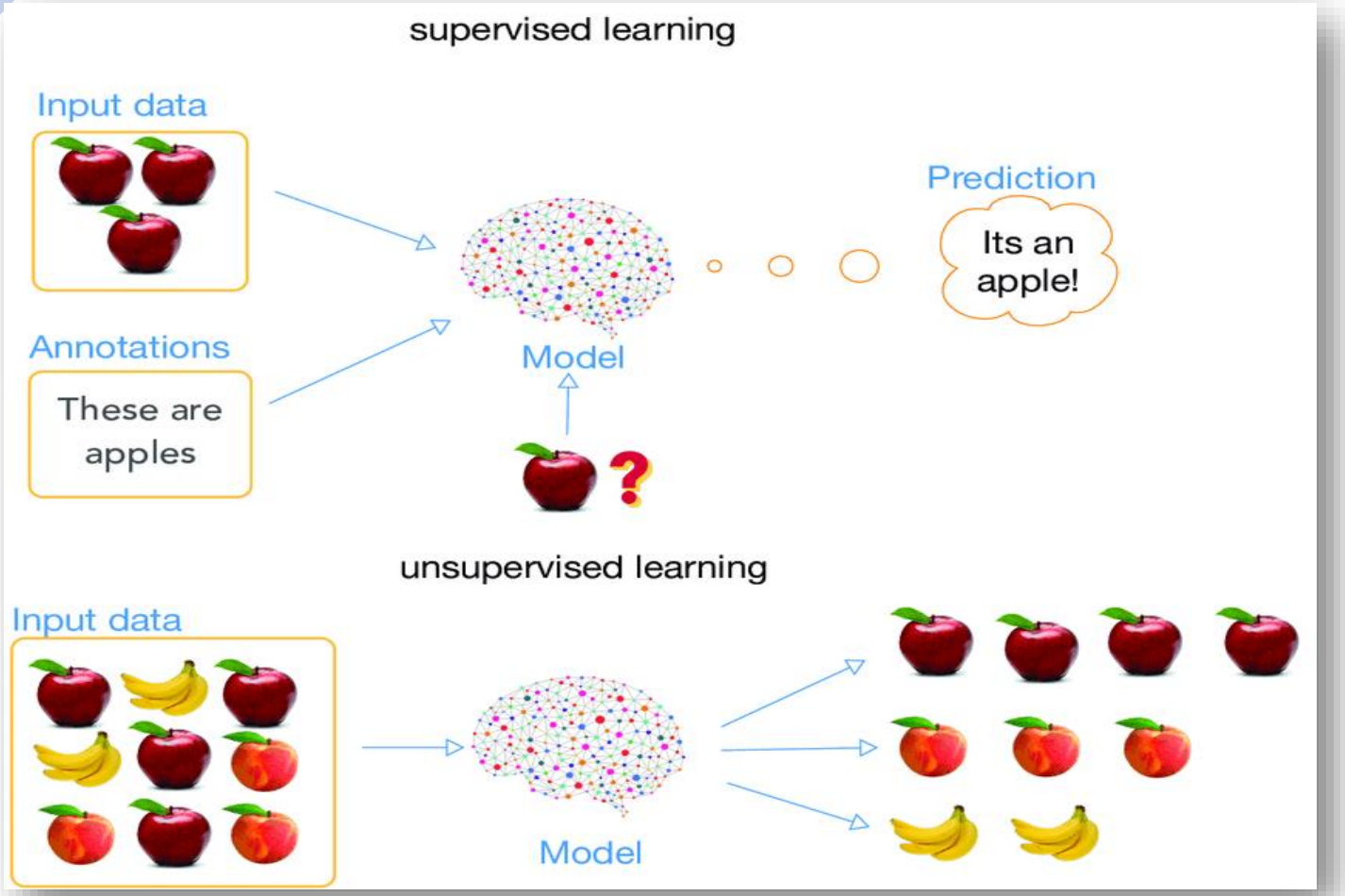


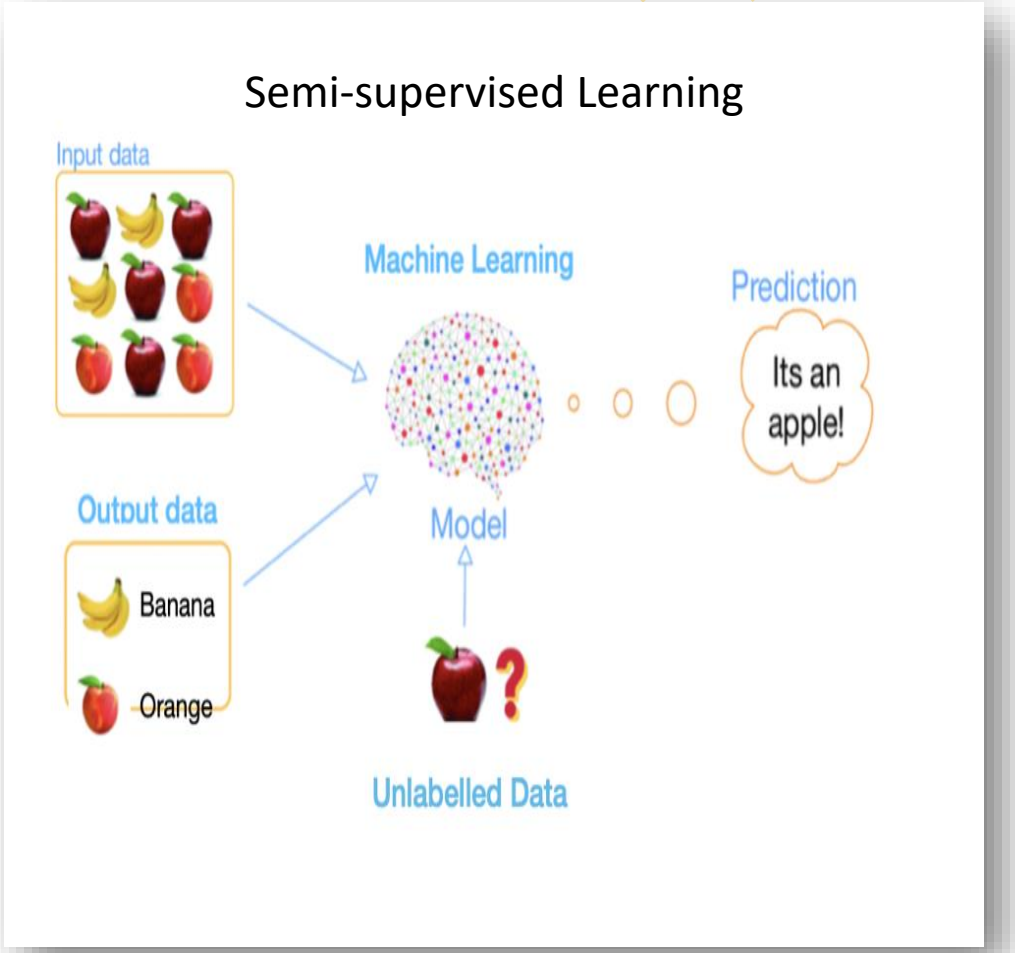




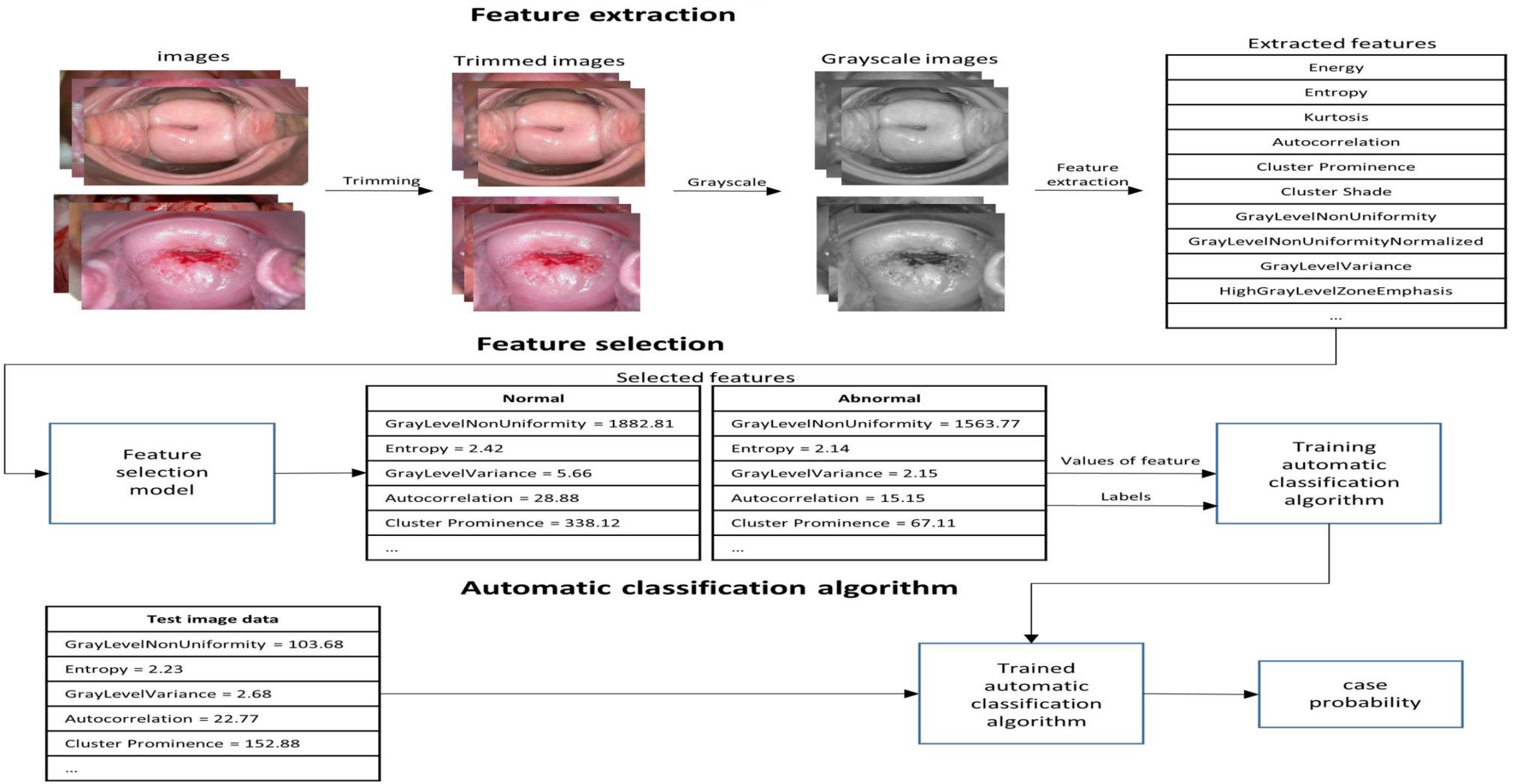
https://trends.google.com/trends/explore?q=deep%20learning&geo=GB

Deep Learning

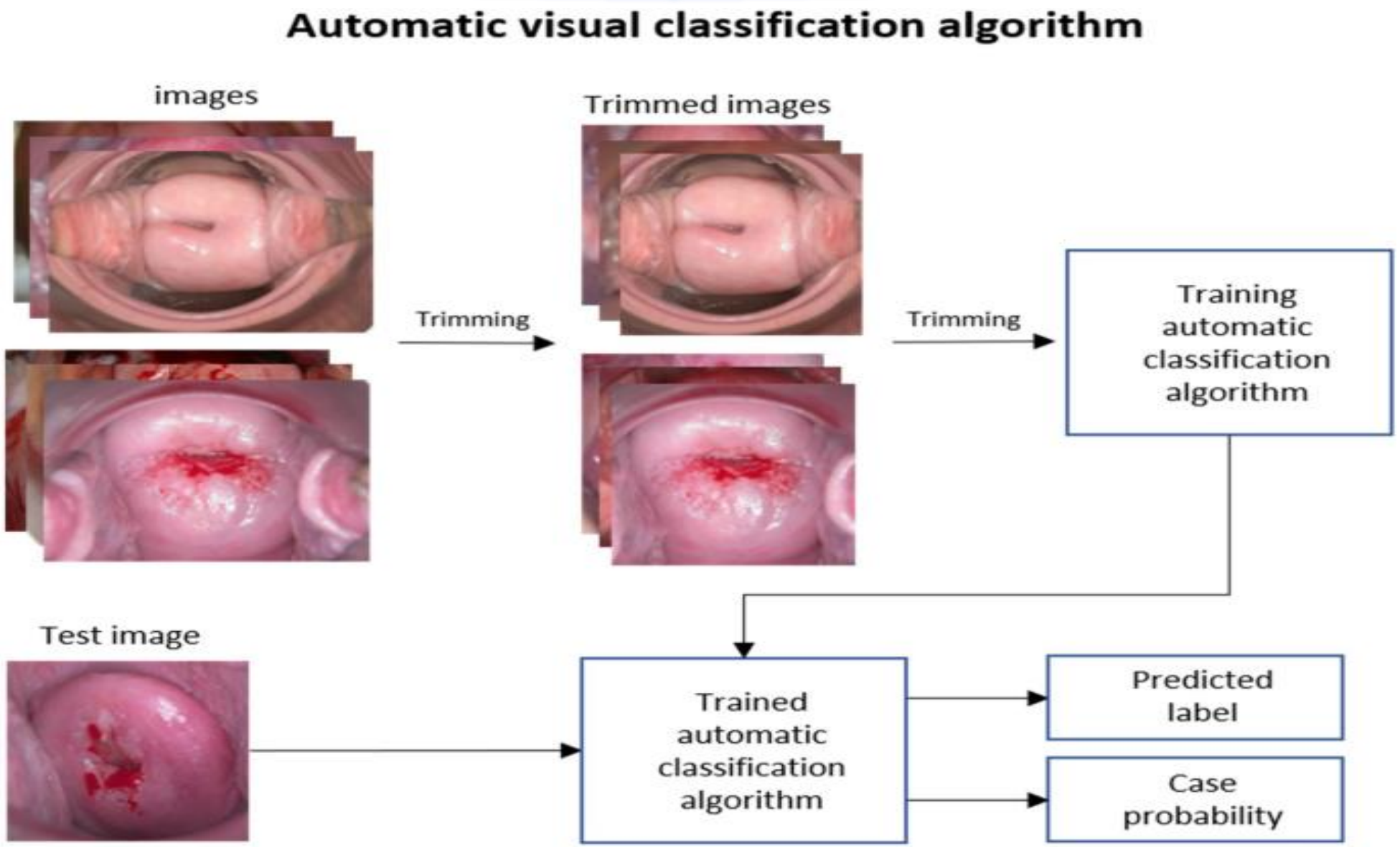




MACHINE LEARNING PROCESS



DEEP LEARNING PROCESS



DEEP LEARNING VS MACHINE LEARNING

Factors	Deep Learning	Machine Learning
Data Requirement	Requires large data (millions of data)	Can train on lesser data (Thousands of data)
Accuracy	Provides high accuracy	Gives lesser accuracy
Training Time	Takes longer to train (A few hours or week)	Takes less time to train (A few seconds or hours)
Hardware Dependency	Requires GPU to train properly	Train on GPU
Hyperparameter Tuning	Can be turned in various different ways	Limited tuning capabilities
Testing time	Very less	High depending on data size and algorithm
Feature selection	Automatic feature extraction, No human intervention for that	No automatic, Need human intervention to select features
Uses	Simple application	Complex application like autonomous
Data interpretation	Difficult	Some ML algorithm easy to interpret
Number of algorithm	Few	Many

HOW TO START DEEP LEARNING?

Setting up your system

- Basic system command and Git
- CPU vs GPU vs TPU

Key ML Concepts

- Type of Learning, Evaluation Metrics, Validation Techniques, gradient descent
- Linear Models,
- Overfitting and Underfitting,
- bias-variance

Programming Language

- Basic Python (datatypes, data structure, loops)
- Pandas, NumPy, Scikit-learn, Matplotlib
- Tensor flow, Keras, PyTorch

Caffe

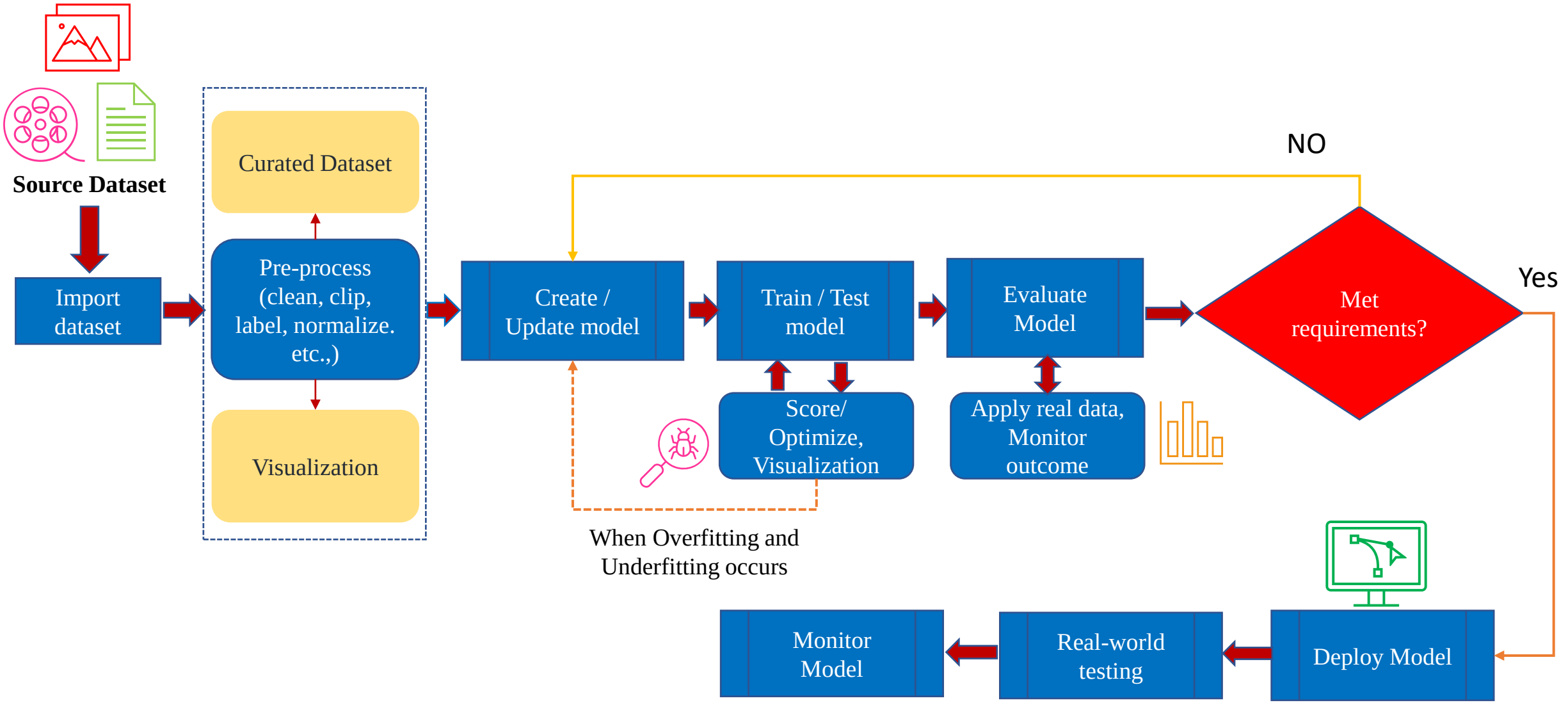
Linear Algebra and Calculus

- Scalar, Vectors, and Matrices
- Derivatives, Partial Derivatives and Chain Rule

Statistics and Probability

- Descriptive Statistics
- Probability, Random variables, the probability distribution

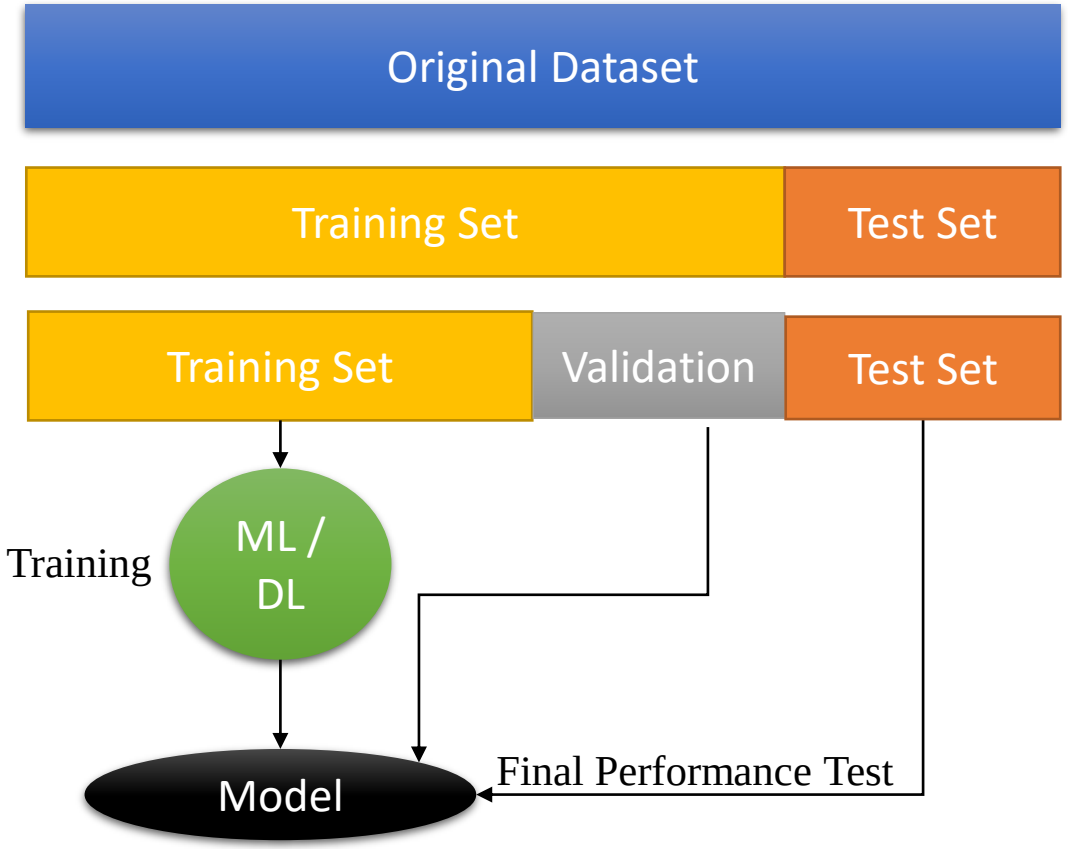
END-TO-END DL PROCESS



HOW/WHERE TO GET A DATASET?

A dataset is a collection of data in which data is arranged in some order. A dataset can contain any data from a series of an array to a database table.

Need of Dataset



Popular sources for Deep Learning datasets

- Kaggle
- UCI machine learning repository
- Datasets via AWS
- Google's Dataset Search Engine
- Microsoft Datasets
- Awesome Public Dataset Collection
- Government Dataset
  - Indian Government datasets
  - US Government Datasets
  - Northern Ireland Public Sector Datasets
  - European Union Open Data portal
- Computer Vision Datasets
- Scikit-learn Datasets
- Research papers

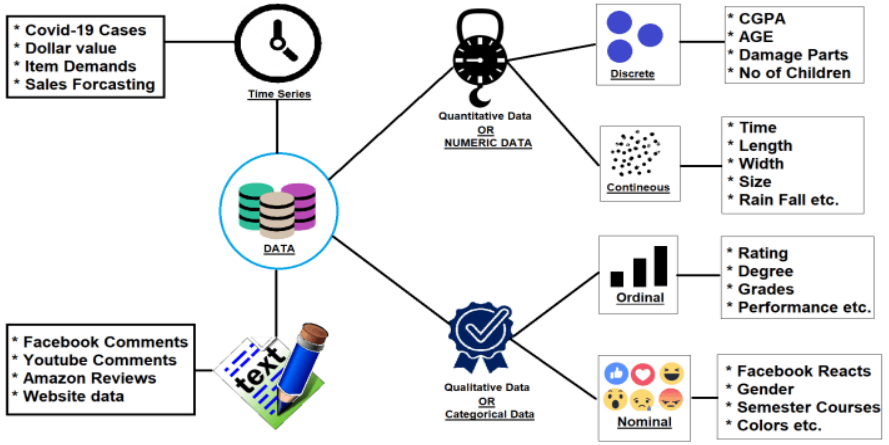
TYPE OF DATASETS

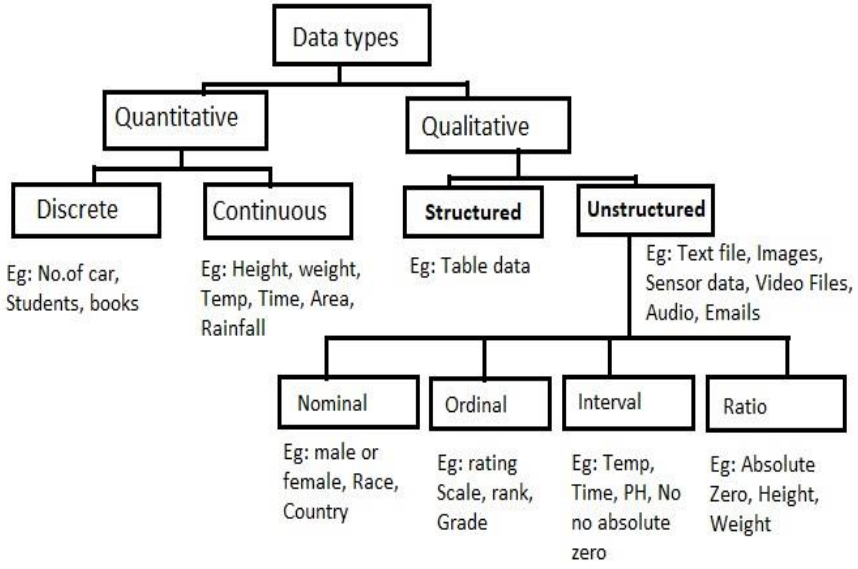
Assume: Patients medical history

- Test report (Numerical data – BP, Sugar, Heart rate, etc.,)
- Radiology report (Image +Text)
- Patient's data (Time series data)

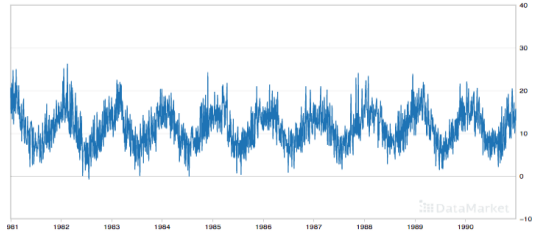
Findings:

- What are the % chances that the patient will survive and how many days he will be in the hospital. Which algorithm you will use and why?
- What is the mortality rate i.e. whether a person will die in the ICU in the following week or not. Which algorithm you will choose and why?
- Define some extra features useful for your problem that is missing in the question. In case this is an AI-powered doctor, how will it communicate the current condition to the patient's family?





	A	B	C	D
1	First Name	Last Name	Age	Salary
2	Jon	Smith	36	26500
3	Helen	Mirren	22	21000
4	David	Cameron	29	39000
5	Brad	Pitt	52	45000
6	Anna	Starolsky	41	22500
7	Peter	Piper	20	31500
8	David	Duck	19	15700
9	Julie	Walters	33	19000

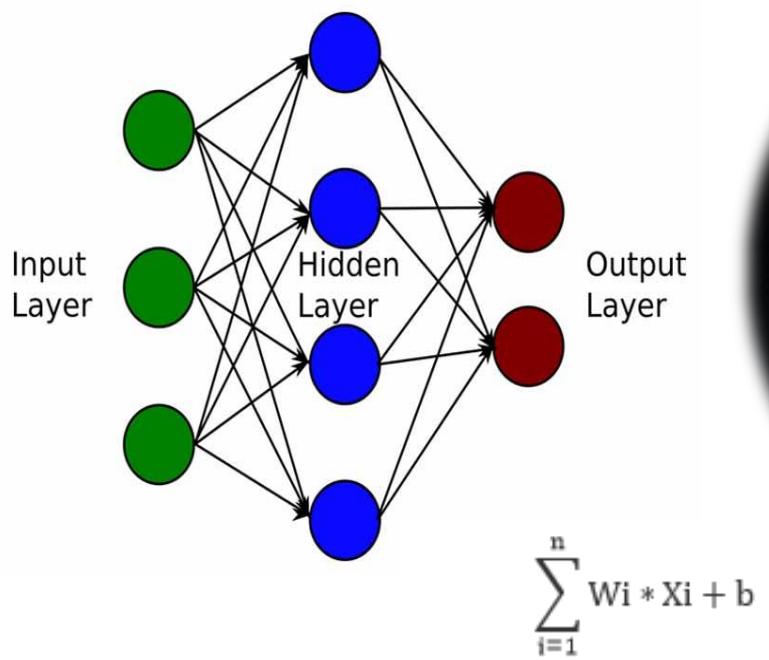


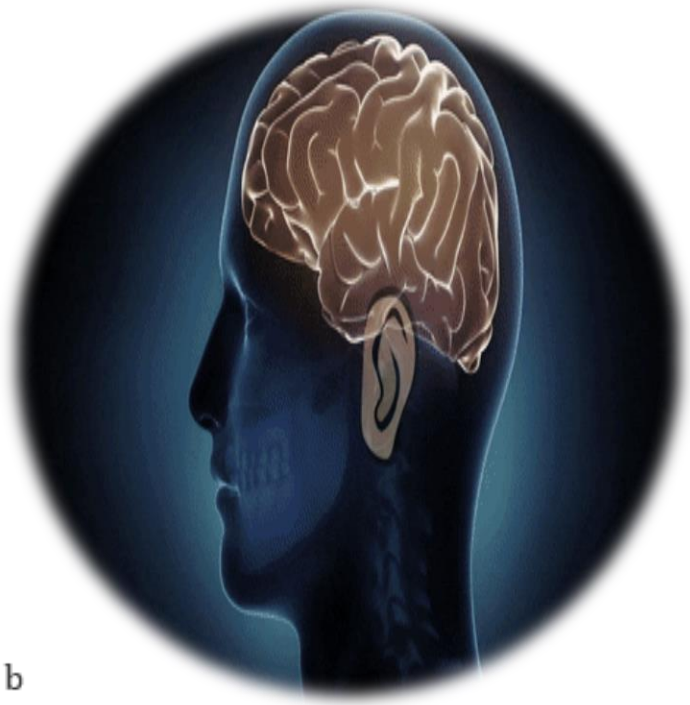
ARTIFICIAL NEURAL NETWORK (ANN)

- Inspired by the working of a human brain
- Learn from past experience.

Components:

- Input layer
  - It accepts input
- Hidden layer
  - Perform calculation to find features and patterns.
- Output layer
  - It gives output

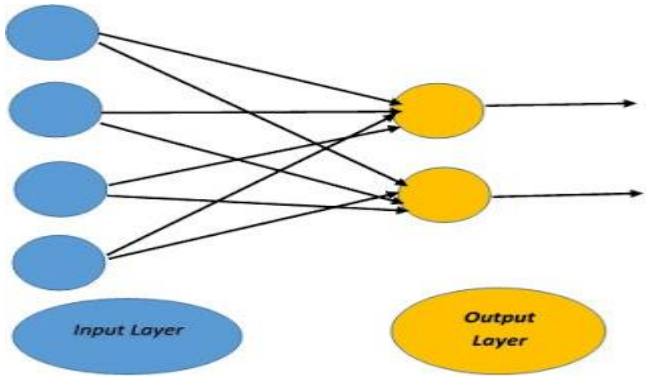


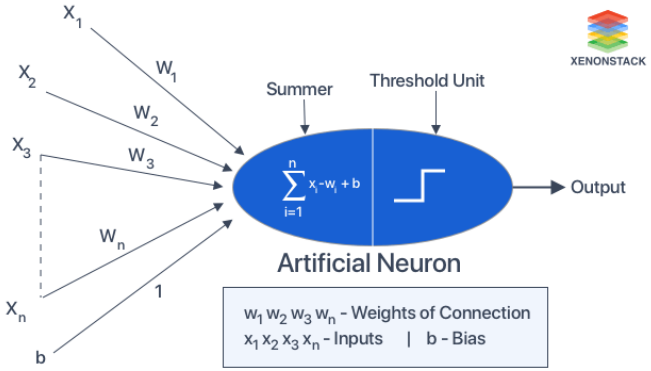


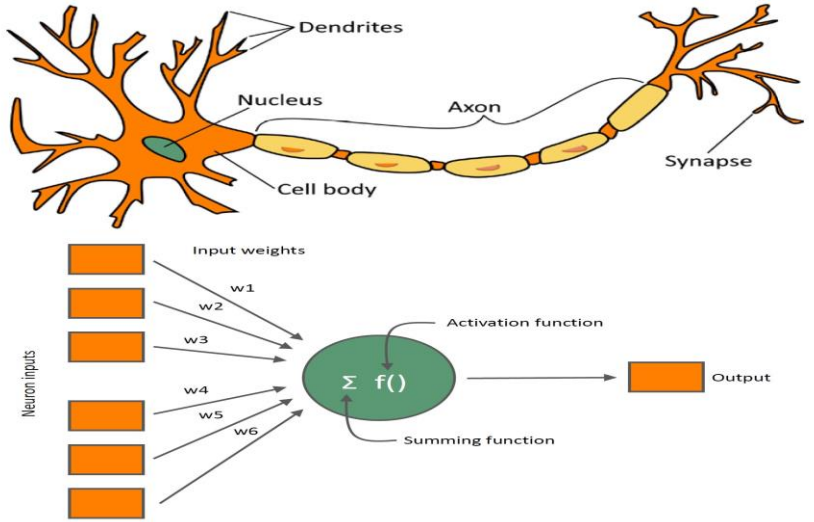
Neural Networks are a set of algorithms that tries to recognize the patterns, relationships, and information from the data through the process which is inspired by and works like the human brain/biology.

PERCEPTRON AND MULTI-LAYER PERCEPTRON

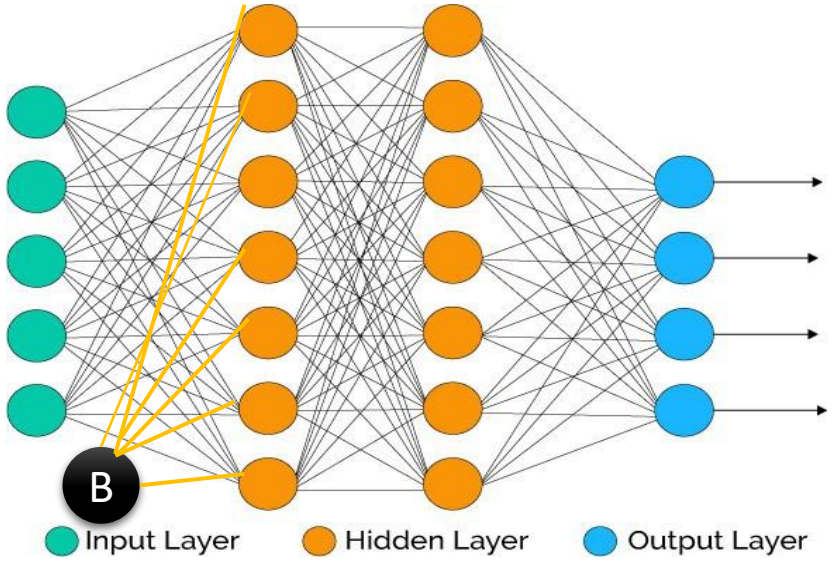
Perceptron is a simple form of Neural Network and consists of a single layer where all the mathematical computations are performed.







Multilayer Perceptron known as **Artificial Neural Networks** consists of more than one perception which is grouped together to form a multiple layer neural network.

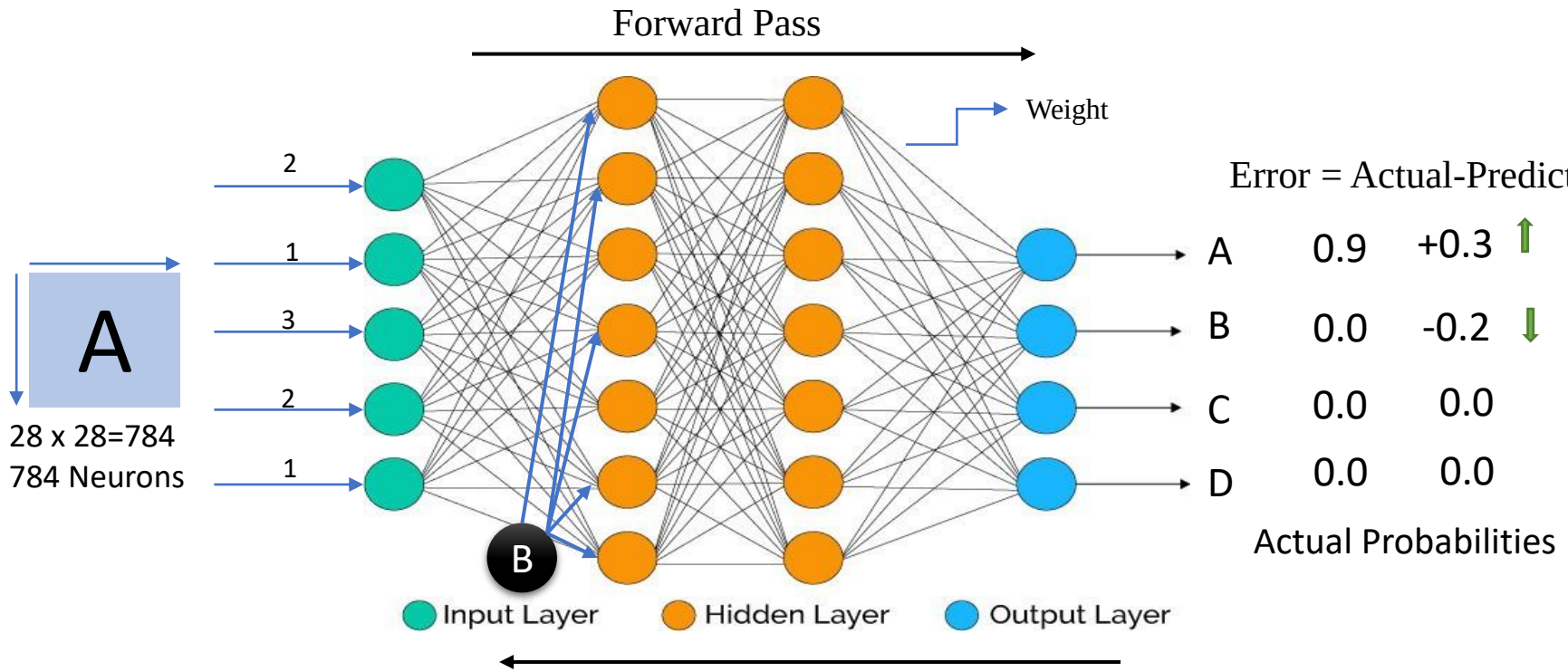


Bias: A bias neuron allows for more variations of weights to be stored. It plays a vital role by making it possible to move the activation function to the left or right on the graph.

STEP BY STEP WORKING OF THE ANN

X=[2,1,3,2,1]

b=0



Error = Actual-Prediction

A	0.9	+0.3 ⬆
B	0.0	-0.2 ⬇
C	0.0	0.0
D	0.0	0.0

Actual Probabilities

H1= f(w.x+b)
= f((0x2)+(1x0)+(3x0)+(2x1)+(1x1)+0)
= f(3)
= 0.9526

Output =f(w.[h1,h2]+b)
=0.7

Loss=[(actual output)-(predicted output)]^2

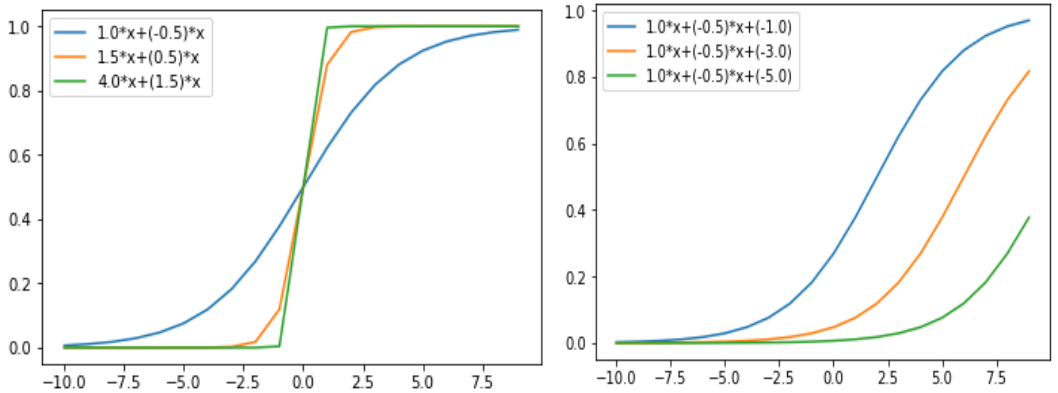
AF (Sigmoid)=$\frac{1}{1+e^{-x}}$

https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network

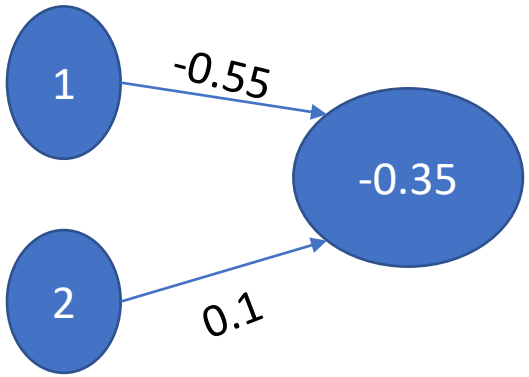
BIAS

How bias impact learning ?

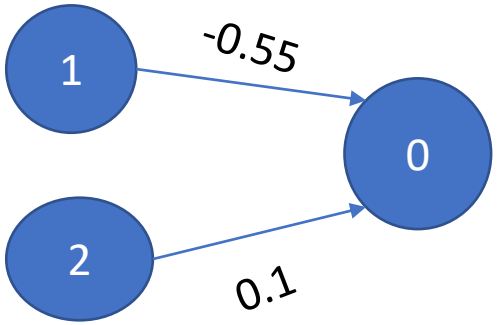
- Each neuron has a bias
- Biases are learnable (Backpropagation)
- Bias determines if a neuron is activated
- Bias increase the flexibility of the model
- Weights and bias influence the learning
- Bias is a constant which helps the model in a way that it can fit best for the given data



$g(w_1,x_1+w_2,x_2+\ldots.+w_nx_n+b)$



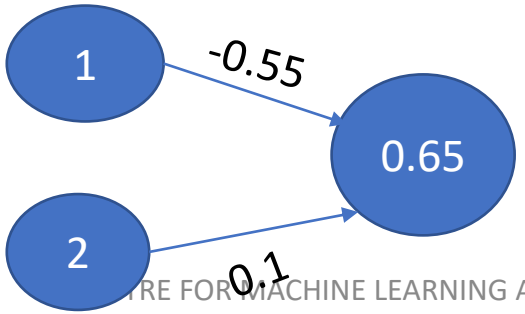
1(-0.55)+2(0.1)=-0.35



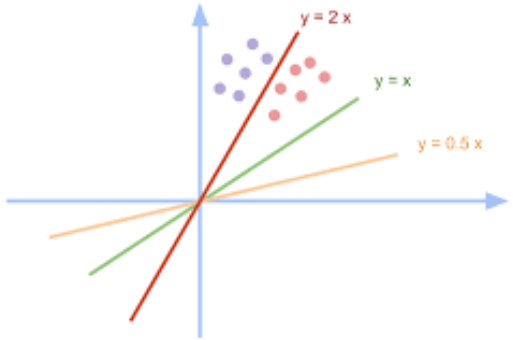
Non activated and no learning

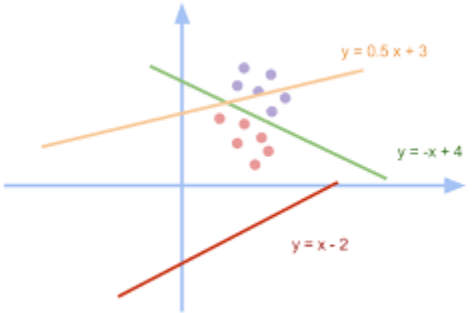
1(-0.55)+2(0.1)+1=0.65

Relu (0.65)=0.65



Activated and Learning





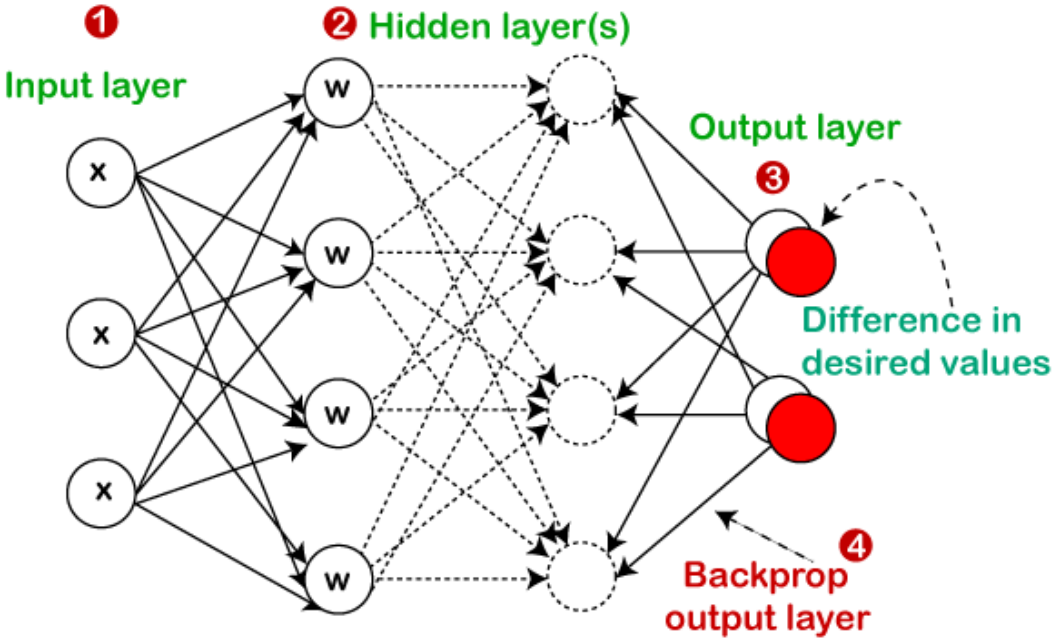
https://www.youtube.com/watch?v=pPOZODXH61s

WHAT IS BACK PROPAGATION AND HOW IT WORKS?

Back Propagation is the process of updating and finding the optimal values of weights or coefficients which helps the model to minimize the error i.e difference between the actual and predicted values.

How the weights are updated and new weights are calculated?

The weights are updated with the help of optimizers.



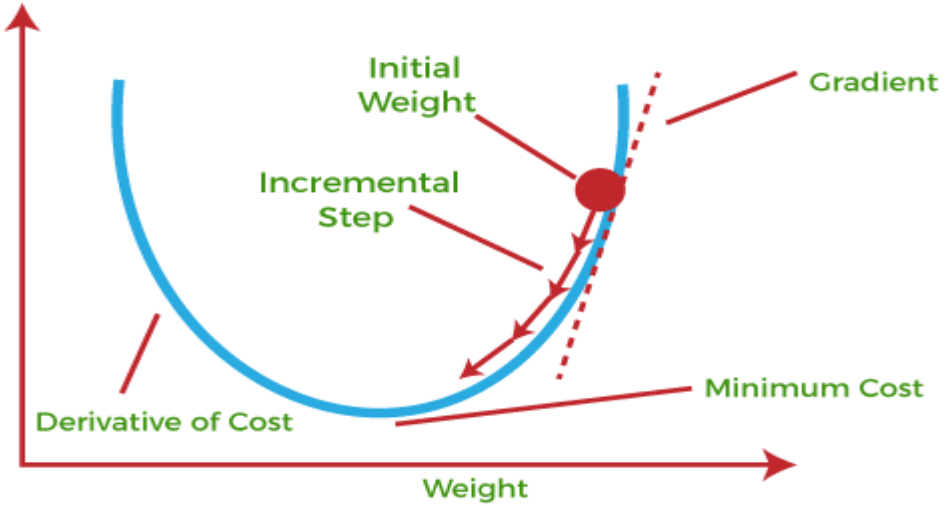
Back Propagation with Gradient Descent

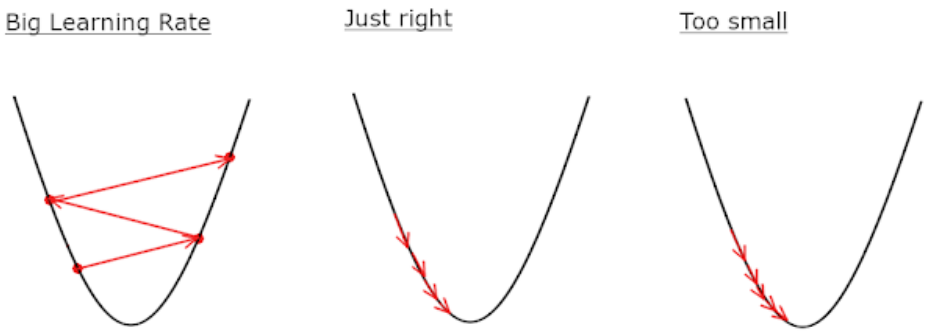
Gradient Descent is one of the optimizers which helps in calculating the new weights.

https://www.jeremyjordan.me/neural-networks-training/

Steps to achieve the global minima:

- **weights are initialized randomly** and **the errors are calculated** after all the computation
- **Gradient is calculated i.e derivative of the error**
- **Learning rate** which is the parameter also known as step size to control the speed or steps of the backpropagation
- Updating of weights **continues till reach global minima and loss is minimized**.

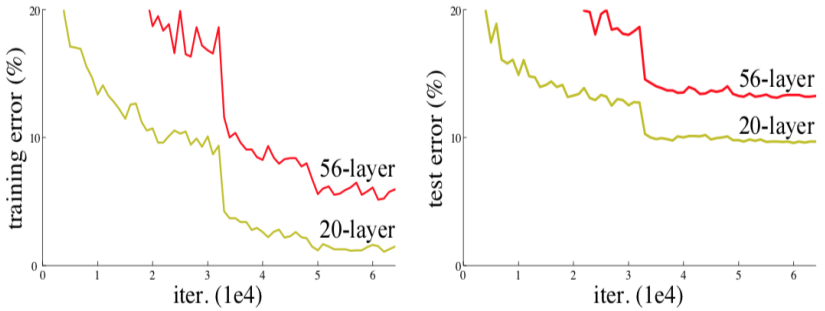


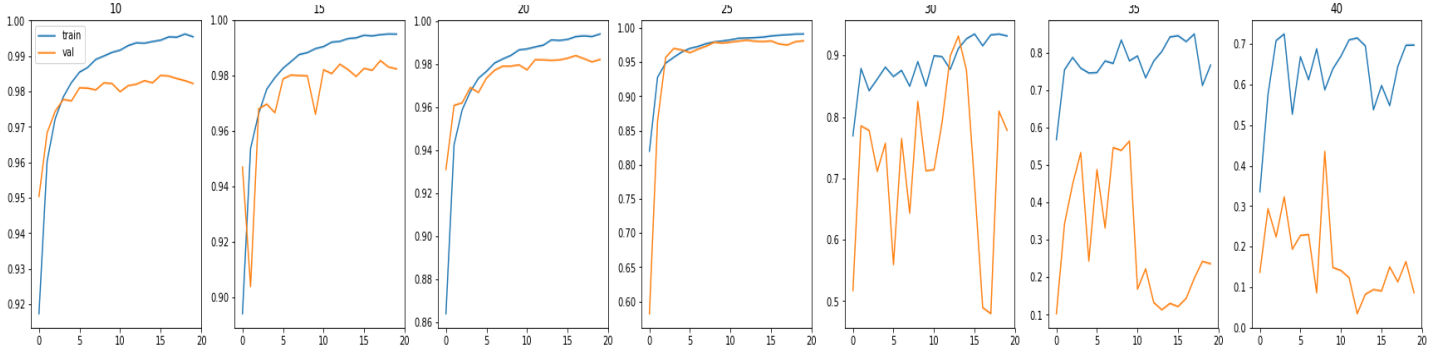


Learning Rate: **It should not be very small** as it will take time to converge as well as **it should not be very large** that it doesn't reach global minima at all.

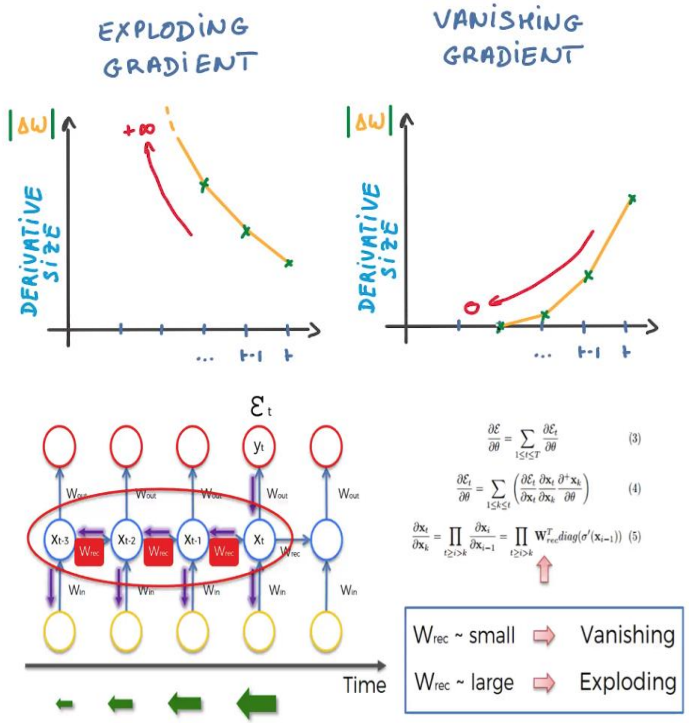
VANISHING GRADIENT PROBLEM IN DL

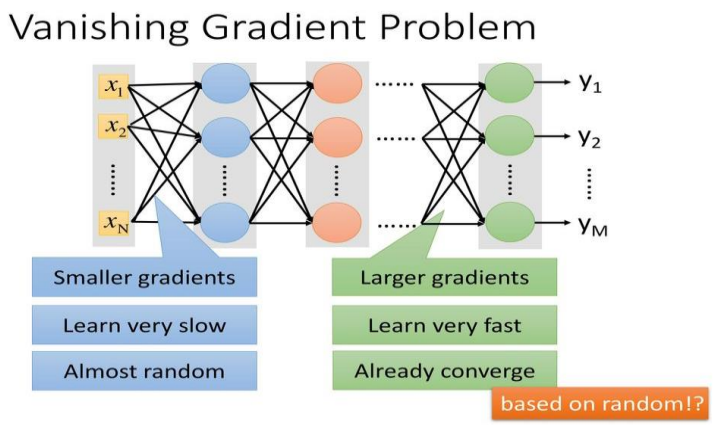
- Gradients that modify the nodes and edges of a neural network deteriorate in magnitude from the output layers towards the input layers during training with backpropagation.
- It was cocurated in the early days of DL when using the sigmoid activation function.
- Use ReLU instead of the sigmoid activation function.
- Increasing the depth of a neural network generally leads to increased accuracy.
- However, with the increasing number of layers in neural networks, the gradients of the loss function with respect to the unknown parameters (weights and biases) may either explode or vanish.





MLP with Batch Normalization and ReLU





NEURAL NETWORK VS DEEP LAERNING

NN

- A neural network is a model of neurons inspired by the human brain.
- Neurons take input, process it, and pass it on to other neurons present in the multiple hidden layers of the network, till the processed output reaches the Output Layer.
- Artificial Neural Network (ANN) has only three layers – the input layer, output layer, and a hidden layer.
- It generally takes less time to train them.
- They have a lower accuracy than Deep Learning Systems.
- Neural Networks use neurons to transmit data in the form of input to get output with the help of the various connections.

Ex:

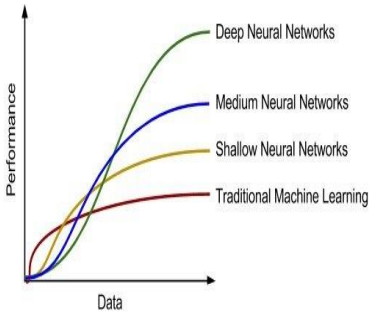
- Feed Forward Neural Networks
- Recurrent Neural Networks
- Symmetrically Connected Neural Networks

DNN

- Deep learning neural networks are distinguished from neural networks on the basis of their depth or number of hidden layers.
- Mimic the computing capabilities of the human brain and create patterns similar to those used by the brain for making decisions.
- It can learn from unstructured or unlabeled data.
- The word “deep” in Deep Learning refers to the number of hidden layers i.e. depth of the neural network.
- It generally takes more time to train them.
- They have a higher accuracy than Deep Learning Systems
- Deep Learning is associated with the transformation and extraction of features that attempt to establish a relationship between stimuli and associated neural responses present in the brain.

Ex:

- Recursive Neural Networks
- Unsupervised Pre-trained Networks
- Convolutional Neural Networks



ANN IN KERAS SIMPLE USE CASE

Problem Statement: Based on the previous records create a Deep Learning-based predictor that will help us to identify if the student is potentially going to fail this year, so the teacher can put more focus on that group of students.

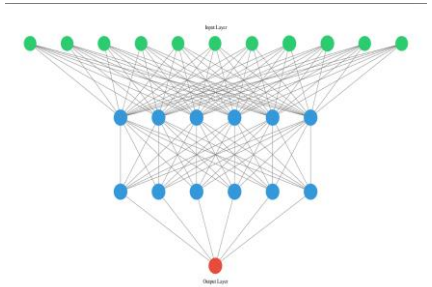
	RowNumber	StudentId	StudentScore	School	Gender	Age	tenure	previousScore	sportsParticipated	Sibling	IsRegular	everFailed	IsFailed
1566	1567	10023959	962	Kendriya Vidyalaya	girl	7	2	349	0	0	0	0	1
5167	5168	10046947	768	Govt Primary School	boy	9	2	463	6	0	0	1	1
6246	6247	10044863	988	Govt Primary School	boy	7	4	698	3	1	1	0	0
5491	5492	10022268	579	Kendriya Vidyalaya	girl	8	3	502	3	0	1	0	1
3272	3273	10002543	824	Govt Primary School	girl	11	4	452	0	0	0	0	1

	StudentScore	School	Gender	Age	tenure	previousScore	sportsParticipated	Sibling	IsRegular	everFailed
3285	615	1	1	7	3	266	3	1	0	1
2616	650	2	0	9	3	576	1	0	0	0
5022	941	0	0	6	4	541	6	0	0	0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import keras
from keras.models import Sequential
cf = Sequential()
from keras.layers import Dense
cf.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
cf.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
cf.add(Dropout(rate = 0.1))
cf.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

from ann_visualizer.visualize import ann_viz;
ann_viz(network, title="nn");
```



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

```
cf.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

```
y_prediction =cf.predict(X_test)
```

```
accuracies = cross_val_score(estimator = cf, X = X_train, y = y_train, cv = 10, n_jobs = -1)
```

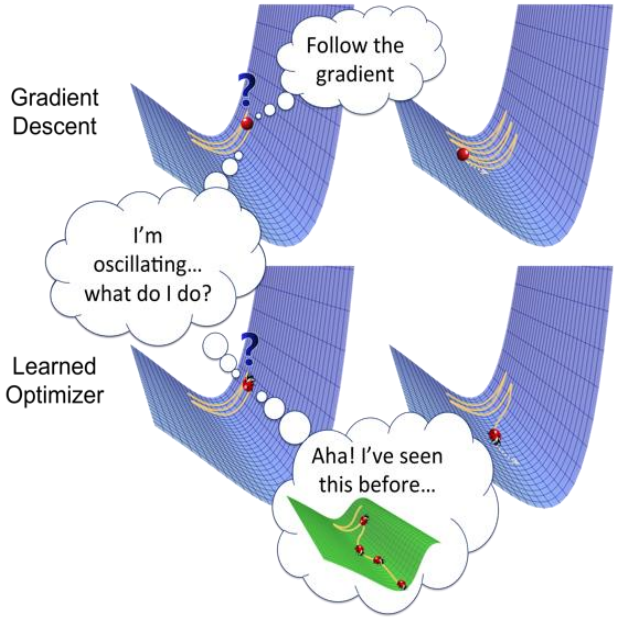
```
parameters = {'batch_size': [10, 15], 'epochs': [10, 50], 'optimizer': ['adam', 'rmsprop']}
gv_search = GridSearchCV(estimator = cf, param_grid = parameters, scoring = 'accuracy', cv = 10)
gv_search = gv_search.fit(X_train, y_train)
best_param = gv_search.best_params_
best_acc = gv_search.best_score_
```

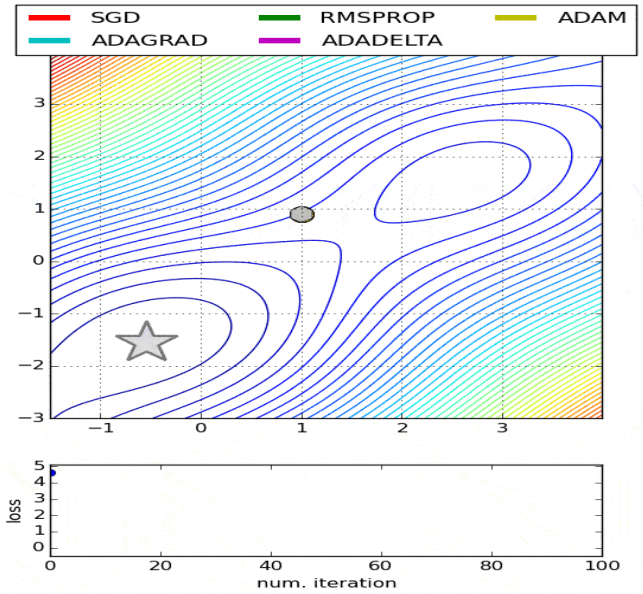
Grid Search

OPTIMIZERS IN DEEP LEARNING

What are Optimizers?

- Minimize an error function or maximize the efficiency of production
- Change the attributes of neural networks such as weights and learning rate in order to reduce the losses and improve accuracy.





Types of Optimizers in DL

Can we create our Own Optimizers for our problem if needed?

- Gradient Descent
- Learning Rate
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- SGD with Momentum
- Nesterov Accelerated Gradient (NAG)
- AdaGrad(Adaptive Gradient Descent)
- RMS-Prop (Root Mean Square Propagation)
- AdaDelta
- Adam(Adaptive Moment Estimation)

OPTIMIZERS IN DEEP LEARNING

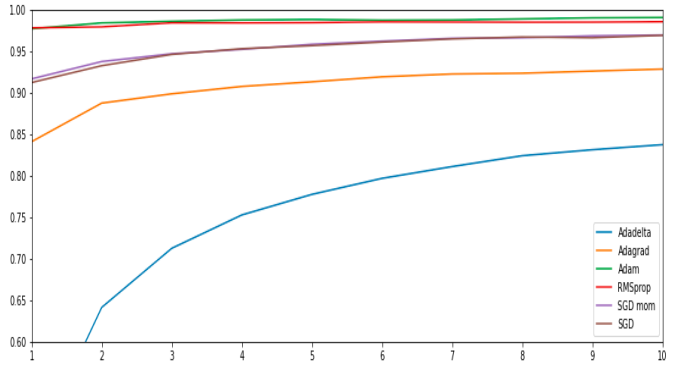
```
import keras from keras.datasets
import mnist from keras.models
import Sequential from keras.layers
import Dense, Dropout, Flatten from keras.layers
import Conv2D, MaxPooling2D from keras
import backend as K (x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
```

```
x_train= x_train.reshape(x_train.shape[0],28,28,1)
x_test= x_test.reshape(x_test.shape[0],28,28,1)
input_shape=(28,28,1)
y_train=keras.utils.to_categorical(y_train)#,num_classes=)
y_test=keras.utils.to_categorical(y_test)#, num_classes)
x_train= x_train.astype('float32') x_test= x_test.astype('float32')
x_train /= 255
x_test /=255
```

```
batch_size=64
num_classes=10
epochs=10
def build_model(optimizer):
   model=Sequential()
   model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape))
   model.add(MaxPooling2D(pool_size=(2,2))) model.add(Dropout(0.25))
   model.add(Flatten())
   model.add(Dense(256, activation='relu'))
   model.add(Dropout(0.5))
   model.add(Dense(num_classes, activation='softmax'))
   model.compile(loss=keras.losses.categorical_crossentropy, optimizer= optimizer,
                                                    metrics=['accuracy'])
 return model
```

```
optimizers = ['Adadelta', 'Adagrad', 'Adam', 'RMSprop', 'SGD']
for i in optimizers:
    model = build_model(i)
hist=model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
validation_data=(x_test,y_test))
```

Optimizer	Epoch 1 Val accuracy | Val loss	Epoch 5 Val accuracy | Val loss	Epoch 10 Val accuracy | Val loss	Total Time
Adadelta	.4612 | 2.2474	.7776 | 1.6943	.8375 | 0.9026	8:02 min
Adagrad	.8411 | .7804	.9133 | .3194	.9286 | 0.2519	7:33 min
Adam	.9772 | .0701	.9884 | .0344	.9908 | .0297	7:20 min
RMSprop	.9783 | .0712	.9846 | .0484	.9857 | .0501	10:01 min
SGD with momentum	.9168 | .2929	.9585 | .1421	.9697 | .1008	7:04 min
SGD	.9124 | .3157	.9569 | 1451	.9693 | .1040	6:42 min



ACTIVATION FUNCTION

- Segregate between relevant and irrelevant information.
- Mechanism for classifying incoming information as *"useful"* or *"less-useful"* in the case of Neural Networks.
- Network uses the important information and suppresses the irrelevant data points.

Can we do without an activation function?

- A neural network without an activation function is essentially just a linear regression model.

How to choose the Right Activation Function?

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results

- Elu
- Binary
- Linear
- Sigmoid → Classifier
- Tanh
- ReLU
- Leaky ReLU
- Parameterised ReLU
- Exponential Linear Unit
- Swish
- Softmax

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

https://keras.io/api/layers/activations/ https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

LOSS FUNCTION

- Loss is the measure of our model's accuracy.
- Difference between actual and predicted values.
- Different loss functions for regression and classification

Example:

- Cross-entropy is used to define loss function for the optimization.
- MQE for calculating error between actual and predicted

Probabilistic Loss

- Cross-entropy
- Binary Cross-entropy
- Categorial Crossentropy

Regression Loss

- Absolute error
- Squared error
- MQE
- Mean Absolute Error

Hinge losses for "maximum-margin" classification

- Huber loss
- squaredHinge Class

https://keras.io/api/losses/

BATCH SIZE VS EPOCH VS ITERATION

Epoch

- An epoch represents one iteration over the entire dataset
- One forward and backward pass of all training data
- (not sufficient for one time so go multiple times for update weights)

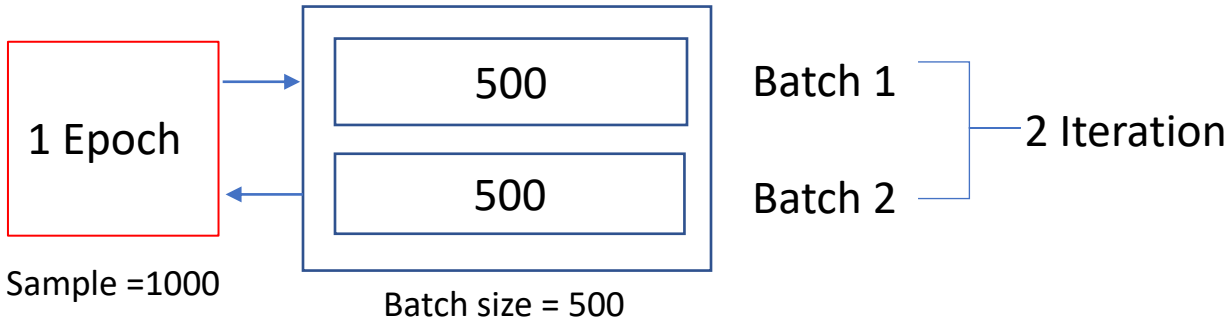
Batch

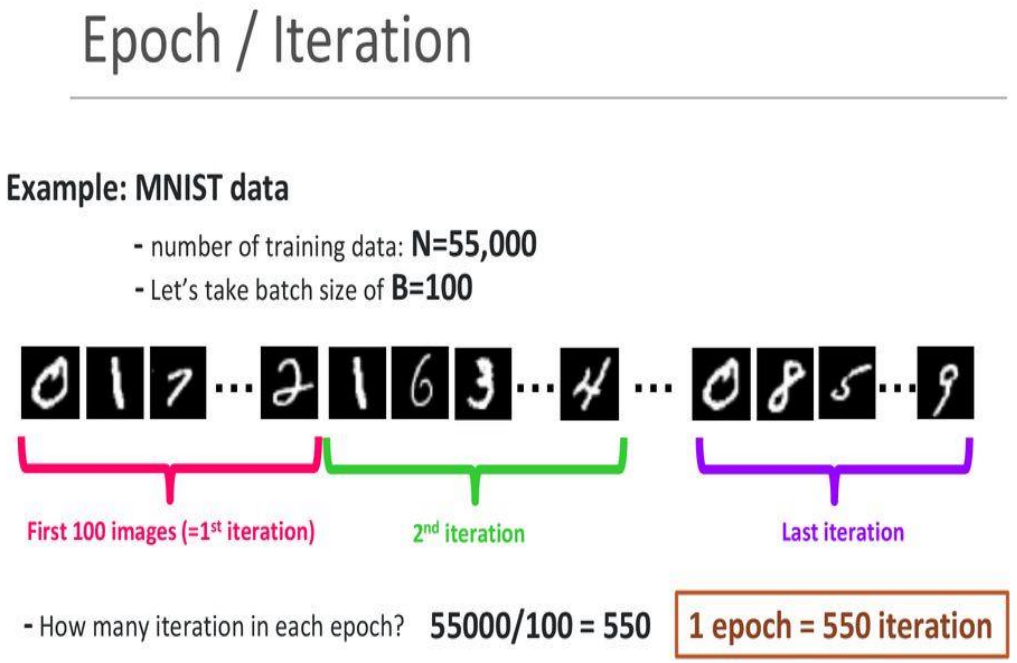
- Divide the dataset into a number of batches
- Batch size influences both speed and performance.
- The number of training examples in one forward and backward pass

Iteration

- Number of passes need to complete one epoch

If we have 1000 images as Data and batch size is 500, then an Epoch should run 1000/500=2 iteration. Therefore 1 epoch=500 iteration





Assume : Total image=200, Batch Size=5 epoch=1000. 200/5=40 iteration.
It takes a total of 40,000 batches during the entire training process

HOW TO CHOOSE OPTIMAL NUMBER OF EPOCH?

- The number of epochs will decide how many times we will change the weights of the network.
- As the number of epochs increases, the same number of times weights are changed in the neural network and the boundary goes from underfitting to optimal to overfitting.

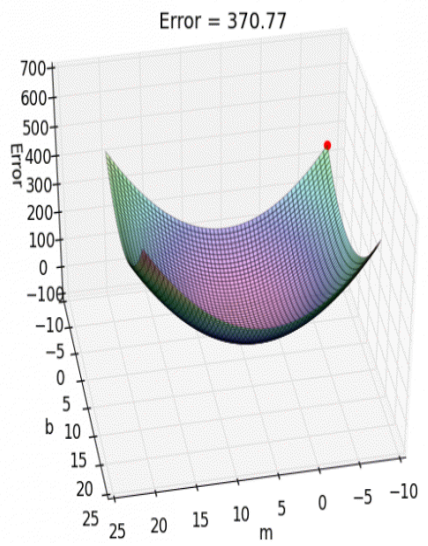
So, what is the right number of epochs?

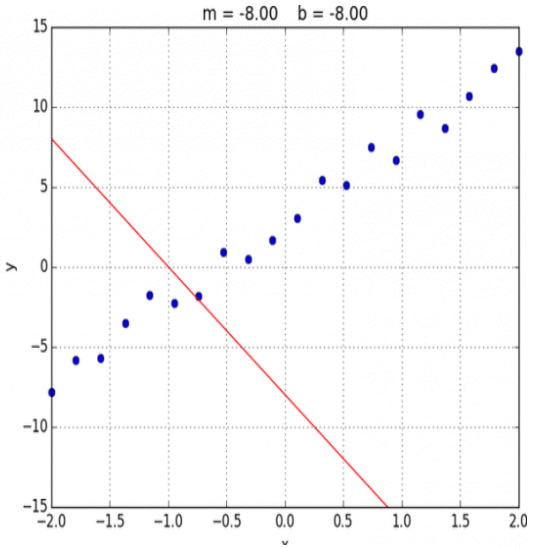
- Number of epochs is not that significant
- Set the number of epochs as high as possible and terminate the training when validation errors start increasing.

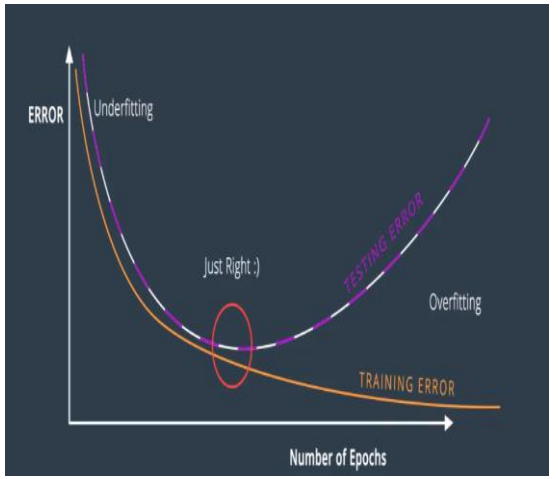
Batch Gradient Descent -Batch size is set to the total number of examples in the training dataset.

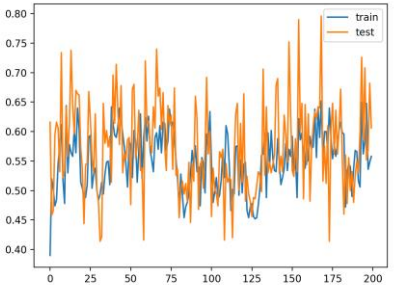
Stochastic Gradient Descent -Batch size is set to one.

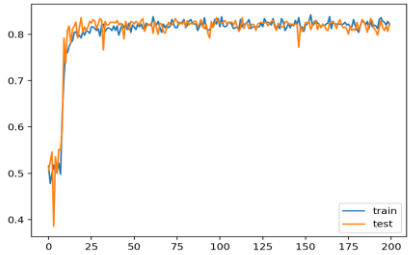
Minibatch Gradient Descent - Batch size is set to more than one and less than the total number of examples in the training dataset.





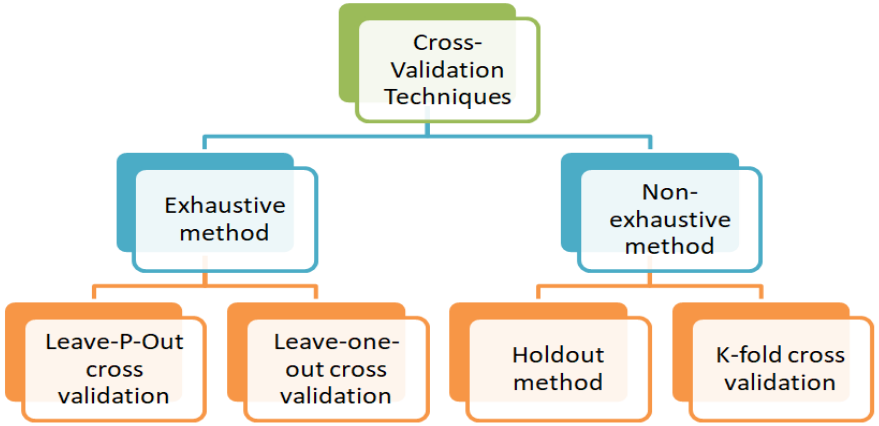


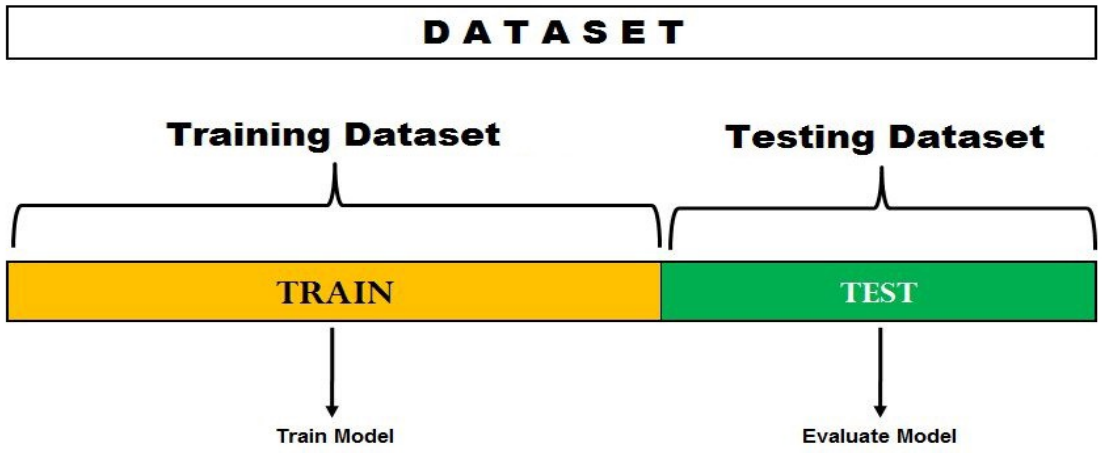




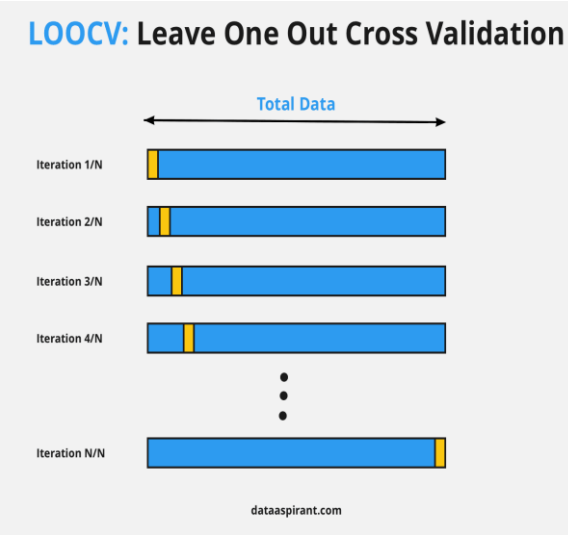
CROSS VALIDATION

- Deep learning models are not always stable and have to evaluate the stability of the machine learning model. (It means Accuracy change over time in normal train & test split)
- Cross-Validation is a technique used to assess how well our deep learning models perform on unseen data.
- To overcome over-fitting problems, use a technique called Cross-Validation.

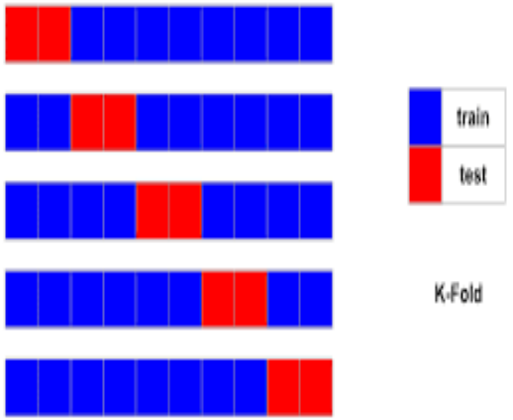




Hold-out Method : 70-30, 60-40,75-25,80-20 or 50-50 (Train data >= Test data)



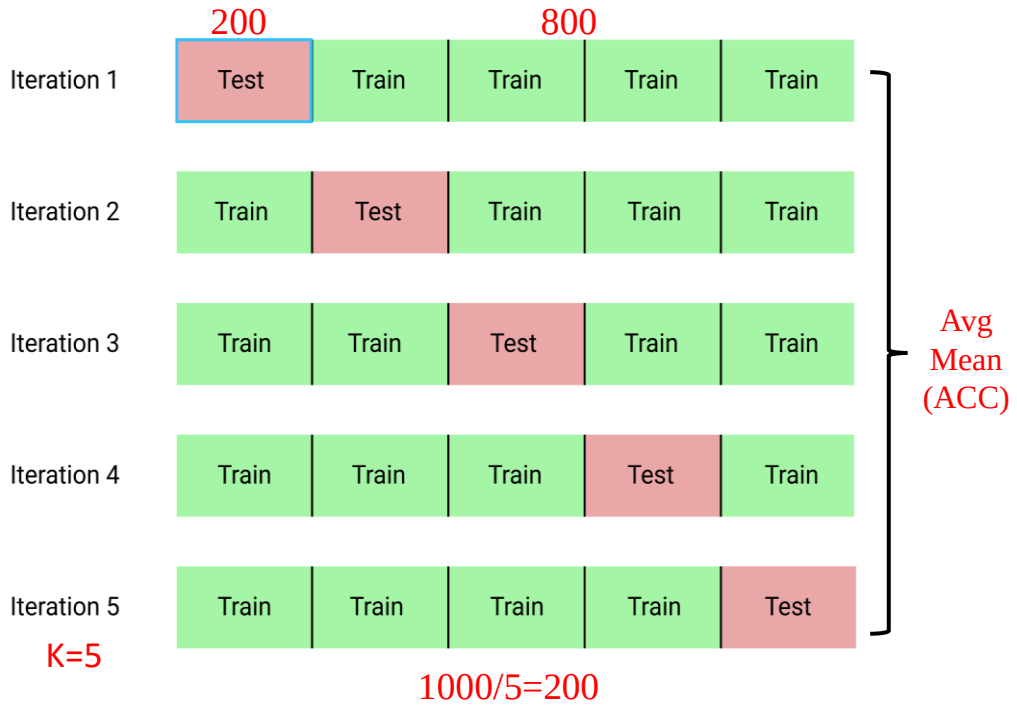
1- test , remaining -training



Leave-P-Out

K-FOLD CROSS VALIDATION

- Divide the sample parts into k parts
- Use k-1 of the parts for training, and 1 for testing
- Repeat the procedure k times, rotating the test set.
- Determine an expected performance metric (mean square error, misclassification error rate, confidence interval, or another appropriate matric) based on the results across the iterations

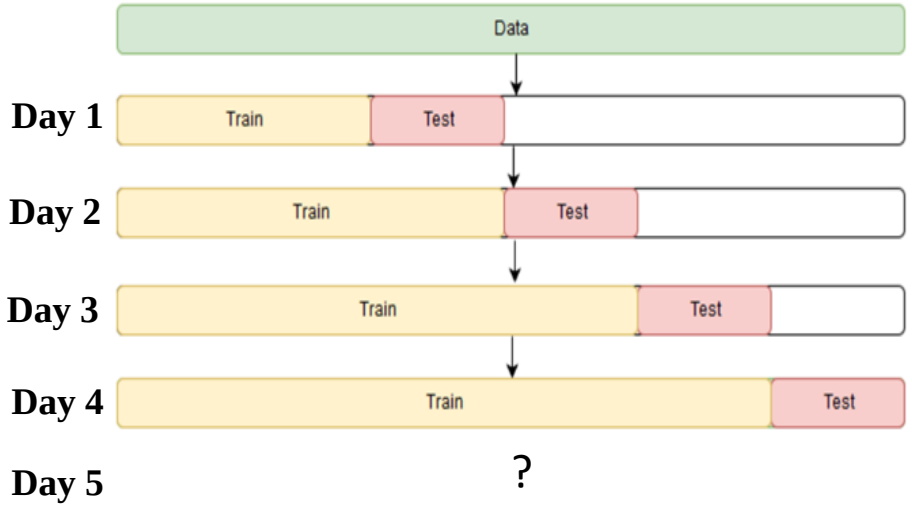


Stratified Cross Validation

- Here, the proportion of classes in variables will get distributed equally among test train sets.
- So test-train split is balance.
- Thus the disadvantage of k fold is solved by stratified cross-validation.

Time series Cross-validation

Eg: Stock prize detection



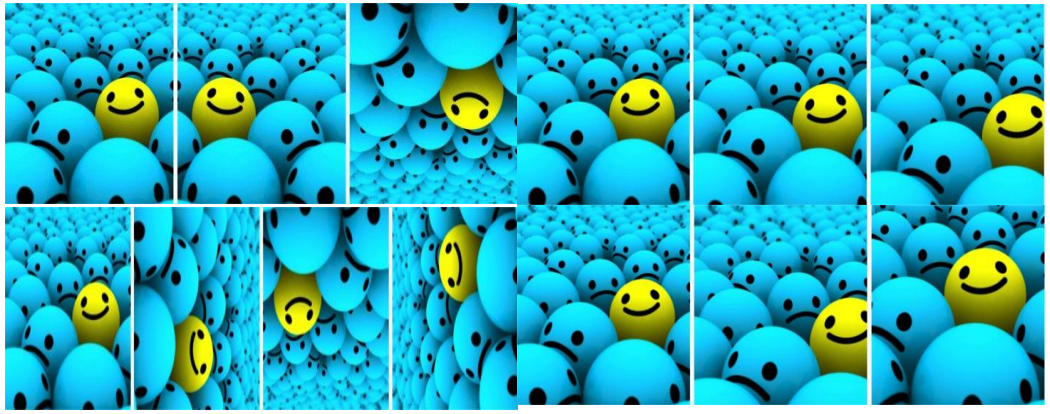
DATA AUGMENTATION

- A convolutional neural network that can robustly classify objects even if they its placed in different orientations is said to have the property called invariance.

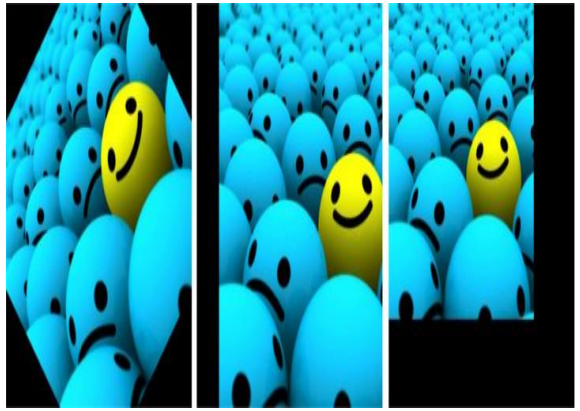
There are two types:

- Offline augmentation
- Online augmentation (on the fly)

Flip, Rotation, Scale Crop, Translation, Gaussian Noise



Interpolation



Constant, Edge, Reflect, Symmetric, Wrap

Conditional GAN

Make sure to **not increase irrelevant data**.

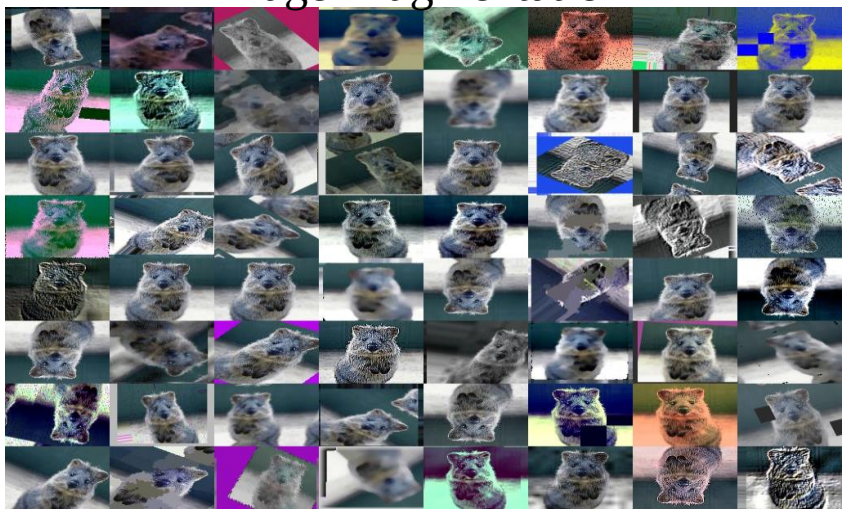
Your neural network is only as good as the data you feed it.

DATA AUGMENTATION

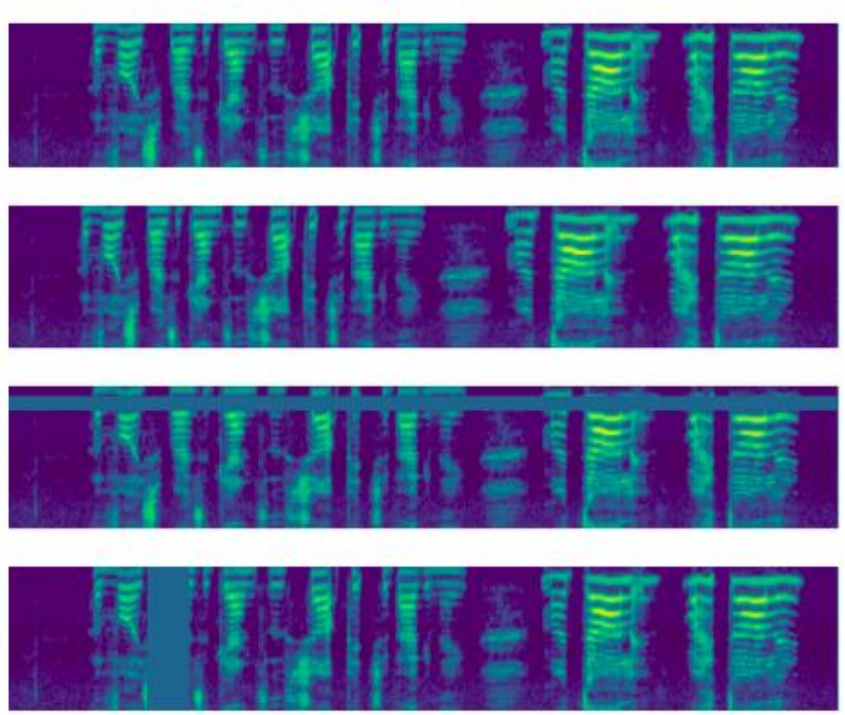
- Using flow method
- Using flow_from_dataframe method
- Using flow_from_directory method

```
rotation_range=0,
width_shift_range=0.0,
height_shift_range=0.0,
brightness_range=None,
shear_range=0.0,
zoom_range=0.0,
channel_shift__range=0.0,
fill_mode="nearest",
horizontal_flip=False,
vertical_flip=False,
rescale=None,
preprocessing_function=None,
validation_split=0.0
```

Image Augmentation



- https://fairyonice.github.io/Learn-about-ImageDataGenerator.html
- https://medium.com/analytics-vidhya/understanding-image-augmentation-using-keras-tensorflow-a6341669d9ca



Audio Augmentation

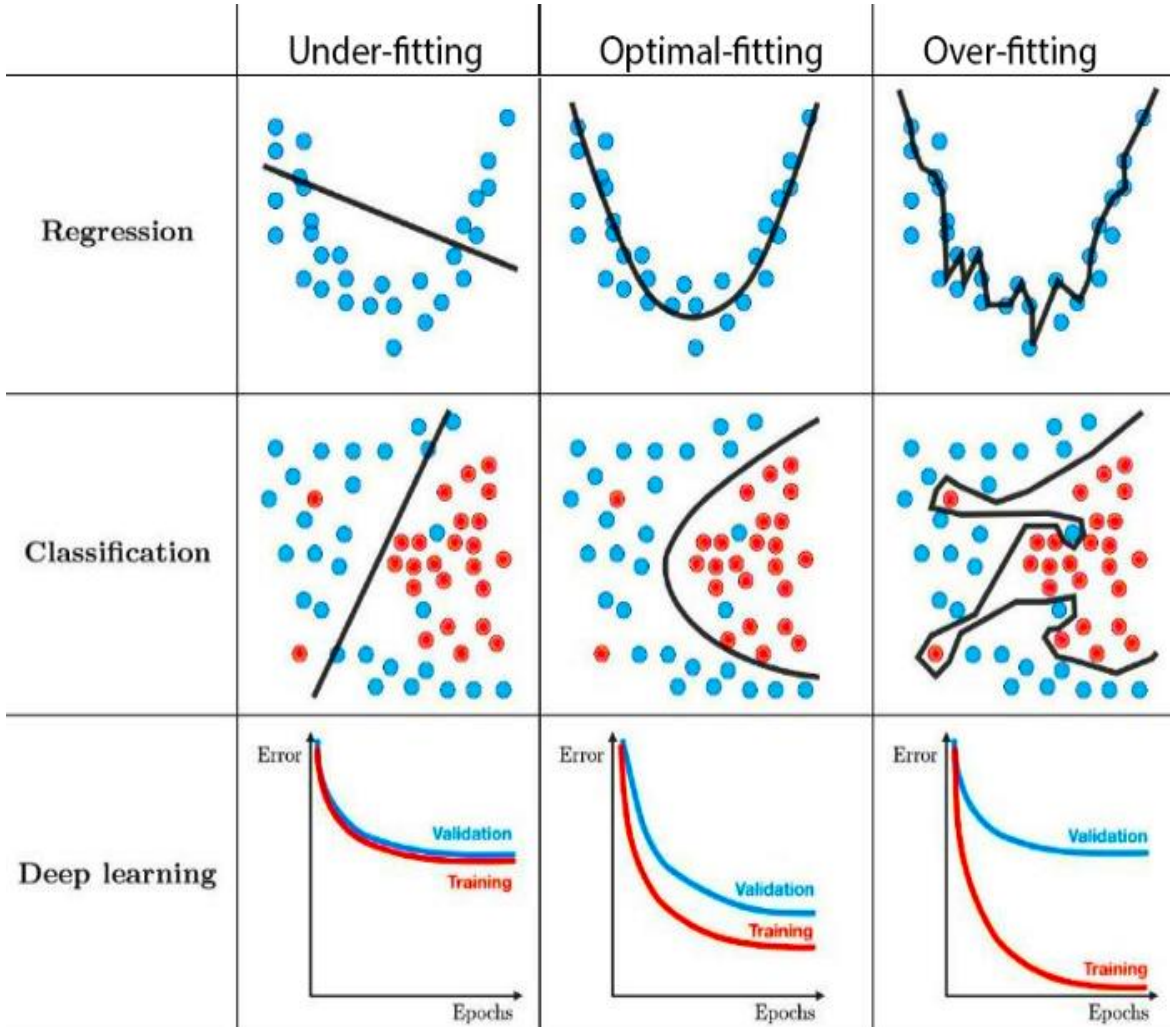
OVERFITTING & UNDERFITTING

Overfitting

- The models perform poorly on new examples (e.g., Testing) as it is too highly trained to the specific training examples (pick up patterns and noises)

Underfitting

- The model performs poorly on new examples as it is too simplistic to distinguish between them.



OVERFITTING & UNDERFITTING

Overfitting

- Model is too complex
- Accurate for the training set
- Not accurate for the testing set

Solution:

- Need to reduce model complexity
- Reduce the number of features
- Apply L1 & L2 regularization
- Increase training data
- Dropouts for Neural Network
- Early stopping during training phase
- ReduceLROnPlateau – reduce learning rate

Underfitting

- Model is not complex enough
- Not accurate for the training set
- Not accurate for the testing set

Solution:

- Need to increase the complexity
- Increase number of features
- Reduce regularization
- Increase training data
- Add no. of training epochs
- Remove noise from data

EARLY STOPPING METHOD

```
filepath='/content/drive/My Drive/All ss'
early_stopping = EarlyStopping ( monitor='val_accuracy',
        mode='max', min_delta=0.0001, baseline=0.98, patience=6,
        restore_best_weights=True, verbose=1 )
checkpoint = ModelCheckpoint
(filepath,monitor='val_loss',mode='min',save_best_only=True,verbos
e=1)
 lr_scheduler = ReduceLROnPlateau ( monitor='val_accuracy',
                    patience=3, factor=0.25, min_lr=1e-5 )
 callbacks = [ early_stopping, lr_scheduler, TerminateOnNaN(),
checkpoint ]
```

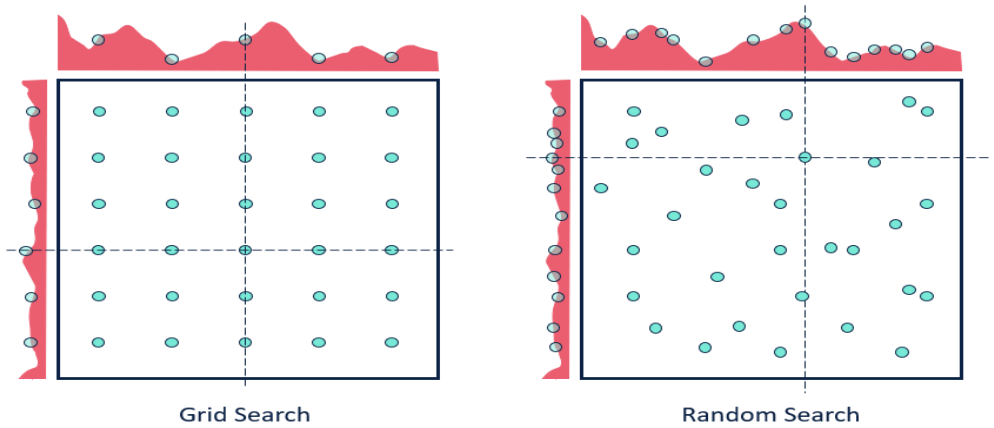
```
model = build_cnn()

history = model.fit ( x= X_train, y= y_train, validation_data=
(X_valid, y_valid), batch_size=64, epochs=30, callbacks=callbacks,
use_multiprocessing=True )
```

Monitor : Quantity to be monitored.
Mode : auto, min, max
Min-delta : Minimum change in the monitored quantity to qualify as an improvement
Baseline : Baseline value for the monitored quantity.
Patience : Number of epochs with no improvement.
Min_lr : lower bound on the learning rate.
Factor : The factor by which the learning rate will be reduced.
Cooldown : The number of epochs to wait before resuming normal operation after lr has been reduced.
ReduceLROnPlateau () : reduce the learning rate.

Restore_best_weight : Whether to restore model weights from the epoch with the best value of the monitored quantity.
Verbose : Verbosity mode, 0 or 1. Mode 0 is silent, and mode 1 displays messages.
TerminateOnNaN() : Callback that terminates training when a NaN loss is encountered.

HYPER PARAMETER TUNING

- The process of setting the hyper-parameters requires expertise and extensive trial and error.
- Hyper-parameters: learning rate, batch size, momentum, and weight decay.
- Learning rate starts with 0.0001 to 1
- Batch size : 10, 30, 40…
- Momentum : 0.99,0.97,0.95, and 0.9
- Dropout : 0.3,0.5,025



- Try to divide the original problem into smaller ones.
- Increase the frequency of monitoring validation accuracy.
- Plot one curve per learning rate (each epoch)
- Tuning regularization parameter (Weight Delay)
- Tuning Mini-Batch Size -Plot the validation accuracy versus time
- Tuning training Epoch
- Use different cost function
- Try different regularization
- Initialize weights differently
- Apply various gradient descent
- Try Hessian technique
- Try different activation functions

https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide

HYPER PARAMETER TUNING

- Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance.
- It works by running multiple trials in a single training process.

How to do Hyperparameter Tuning?

- Manual search
- Automatic Search

Hyperparameter Tuning Method:

- Random Search - Random combination of hyperparameters
- Grid Search – Specific Order
- Bayesian Optimization - Minimal point in the minimum number of steps.
- Tree-structed Parzen estimators (TPE) - Similar to Bayesian optimization.

Hyperparameter Tuning Algorithm

- Hyperband
- Population-based training (PBT)
- BOHB (Bayesian Optimization and HyperBand)

Popular Tools/Library for Hyperparameter Tuning

- Scikit-learn
- Scikit-Optimize
- Optuna
- Hyperopt
- Keras Tunner
- BayesianOptimization
- GPyOpt
- SigOpt
- Fabolas
- Spearmint
- Metric Optimization Engine

https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/#:~:text=The%20hyperparameters%20to%20tune%20are,layers%20can%20affect%20the%20accuracy.

KERAS TUNNER

```
!pip install -q -U keras-tuner
```

```
import tensorflow as tf
from tensorflow import Keras
import numpy as np
```


```
fashion_mnist=keras.datasets.fashion_mnist
```

```
(train_images,train_labels),(test_images,test_
                labels)=fashion_mnist.load_data()
```

```
train_images=train_images/255.0
test_images=test_images/255.0
```

```
train_images=train_images.reshape(len(train_images),28,28,1)
test_images=test_images.reshape(len(test_images),28,28,1)
```

https://colab.research.google.com/drive/1kPO_YjAyNjc6YeRnw2b3dTkjEKhq0MOl

```
def build_model(hp):

  model = keras.Sequential([

   keras.layers.Conv2D(
       filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
       kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
       activation='relu',
       input_shape=(28,28,1)
   ),
keras.layers.Conv2D(
       filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
       kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
       activation='relu'
   ),
  keras.layers.Flatten(),
   keras.layers.Dense(
       units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
       activation='relu'
   ),
   keras.layers.Dense(10, activation='softmax')
  ])

  model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', v
                                                          alues=[1e-2, 1e-3])),
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
  return model
```

```
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters
import HyperParameters
```

```
tuner_search=RandomSearch(build_model,
                objective='val_accuracy',
max_trials=5,directory='output',
project_name="Mnist Fashion")
```

```
tuner_search.search(train_images,train_labels,epochs
=3,validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000
samples Epoch 1/3 54000/54000
[==============================] - 15s
278us/sample - loss: 0.3919 - accuracy: 0.8587
- val_loss: 0.2863 - val_accuracy: 0.8948
Epoch 2/3 54000/54000
[==============================] - 9s
162us/sample - loss: 0.2458 - accuracy: 0.9087
- val_loss: 0.2512 - val_accuracy: 0.9087
Epoch 3/3 54000/54000
[==============================] - 9s
159us/sample - loss: 0.1847 - accuracy: 0.9316
- val_loss: 0.2424 - val_accuracy: 0.9132
```

Trial complete
Trial summary
Hp values:
|-conv_1_filter: 112
|-conv_1_kernel: 3
|-conv_2_filter: 48
|-conv_2_kernel: 5
|-dense_1_units: 80
|-learning_rate: 0.001
|-Score: 0.9131666421890259

Trial complete
Trial summary
Hp values:
|-conv_1_filter: 64
|-conv_1_kernel: 3
|-conv_2_filter: 32
|-conv_2_kernel: 5
|-dense_1_units: 128
|-learning_rate: 0.01
|-Score: 0.8585000038146973

Trial summary
Hp values:
|-conv_1_filter: 80
|-conv_1_kernel: 3
|-conv_2_filter: 32
|-conv_2_kernel: 5
|-dense_1_units: 96
|-learning_rate: 0.001
|-Score: 0.9160000085830688

Model: "sequential" model.summary()

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 80)	800
conv2d_1 (Conv2D)	(None, 22, 22, 32)	64032
flatten (Flatten)	(None, 15488)	0
dense (Dense)	(None, 96)	1486944
dense_1 (Dense)	(None, 10)	970

Total params: 1,552,746 Trainable params: 1,552,746 Non-trainable params: 0

ANN VS CNN IN IMAGE CLASSIFICATION

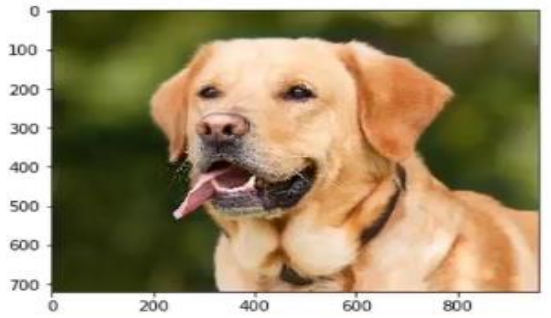
- With ANN, concrete data points must be provided.
- For example, in a model that tries to distinguish between dogs and cats, the width of the noses and length of the ears must be explicitly provided as data points.
- Using ANN, image classification problems become difficult because 2-dimensional images need to be converted to 1-dimensional vectors.
- This increases the number of trainable parameters exponentially.
- Increasing trainable parameters takes storage and processing capability.
- ANN is a comparatively lightweight way of solving data classification problems.

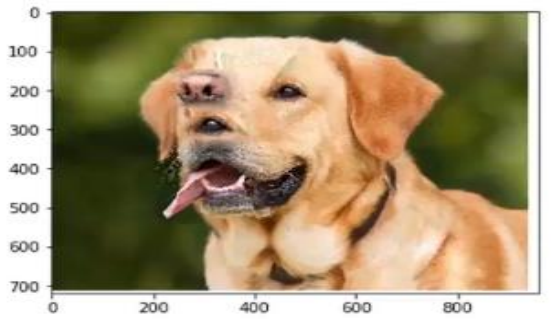
- Spatial features are extracted from image input.
- It automatically detects the important features without any human supervision.
- CNN works in a compatible way with images as input data. Using filters on images results in feature maps.
- CNN doesn't process data in a forward-facing way but rather refers to the same data multiple times when creating maps.
- This is why CNN would be an ideal solution to computer vision and image classification problems.
- CNN tends to be a more powerful and accurate way of solving classification problems.

WHY CONVOLUTIONAL NEURAL NETWORK?

CNN-extract useful features and filters used to reduce parameters

Spatial Orientation





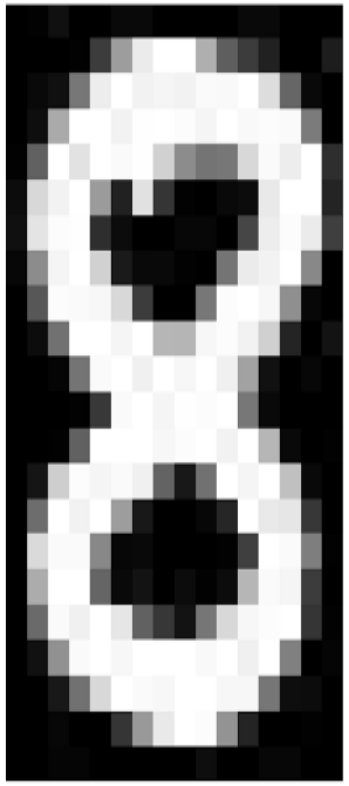
Problem in ANN:

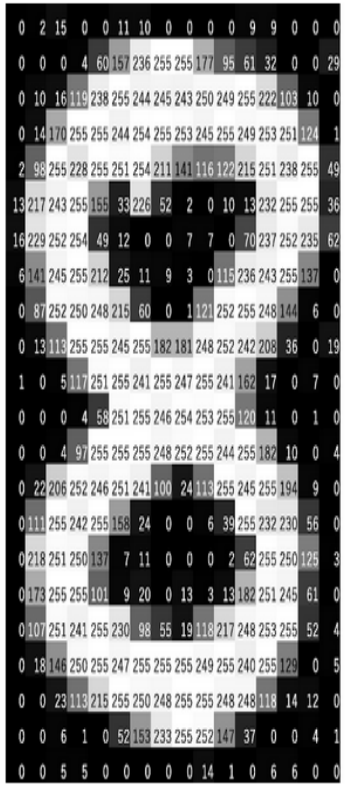
Large number of parameters :

28*28*3=2,352.

what if 224x224*3= 1,50,528

FEATURE EXTRACTION IN CNN

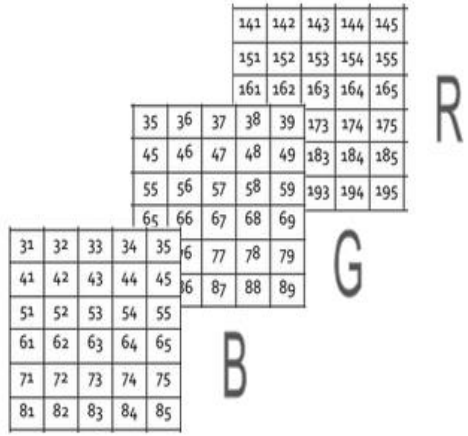




22x18 = 396

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0		
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29		
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0		
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1		
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49		
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36		
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62		
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0		
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0		
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19		
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0		
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0		
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4		
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0		
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0		
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3		
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0		
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4		
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5		
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0		
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1		
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0		

Colour Image



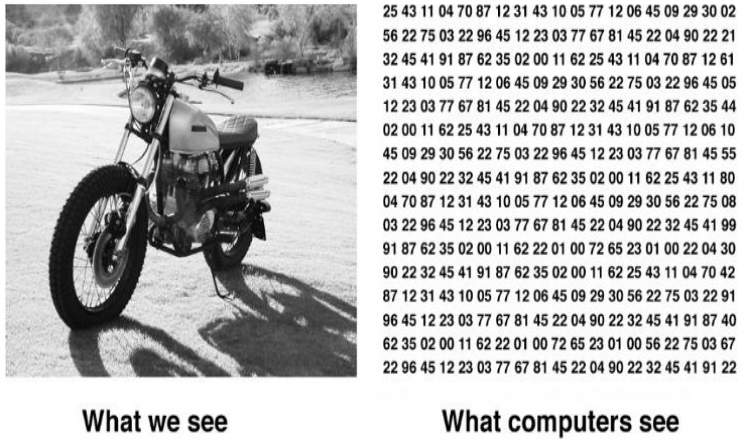
CONVOLUTIONAL NEURAL NETWORK

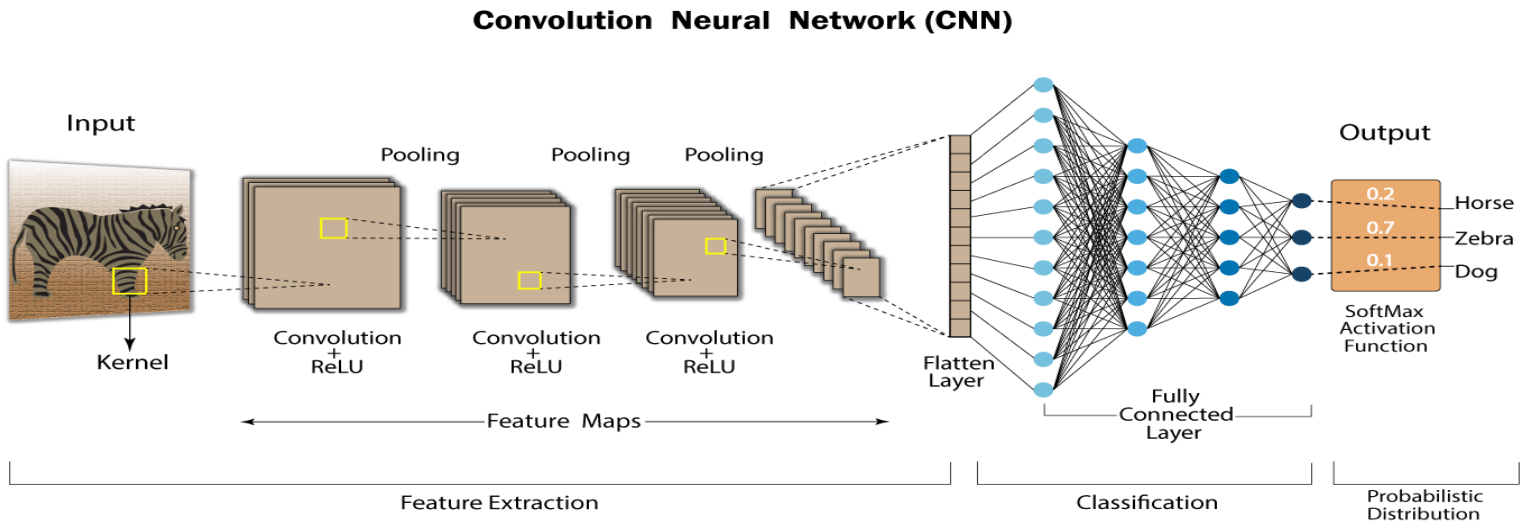
- First neural network called LeNet in 1988
- Used for character recognition tasks like reading zip codes and digits
- Yann LeCun, director of Facebook's AI Research Group
- CNN are a class of DNN that can recognize and classify particular features from image and are widely used for analyzing visual images.

There are two parts to CNN

- Feature extraction
- Classification

- Separates and identifies the various features of the image for analysis in a process called feature extraction
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the images based on the features extracted in previous stages.



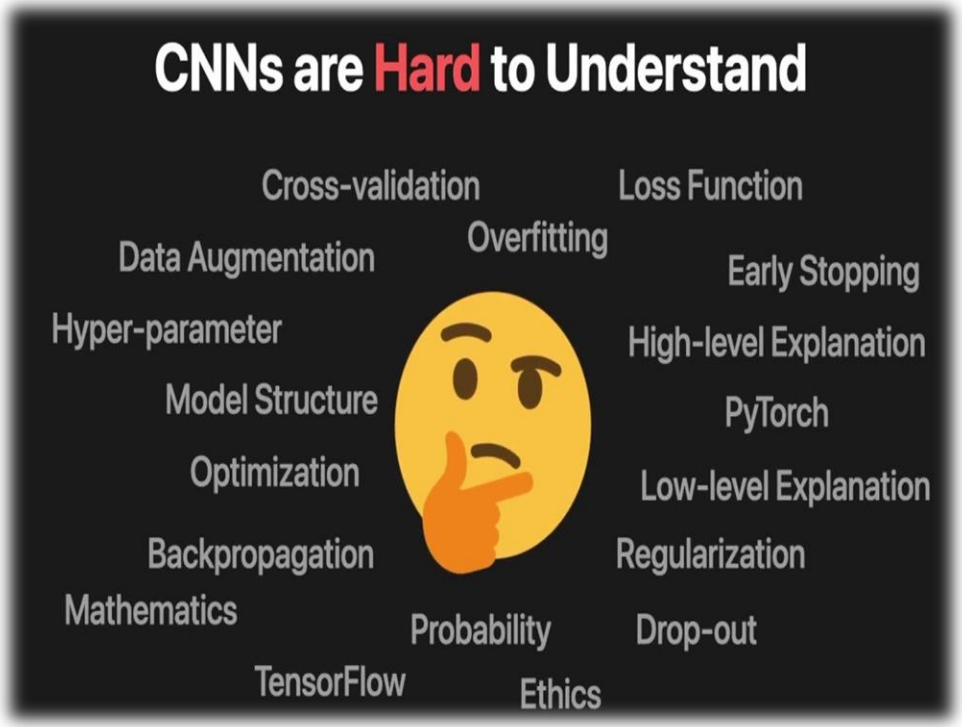


What we know before make CNN?

- Convolution Layers
- Pooling layers
  - Max pooling
  - Min pooling
  - Average pooling
- ReLu Layer
- Fully-connected (FC)
- Dense Layers

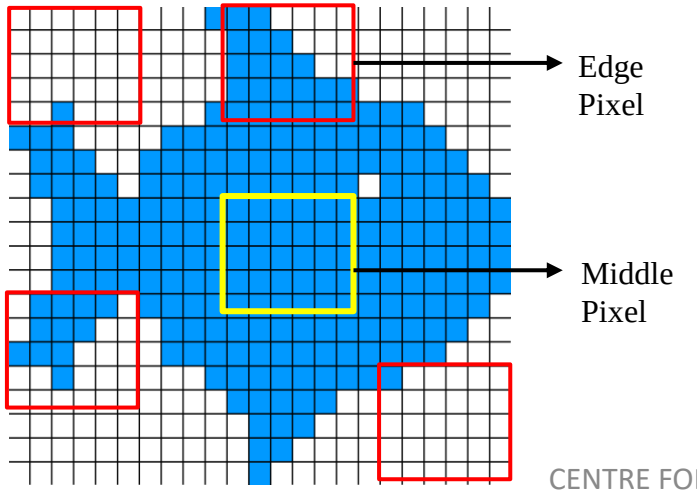
- Dropout Layers
- Batch Normalization
- Regularization
  - L1 & L2

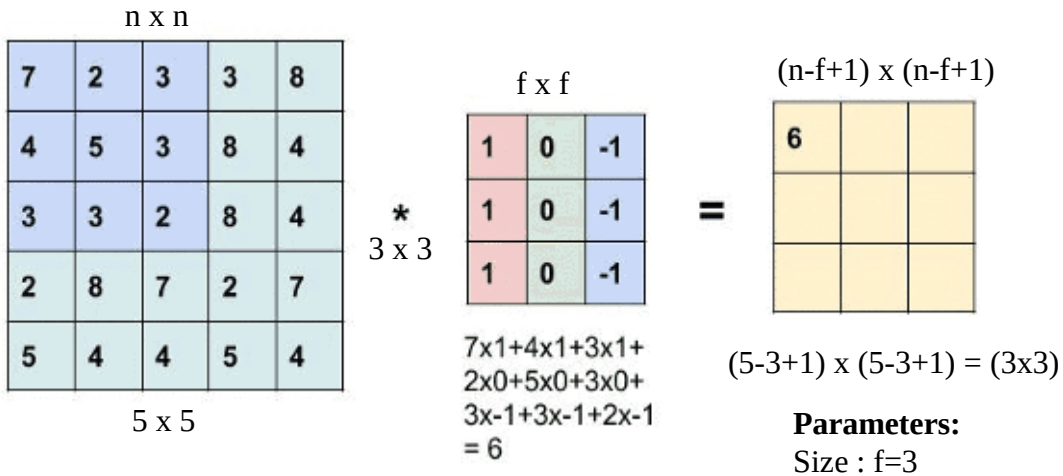
- Padding
  - Valid
  - Same
- Stride
- Filters

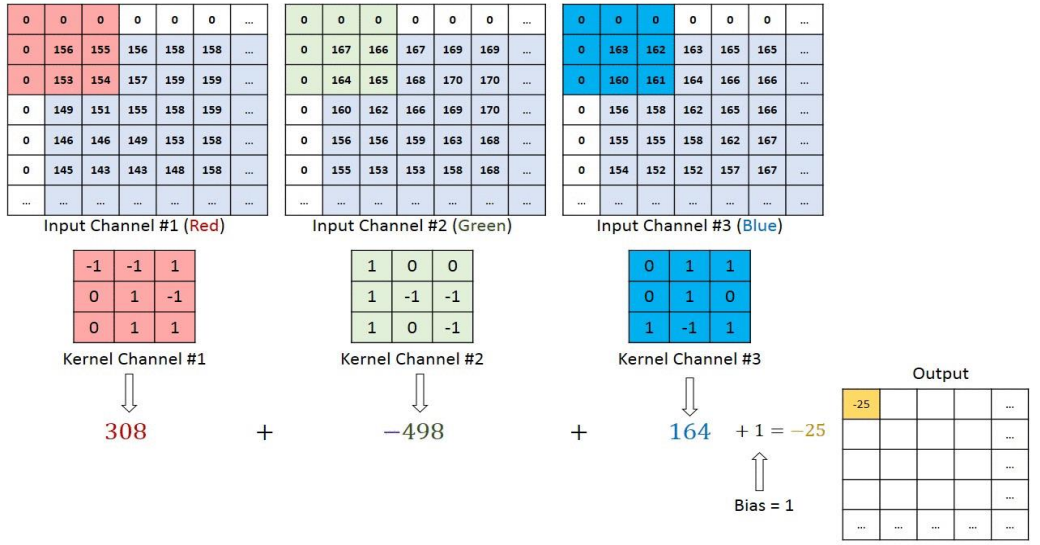


CONVOLUTION LAYER

- First layer used to extract various features from input images
- Mathematical operation performed between the input images and filter of a particular size.
- Dot product is taken between the images & filter.
- The output is termed the feature map.
- Feature map gives information about the image such as the corners and edges,
- Later, the features map is fed to other layers to learn several other features of the input images.
- Here image shrinks every time a convolution operation is performed
- Information on the border is not preserved here.







PADDING LAYER

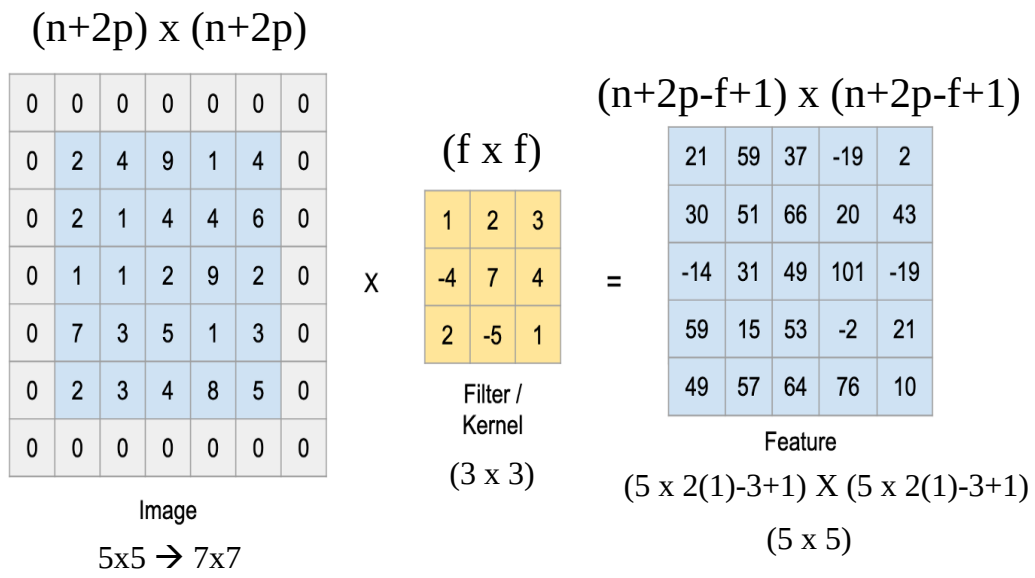
- Padding is simply a process of adding layers of zeros to our input images so as to avoid the problem in the convolution layer.
- Here the information on the borders is preserved as well as the information in the middle of the images.
- This prevents shrinking as, if p = no. of layers of zero added to the borders of the image.

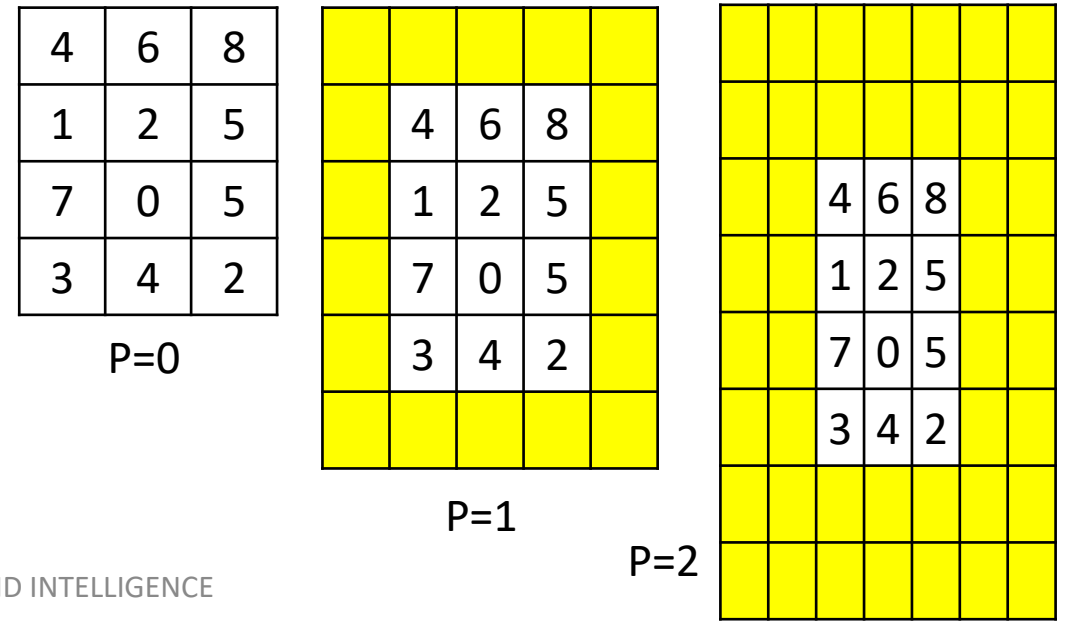
There are two padding names as:

- Valid: It implies no padding at all
- Same: Add "P" padding layers such that the output image has the same dimensions as the input image.

If we use

- (3x3) filter – 1 layer of zero must be added to the borders for the same padding.
- (5x5) filter – 2 layers of zero must be added and so on.



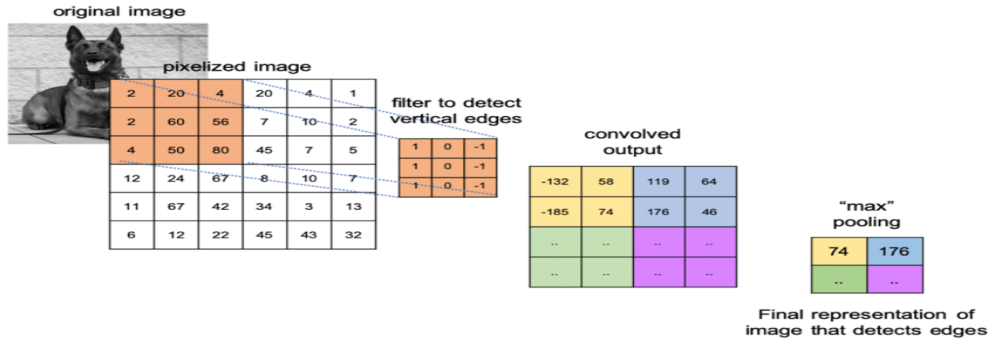


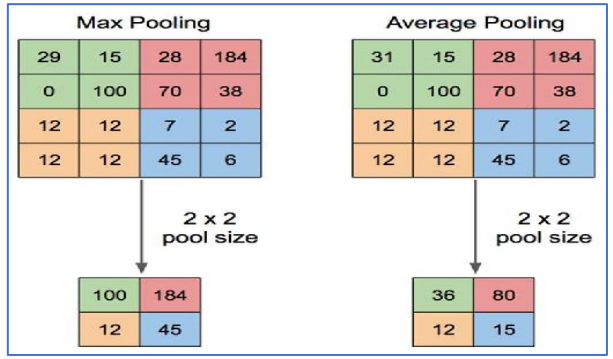
POOLING LAYER

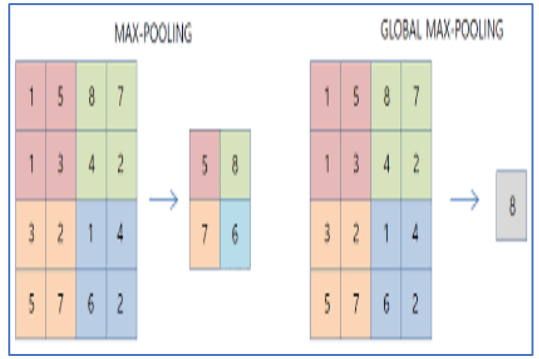
- The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.
- It is used to reduce the dimension of the feature maps.
- It reduces the no. of parameters to learn and the amount of computation performed in the network

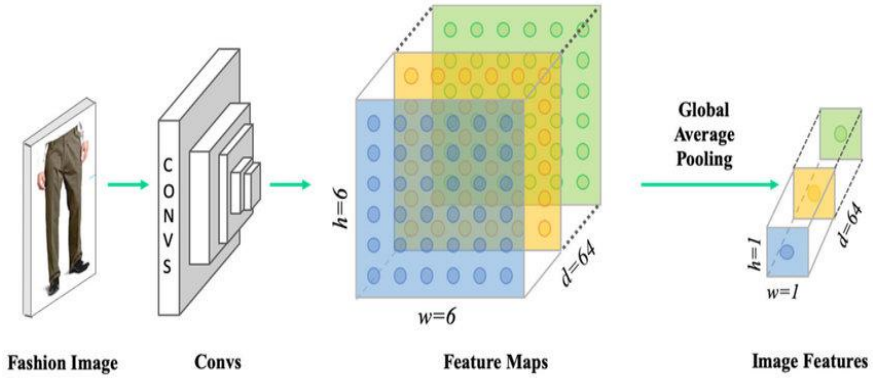
Type of pooling

- Max pooling
  - Select a maximum element and it contains the most prominent features of the previous feature map
- Average pooling
  - Average of the element present in the region of the feature map covered by the filter.
- Global pooling
  - It reduces each channel in the feature map to a single value.



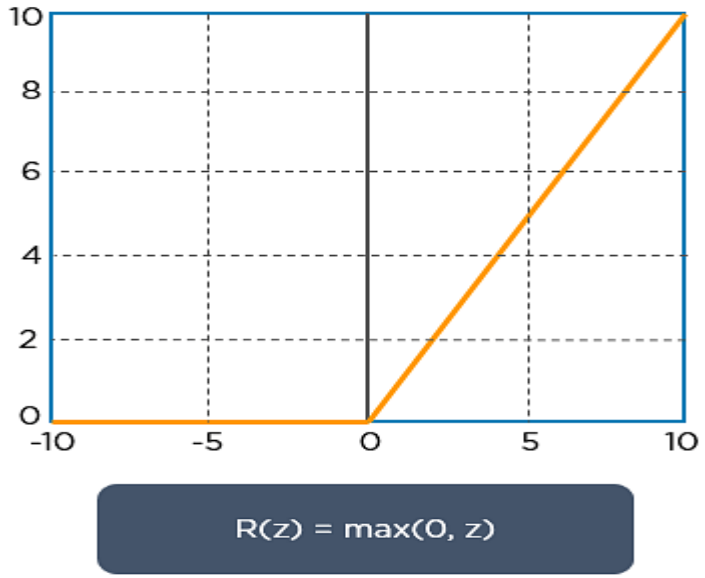


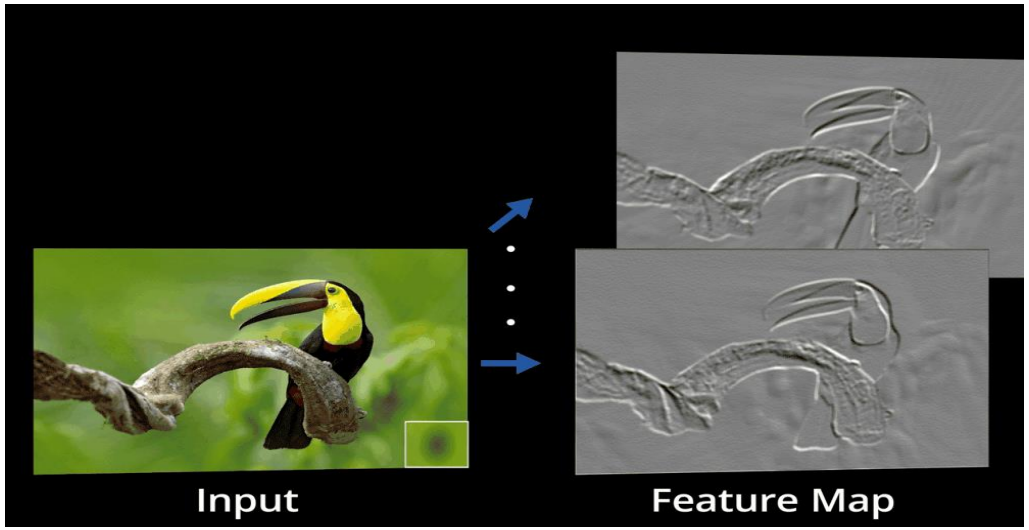


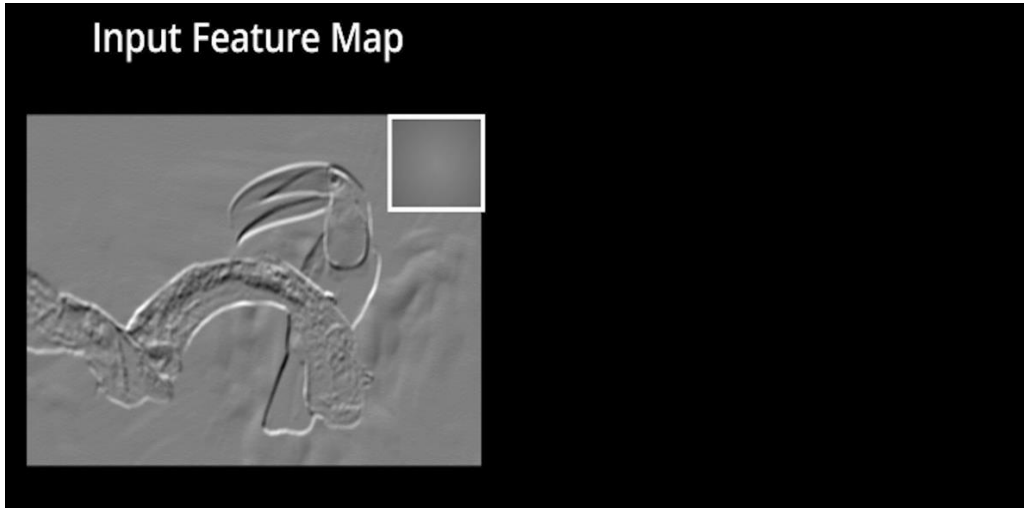


ReLU (Activation) Layer

- ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.
- ReLU performs an element-wise operation and sets all the negative pixels to 0.
- It introduces non-linearity to the network, and the generated output is a rectified feature map.







STRIDE & FILTERS

- Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.
- Filters detect spatial patterns such as edges in an image by detecting the changes in intensity values of the images.
- Each filter in this layer is randomly initialized to some distribution (Normal, Gaussian, etc.)
- Random initialization ensures that each filter learns to identify different features.

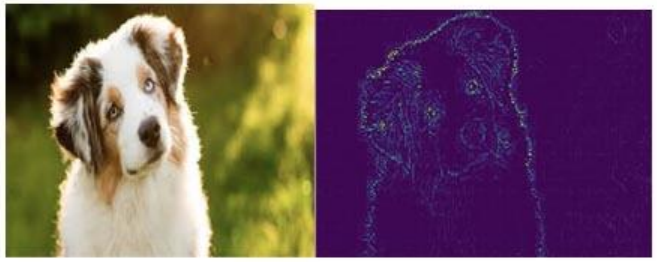
Filter Dimension: (3x3), (5x5), (7x7), (11x11)

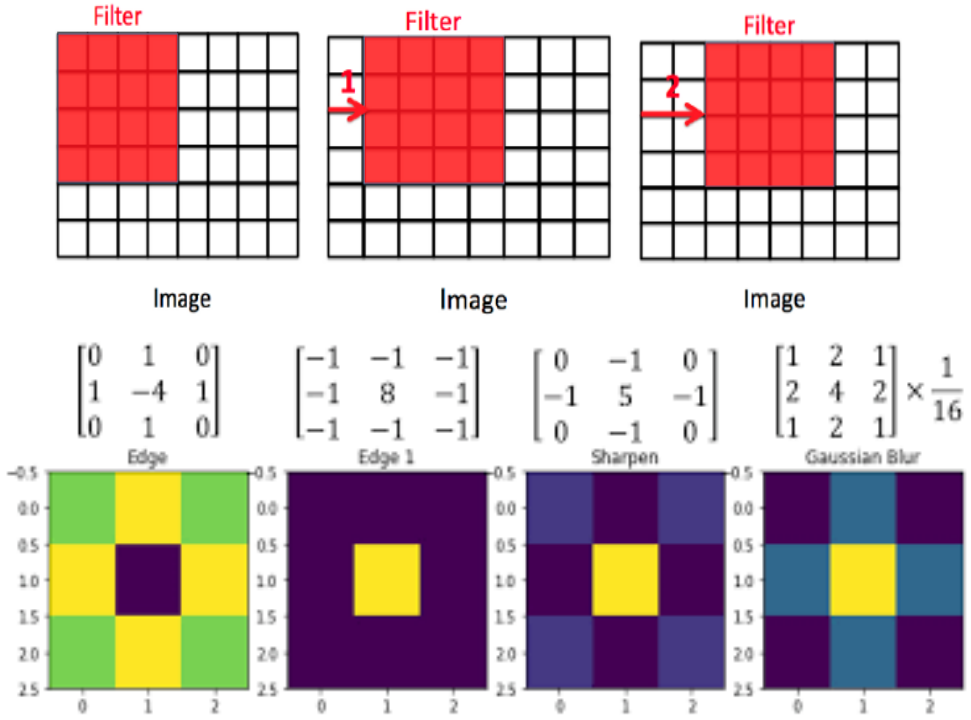
-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

Vertical





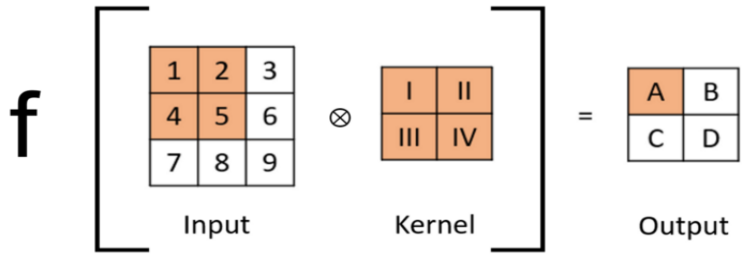
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

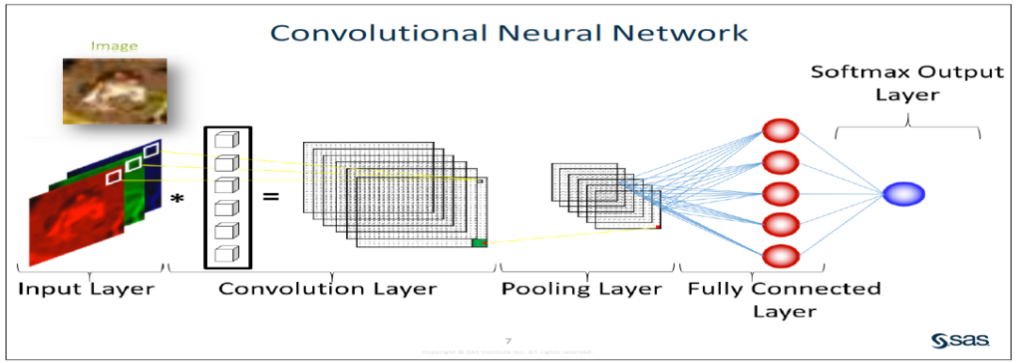
FULLY CONNECTED LAYER

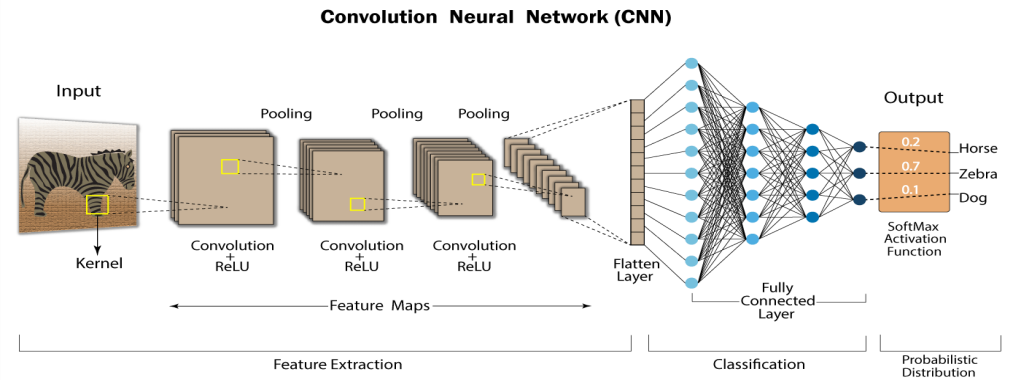
- **It is a feed-forward neural network**.
- Fully Connected Layers form the last few layers in the network.
- The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.
- A fully connected layer **multiplies the input by a weight matrix and then adds a bias vector**.

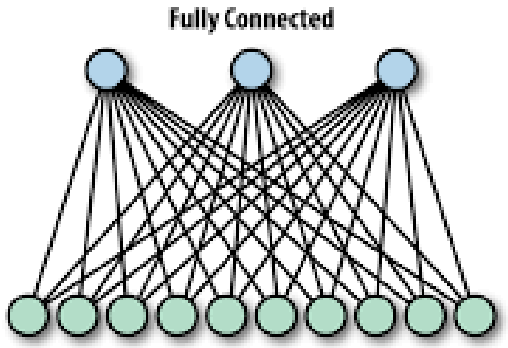
Why do we use a fully conned layer in CNN?

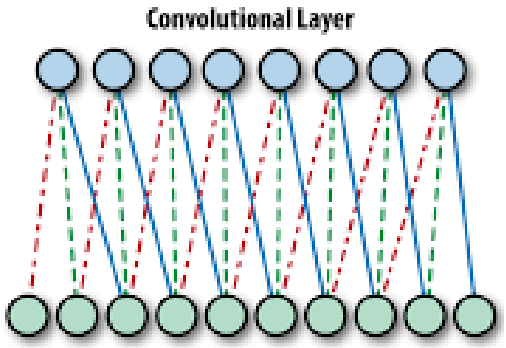
- Classification: After feature extraction, we need to classify the data into various classes, this can be done using a fully connected (FC) neural network.





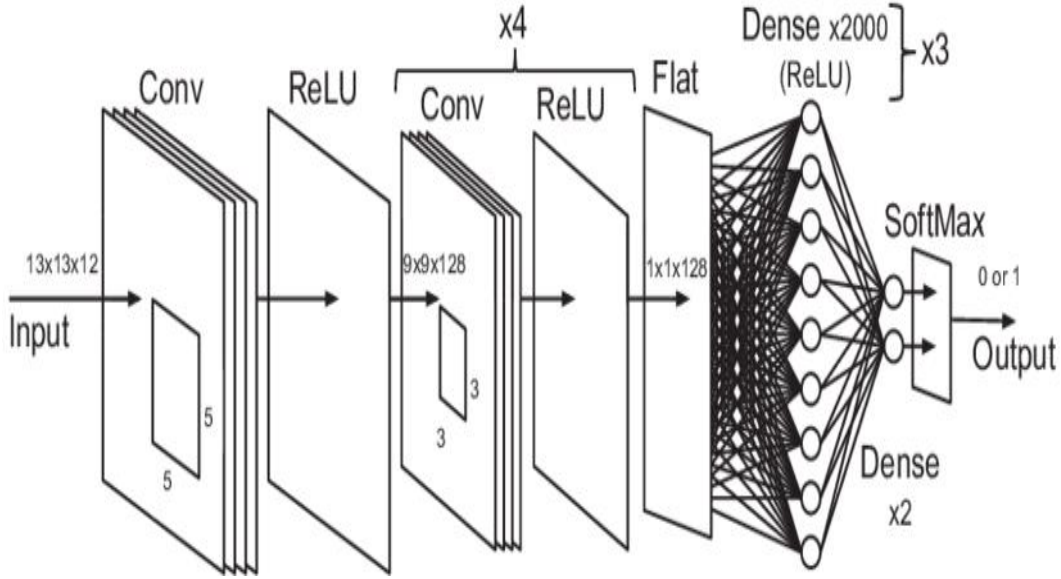


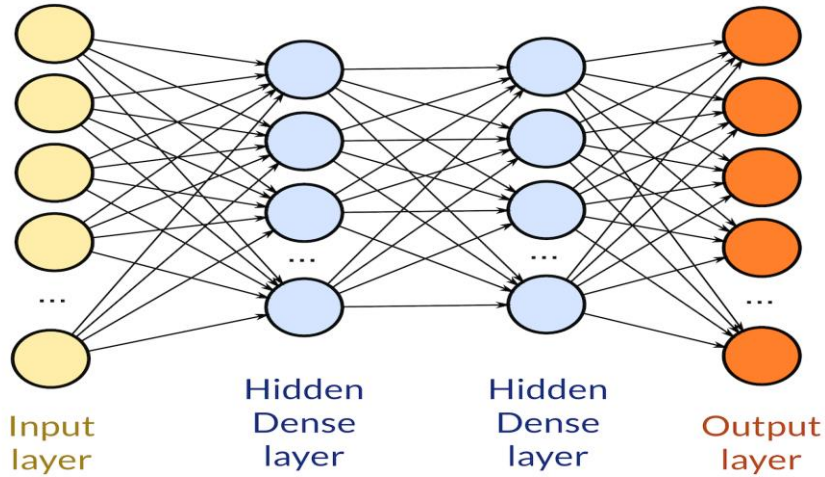




DENSE LAYER

- Dense Layer is a simple layer of neurons in which each neuron receives input from all the neurons of the previous layer, thus called as dense.
- which means the neurons of the layer are connected to every neuron of its preceding layer.
- Dense Layer is used to classify images based on output from convolutional layers.
- Dense layer is basically used for changing the dimensions of the vector.
- Dense layers also apply operations like rotation, scaling, translation on the vector.





REGULARIZATION

- Model performed exceptionally well on train data but was not able to predict test data is called overfitting.
- Regularization is a technique that makes slight modifications to the learning algorithm such that the model generalizes better.
- Improves the model's performance on the unseen data as well.
- The weights then end up having less impact on the loss function which determines the error between the actual label and predicted label.

Techniques:

- Dropout
- Batch normalization
- Data augmentation
- Early stopping

Example:

- In this example, the images have certain features that help the model identify it as a cat, like a cat's whisker, ears, eyes, etc. Each feature is assigned a certain weight. If there are a lot of features then there will be a large number of weights, which will make the model prone to overfitting. So regularization reduces the burden on these weights.

Training images

Testing images

L1 & L2 REGULARIZATION & EARLY STOPPING

- It updates the general cost function by adding another term known as the regularization term.
- The values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models.

In L2 we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

- L2 regularization is also known as *weight decay* as it forces the weights to decay towards zero (but not exactly zero).

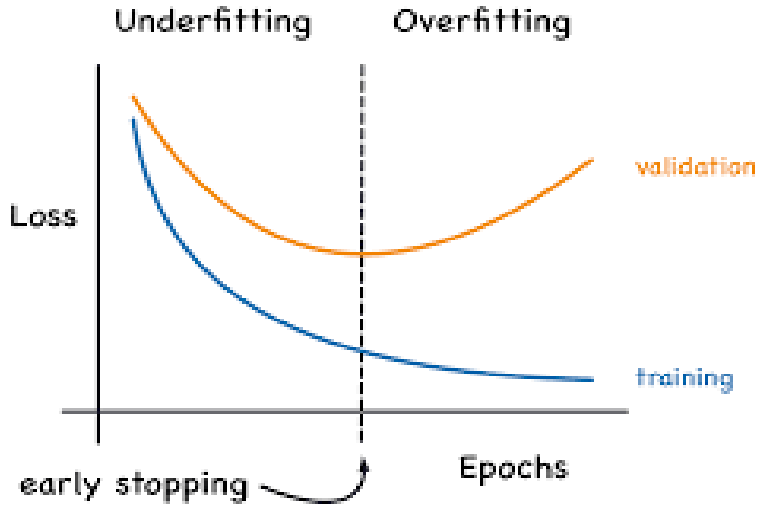
In L1 we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

- Penalize the absolute value of the weights. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.

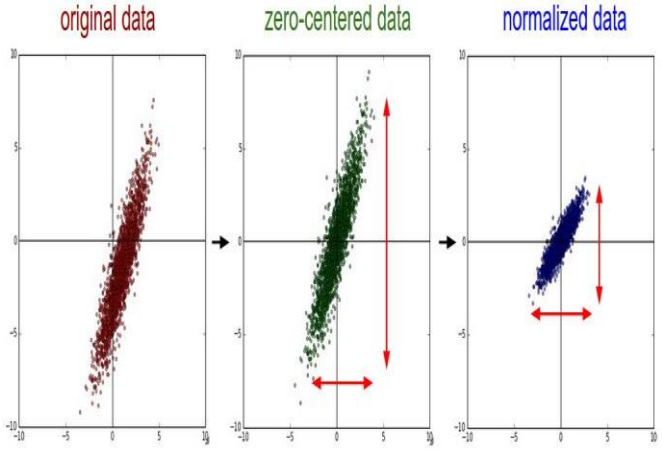
Early Stopping

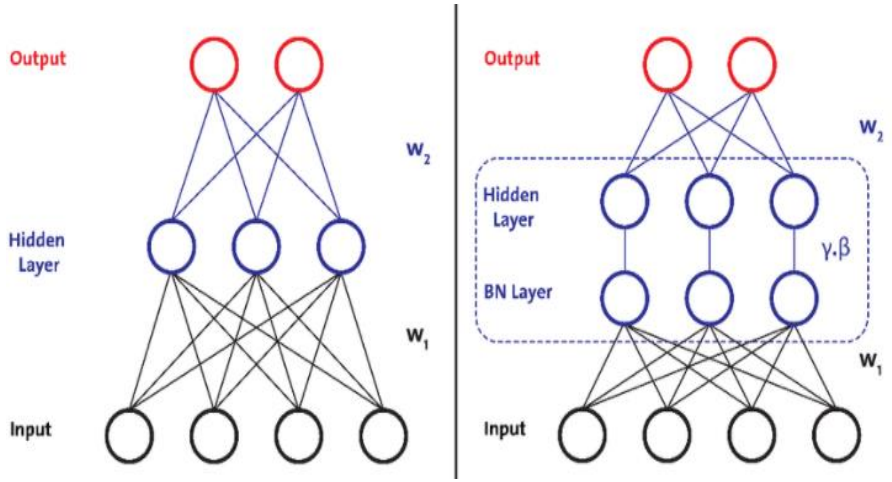
- Early stopping monitoring the DL process of the network from overfitting.
- If there is no more improvement, or worse, the performance on the test degrades, then the learning process is aborted.

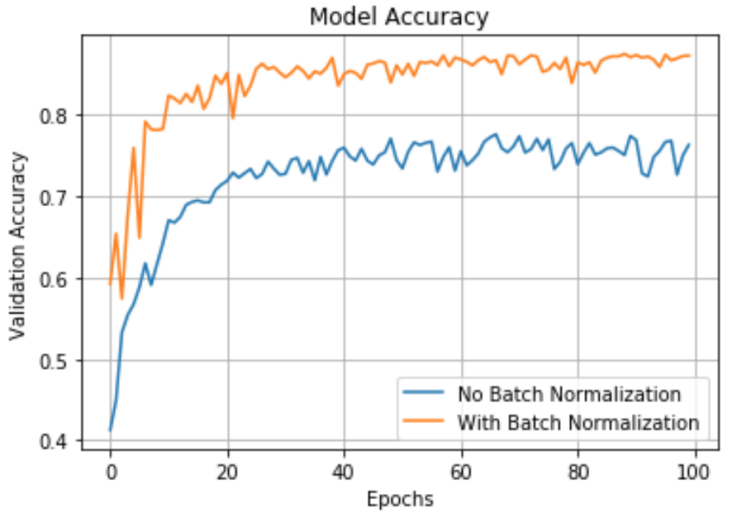


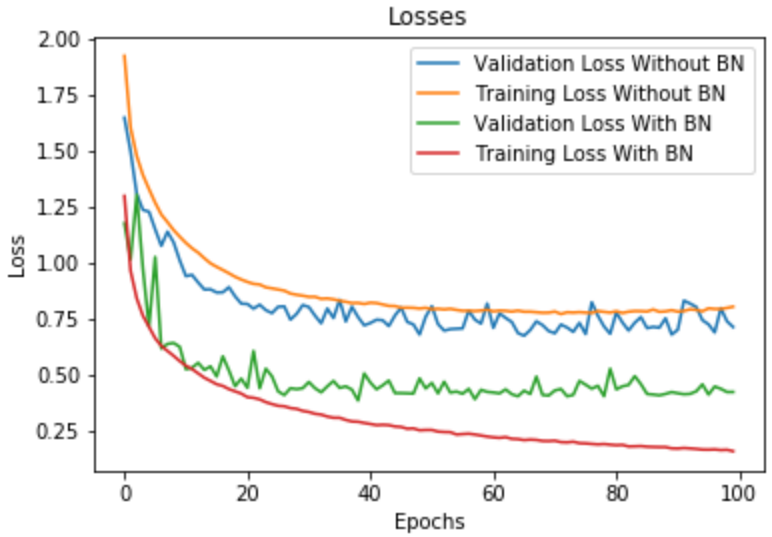
BATCH NORMALIZATION

- It allows every layer of the network to do learning more independently.
- It is a process to make neural networks faster and more stable by adding extra layers in a deep neural network.
- Batch normalization attempts to normalize a batch of inputs before they are fed to a non-linear activation unit (like ReLU, sigmoid, etc.,) during training.
- The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.
- It is often placed just after defining the sequential model and after the convolution and pooling layers.



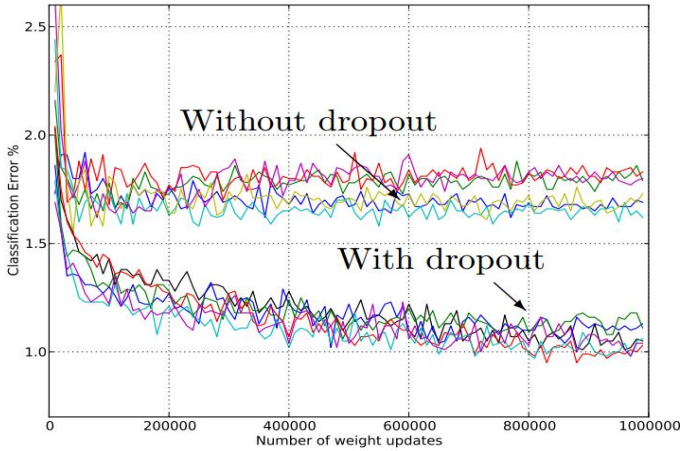


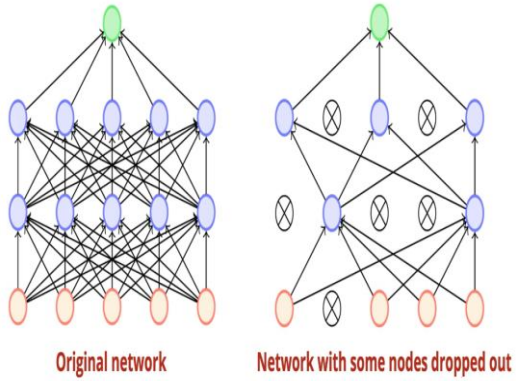


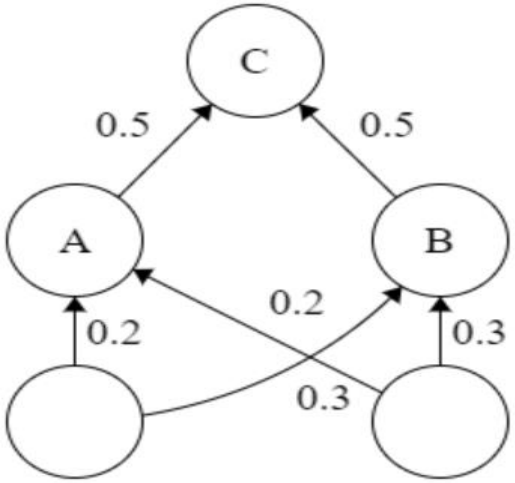


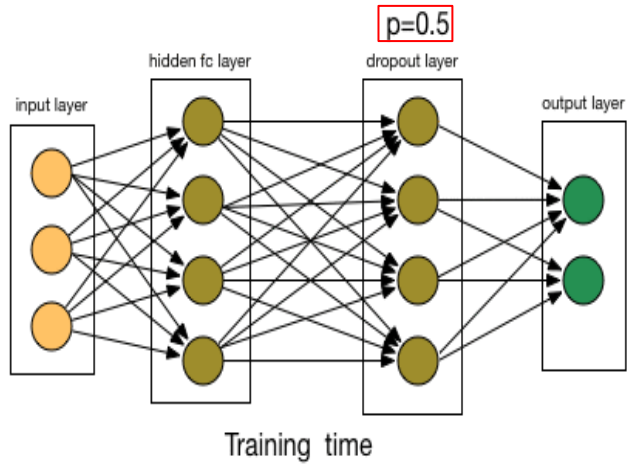
DROPOUT LAYER

- It prevents the overfitting of the training data.
- This can happen when the connection weights for two different neurons are nearly identical.
- Dropouts are added to randomly switching some percentage of neurons of the network.
- When the neurons are switched off the incoming and outgoing connection to those neurons is also switched off.
- It enhances the learning of the model.
- Dropouts are usually advised not to use after the convolution layers, they are mostly used after the dense layers of the network.









VISUALIZING OF EACH LAYER IN CNN

```
from keras import models
from keras import layers

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation ='relu',
                        input_shape =(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation ='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation ='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation ='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation ='relu'))
model.add(layers.Dense(1, activation ="sigmoid"))

model.summary()
```

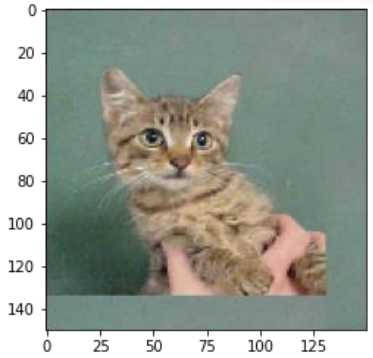
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3, 453, 121
Trainable params: 3, 453, 121
Non-trainable params: 0
```

CALCULATING THE NUMBER OF PARAMETERS IN CNN

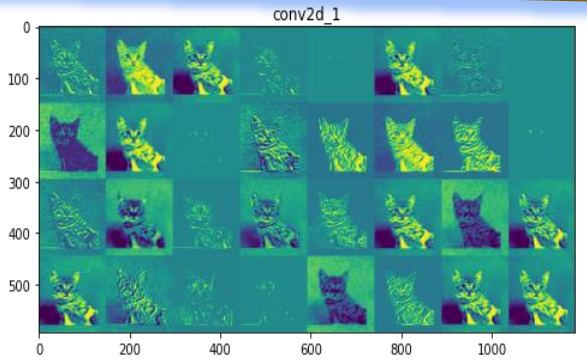
SL.No	Activation Shape	Activation Size	#parameters	Calculation
Input layer	(150x150 *3)	67,500	0	0
Conv1 (f=3,s=1)	(148x148*32)	7,00,928	896	(((3*3*3)+1)*32)
Pool1	(74x74*32)	1,75,232	0	0
Conv2(f=3,s=1)	(72x72*64)	3,31,776	18,496	(((3*3*32)+1)*64)
Pool2	(36x36*64)	82,944	0	0
Conv3(f=3,s=1)	(34x34*128)	1,47,968	73856	(((3*3*64)+1)*128)
Pool3	(17x17*128)	36,992	0	0
Conv4(f=3,s=1)	(15x15*128)	28,800	147584	(((3*3*128)+1)*128)
pool4	(7x7*128)	6,272	0	0
FC1	(6272,1)	6272	3,211,776	6272*512
FC2	(512,1)	512	3,211,776	6272*512
Sigmoid	(1,1)	2	2	514

Total params: 3,453,121 Trainable params:3,453,121 Non Trainable params:0

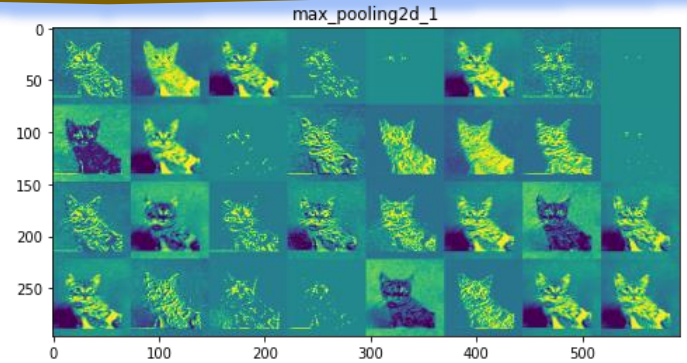
VISUALIZING OF EACH LAYER IN CNN



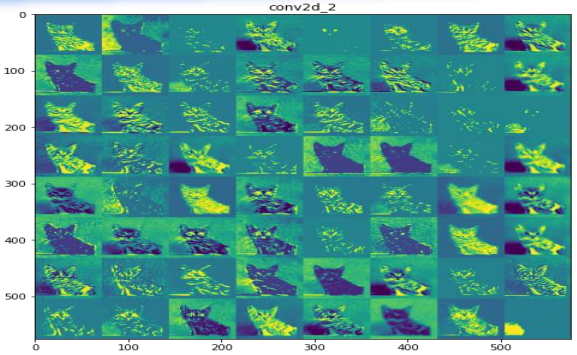
Input



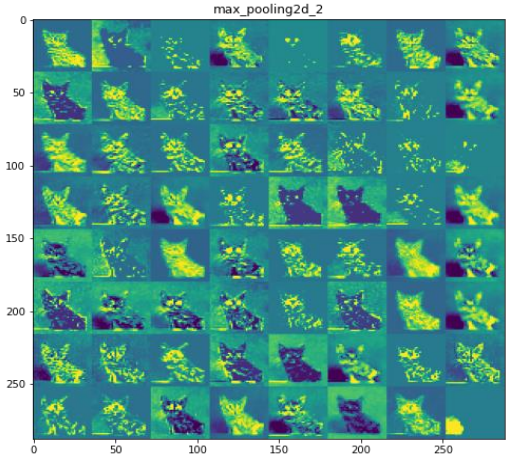
Conv2d_1



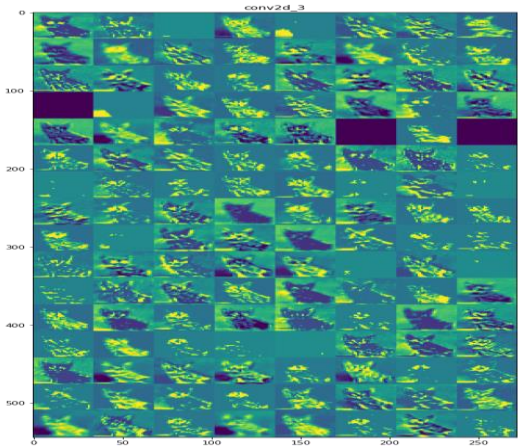
max_pooling2d_1



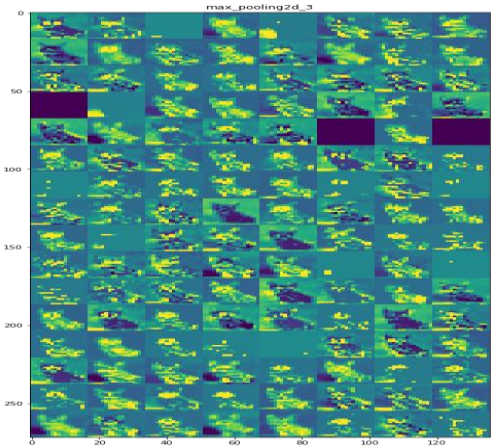
Conv2d_2



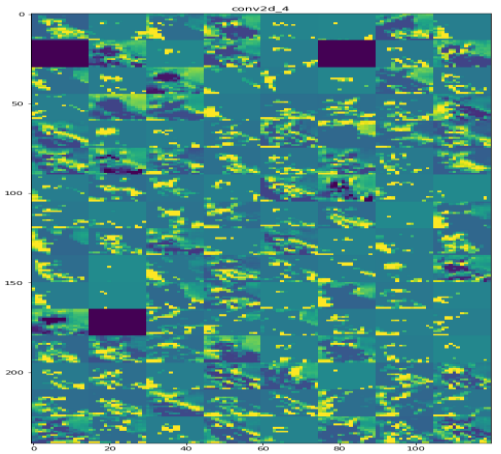
max_pooling2d_2



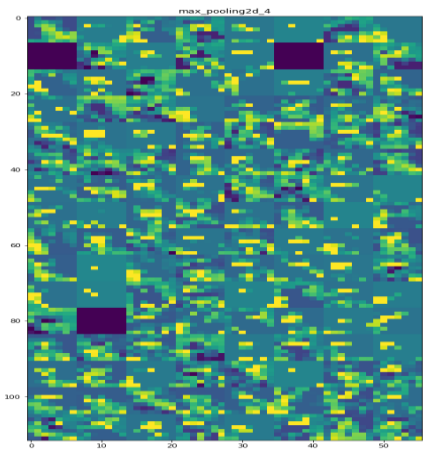
Conv2d_3



max_pooling2d_3



Conv2d_4



max_pooling2d_4

Note: Initial layers are more interpretable and retain the majority of the features in the input image. As the level of the layer increases, features become less interpretable, they become more abstract and they identify features specific to the class leaving behind the general features of the image.

CNN MODEL TRAINING SUMMARY RESULTS

Good Model

```
Epoch 1/10
300/300 [==============================] - 5s 14ms/step - loss: 0.0253 - accuracy: 0.9935 - val_loss: 0.0725 - val_accuracy: 0.9797
Epoch 2/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0179 - accuracy: 0.9958 - val_loss: 0.0791 - val_accuracy: 0.9783
Epoch 3/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0158 - accuracy: 0.9964 - val_loss: 0.0690 - val_accuracy: 0.9807
Epoch 4/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0129 - accuracy: 0.9969 - val_loss: 0.0769 - val_accuracy: 0.9783
Epoch 5/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0107 - accuracy: 0.9976 - val_loss: 0.0713 - val_accuracy: 0.9806
Epoch 6/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0071 - accuracy: 0.9987 - val_loss: 0.0794 - val_accuracy: 0.9791
Epoch 7/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0070 - accuracy: 0.9986 - val_loss: 0.0775 - val_accuracy: 0.9782
Epoch 8/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0058 - accuracy: 0.9989 - val_loss: 0.0759 - val_accuracy: 0.9809
Epoch 9/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0047 - accuracy: 0.9990 - val_loss: 0.0772 - val_accuracy: 0.9803
Epoch 10/10
300/300 [==============================] - 4s 13ms/step - loss: 0.0028 - accuracy: 0.9996 - val_loss: 0.0839 - val_accuracy: 0.9793
313/313 [==============================] - 1s 3ms/step - loss: 0.0839 - accuracy: 0.9793
Baseline Error: 2.07%
```

Overfitting

```
Epoch 1/10
7125/7125 [==============================] - 453s 64ms/step - loss: 0.6933 - acc: 0.4962 - val_loss: 0.6935 - val_acc: 0.4962
Epoch 2/10
7125/7125 [==============================] - 469s 66ms/step - loss: 0.6847 - acc: 0.5453 - val_loss: 0.7055 - val_acc: 0.5087
Epoch 3/10
7125/7125 [==============================] - 456s 64ms/step - loss: 0.6456 - acc: 0.6069 - val_loss: 0.7622 - val_acc: 0.5043
Epoch 4/10
7125/7125 [==============================] - 467s 66ms/step - loss: 0.5993 - acc: 0.6511 - val_loss: 0.8611 - val_acc: 0.5061
Epoch 5/10
7125/7125 [==============================] - 424s 60ms/step - loss: 0.5539 - acc: 0.6872 - val_loss: 0.9861 - val_acc: 0.5069
Epoch 6/10
7125/7125 [==============================] - 425s 60ms/step - loss: 0.5110 - acc: 0.7196 - val_loss: 1.1221 - val_acc: 0.5043
Epoch 7/10
7125/7125 [==============================] - 481s 67ms/step - loss: 0.4681 - acc: 0.7480 - val_loss: 1.2454 - val_acc: 0.5024
Epoch 8/10
7125/7125 [==============================] - 451s 63ms/step - loss: 0.4303 - acc: 0.7719 - val_loss: 1.3952 - val_acc: 0.5008
Epoch 9/10
7125/7125 [==============================] - 509s 71ms/step - loss: 0.3961 - acc: 0.7921 - val_loss: 1.4681 - val_acc: 0.5023
Epoch 10/10
7125/7125 [==============================] - 461s 65ms/step - loss: 0.3721 - acc: 0.8067 - val_loss: 1.4955 - val_acc: 0.4998
```

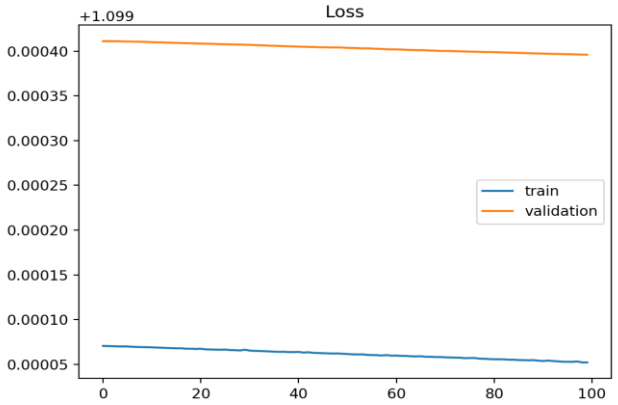
Underfitting

```
Epoch 1/100
194/194 [==============================] - 3492s 18s/step - loss: 3.0128 - accuracy: 0.2934 - val_loss: 1.7569 - val_accuracy: 0.1465
Epoch 2/100
194/194 [==============================] - 146s 753ms/step - loss: 2.8243 - accuracy: 0.3277 - val_loss: 0.9421 - val_accuracy: 0.1469
Epoch 3/100
194/194 [==============================] - 144s 741ms/step - loss: 2.6972 - accuracy: 0.3589 - val_loss: 2.8190 - val_accuracy: 0.3778
Epoch 4/100
194/194 [==============================] - 144s 741ms/step - loss: 2.7382 - accuracy: 0.3385 - val_loss: 2.6606 - val_accuracy: 0.3971
Epoch 5/100
194/194 [==============================] - 144s 744ms/step - loss: 2.7899 - accuracy: 0.3114 - val_loss: 2.0858 - val_accuracy: 0.4015
Epoch 6/100
194/194 [==============================] - 143s 735ms/step - loss: 2.8026 - accuracy: 0.3163 - val_loss: 2.1657 - val_accuracy: 0.3956
Epoch 7/100
194/194 [==============================] - 141s 727ms/step - loss: 2.6561 - accuracy: 0.3515 - val_loss: 2.6220 - val_accuracy: 0.4618
Epoch 8/100
194/194 [==============================] - 140s 722ms/step - loss: 2.6023 - accuracy: 0.3662 - val_loss: 4.7769 - val_accuracy: 0.1476
Epoch 9/100
194/194 [==============================] - 141s 727ms/step - loss: 2.7287 - accuracy: 0.3026 - val_loss: 4.0613 - val_accuracy: 0.3215
Epoch 10/100
194/194 [==============================] - 140s 721ms/step - loss: 2.7586 - accuracy: 0.2907 - val_loss: 2.9873 - val_accuracy: 0.3691
Epoch 11/100
193/194 [============================>.] - ETA: 0s - loss: 2.7070 - accuracy: 0.2934
```

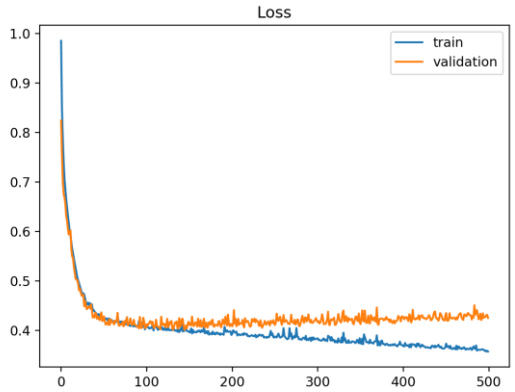
Overfitting

```
Train on 2072 samples, validate on 2072 samples
Epoch 1/10
2072/2072 [==============================] - 16s - loss: 1.1490 - acc: 0.3422 - val_loss: 1.1064 - val_acc: 0.3441
Epoch 2/10
2072/2072 [==============================] - 15s - loss: 1.1047 - acc: 0.3441 - val_loss: 1.1010 - val_acc: 0.3441
Epoch 3/10
2072/2072 [==============================] - 16s - loss: 1.1001 - acc: 0.3441 - val_loss: 1.0988 - val_acc: 0.3441
Epoch 4/10
2072/2072 [==============================] - 16s - loss: 1.0985 - acc: 0.3431 - val_loss: 1.0985 - val_acc: 0.3441
Epoch 5/10
2072/2072 [==============================] - 16s - loss: 1.0986 - acc: 0.3441 - val_loss: 1.0984 - val_acc: 0.3441
Epoch 6/10
2072/2072 [==============================] - 16s - loss: 1.0988 - acc: 0.3441 - val_loss: 1.0984 - val_acc: 0.3441
Epoch 7/10
2072/2072 [==============================] - 17s - loss: 1.0986 - acc: 0.3441 - val_loss: 1.0984 - val_acc: 0.3441
Epoch 8/10
2072/2072 [==============================] - 16s - loss: 1.0987 - acc: 0.3436 - val_loss: 1.0984 - val_acc: 0.3441
Epoch 9/10
2072/2072 [==============================] - 16s - loss: 1.0986 - acc: 0.3441 - val_loss: 1.0984 - val_acc: 0.3441
Epoch 10/10
2072/2072 [==============================] - 16s - loss: 1.0984 - acc: 0.3441 - val_loss: 1.0984 - val_acc: 0.3441
```

DIAGNOSING CNN MODEL BEHAVIOUR FROM GRAPH

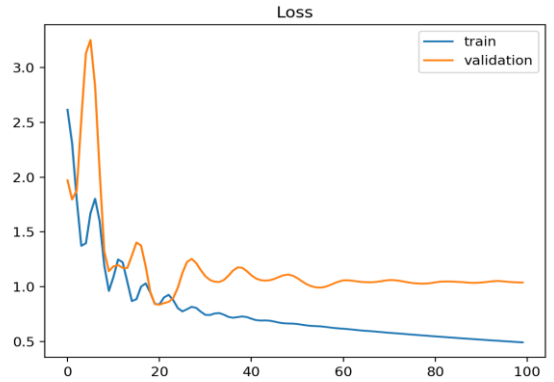


Underfit

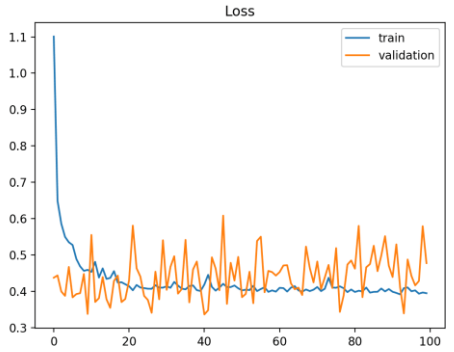


Overfit

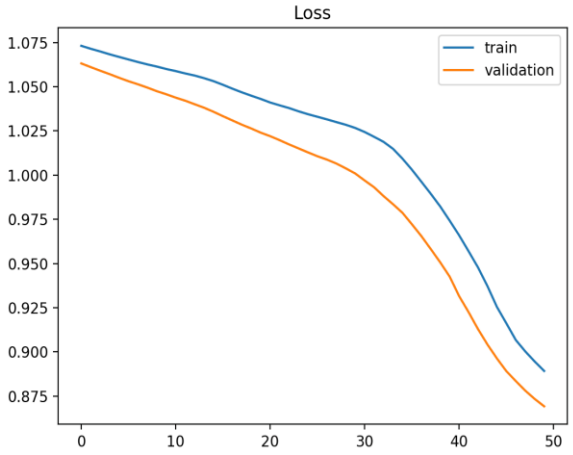
Unrepresentative Dataset



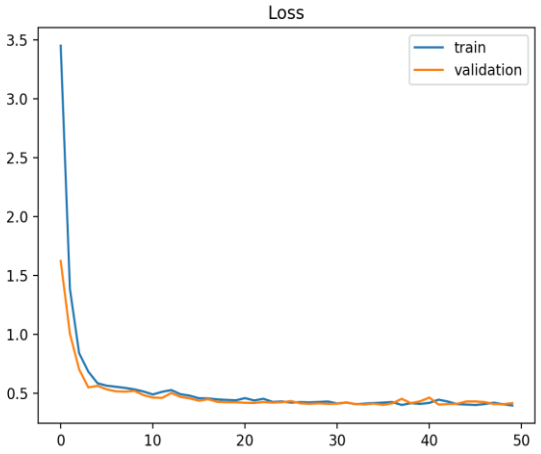
Training Dataset That May Be too Small Relative to the Validation Dataset



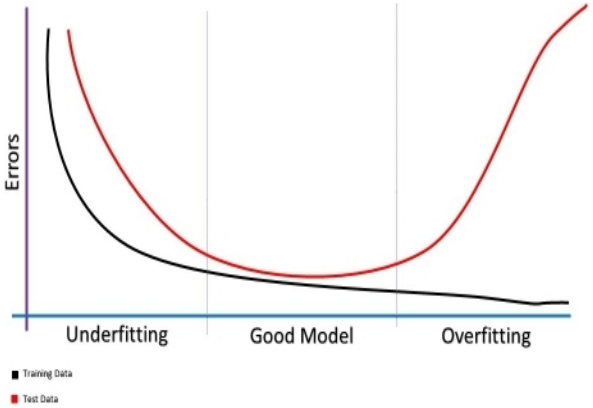
Validation Dataset That May Be too Small Relative to the Training Dataset

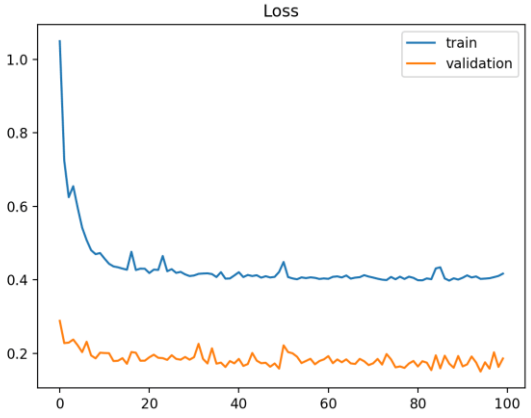


Underfit



Best fit





Validation Dataset That Is Easier to Predict Than the Training Dataset

Evaluation metrics in DL

Evaluation metrics for Classification

- Confusion matrix
- Accuracy
- Precision
- Recall=Sensitivity=True Positive Rate=Hit Rate
- Specificity=True Negative Rate=Selectivity
- F1-Score
- AUC-ROC
- Precision/Recall Curve

Evaluation metrics for Regression

- MSE
- RMSE
- MAE
- MAPE
- R^2 and adjusted R^2

Evaluation metrics for Clustering

- Silhouette Score
- Rand Index
- Adjusted Rand Index
- Mutual Information
- Calinski-Harabasz Index
- Davies-Bouldin Index

https://www.tensorflow.org/api_docs/python/tf/keras/metrics

https://keras.io/api/metrics/#:~:text=A%20metric%20is%20a%20function,used%20when%20training%20the%20model

CLASSIFICATION METRICS IN DL

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	TP	FP
Predicted Negative (0)	FN	TN

Precision NPV

Sensitivity / Recall

Specificity

n = 165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Notations,

TP: True Positive: Number of Points that are actually positive and predicted to be positive

FN: False Negative: Number of Points that are actually positive but predicted to be negative --- Type-II Error

FP: False Positive: Number of Points that are actually negative but predicted to be positive --- Type-I Error

TN: True Negative: Number of Points that are actually negative and predicted to be negative

Accuracy (ACC) = TP+TN/TP+FP+FN+TN
(number of correct predictions made to all)

Precision / PPV = TP / TP+ FP
(the proportion of positive cases that were correctly identified)

TPR (Recall/Sensitivity (SNS) / Hit Rate/ Probability of detection)
= TP / TP+ FN
(the proportion of actual positive cases which are correctly identified)

Negative Predicated value (NPV) = TN / TN+ FN
(number of correctly predicted negative classes to the total number of negative classes)

Specificity (SPC) = TN / TN+ FP
(number of correctly predicted negative classes to the total number of negative classes)

FPR (Fall-out) = FP / FP+ TN
(number of wrongly predicted positive classes to the total number of negative classes)

FNR (Miss rate) = FN / FN+ TP
(number of correctly wrongly negative classes to the total number of positive classes)

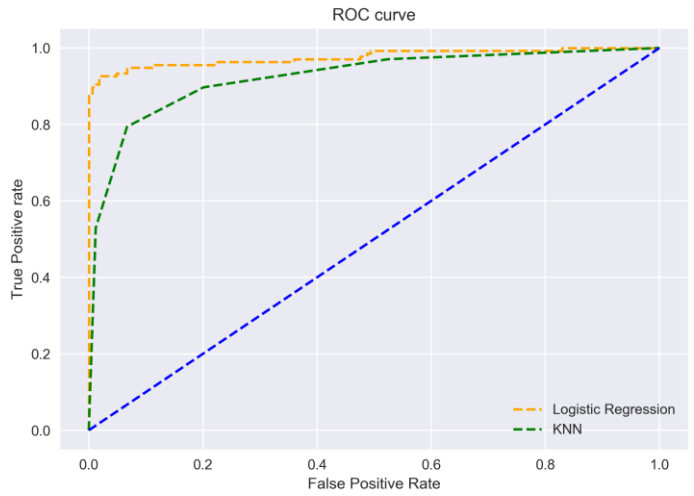
CLASSIFICATION METRICS IN DL

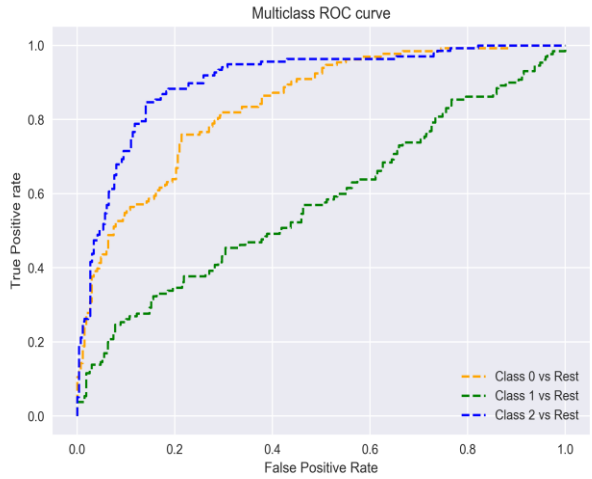
Error Rate / Misclassification Rate = FP+FN / Total Value

False Discovery Rate (FDR) = FP+FP / TP

False Omission Rate (FOR) = FN+FN / TN

F1 Score = 2* (Recall * Precision) / Recall+Precision
(hybrid Metrix for unbalanced dataset)

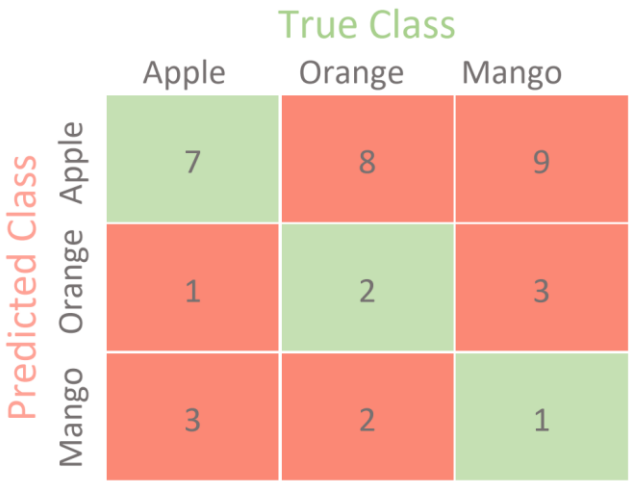




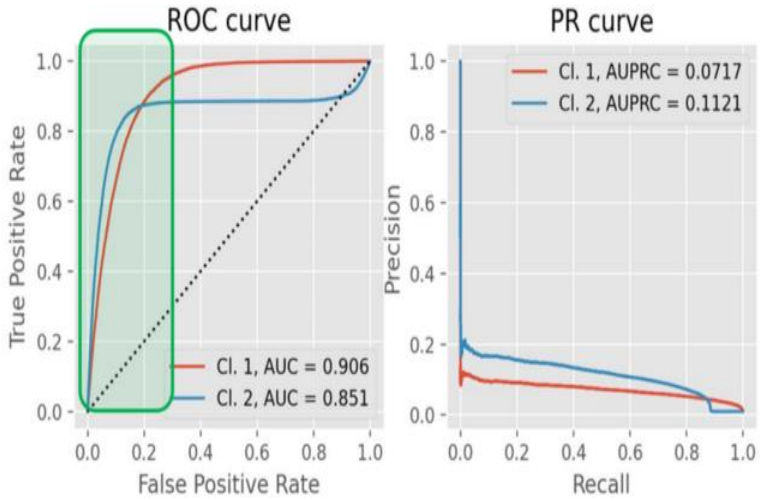
Area Under the ROC curve (AUC-ROC)

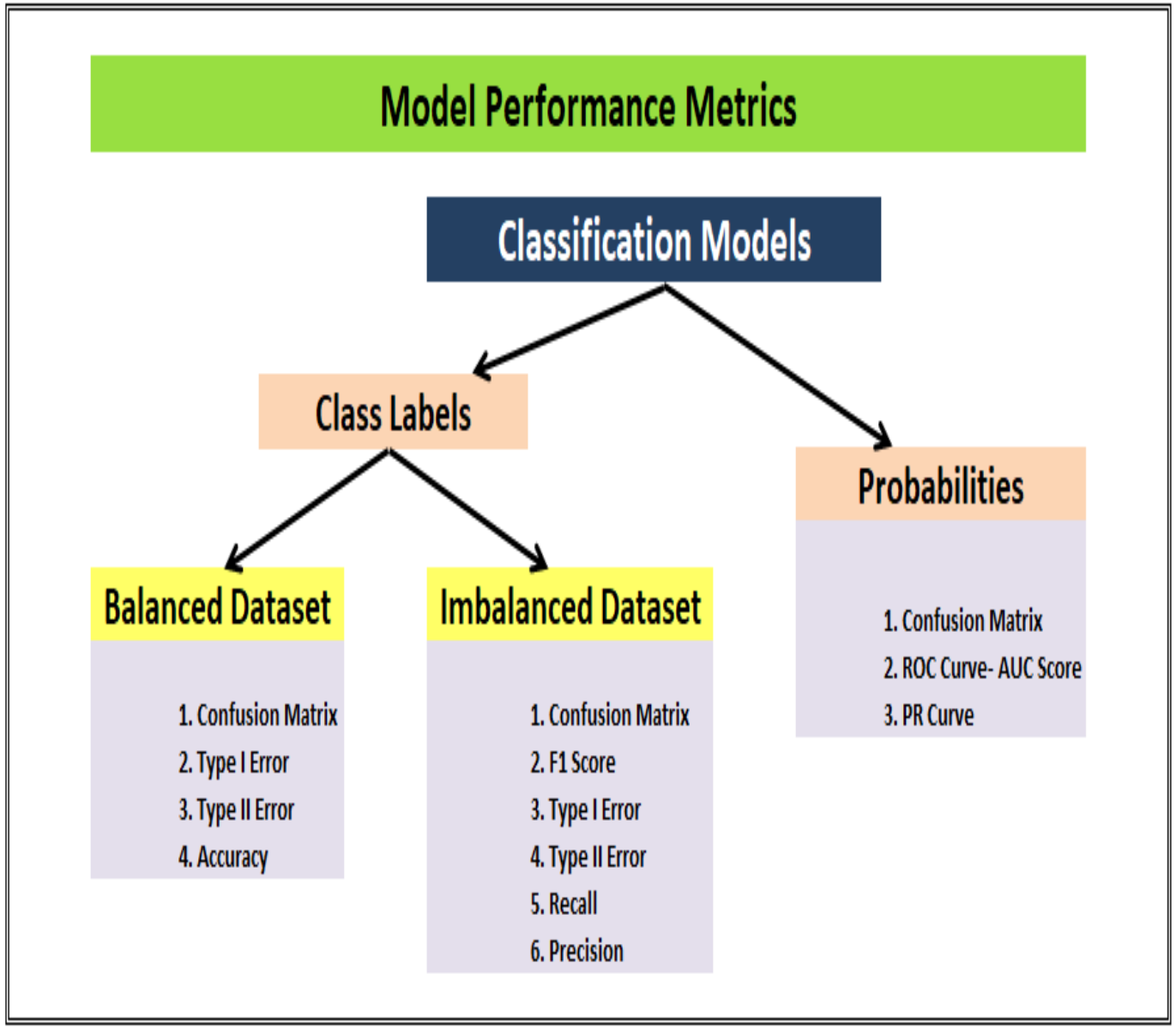
ROC – (Receiver operating characteristic)
(Predicted probability is not "properly calibrated" - plots the TPR against FPR at various threshold values

AUC – (Area Under the Curve)
(Classes are highly unbalanced - summary of the ROC curve)



Precision-Recall Curve - AUPRC



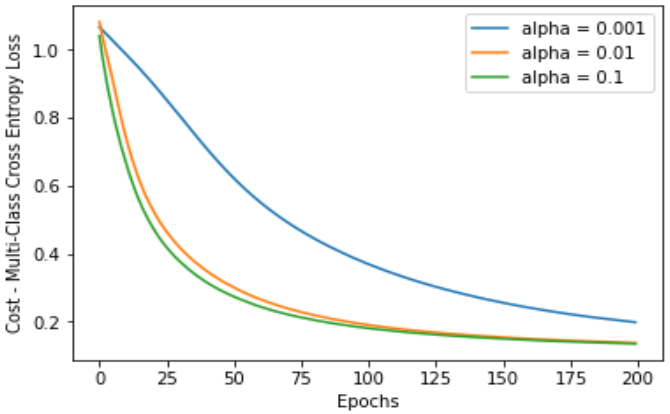


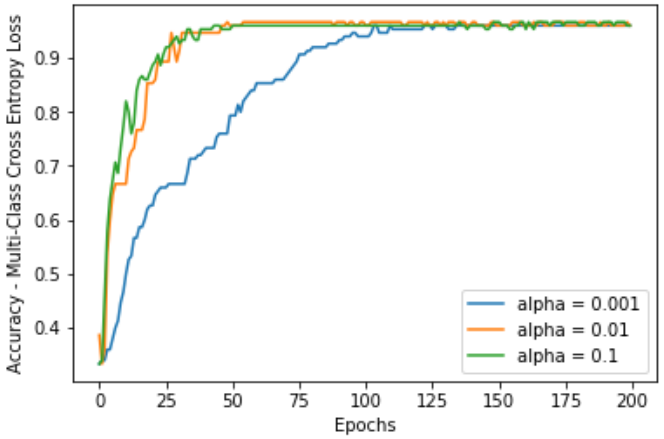
Loss Function Classification

(A loss function maps decisions to their associated costs)

Loss Function – Error Function (Single training)

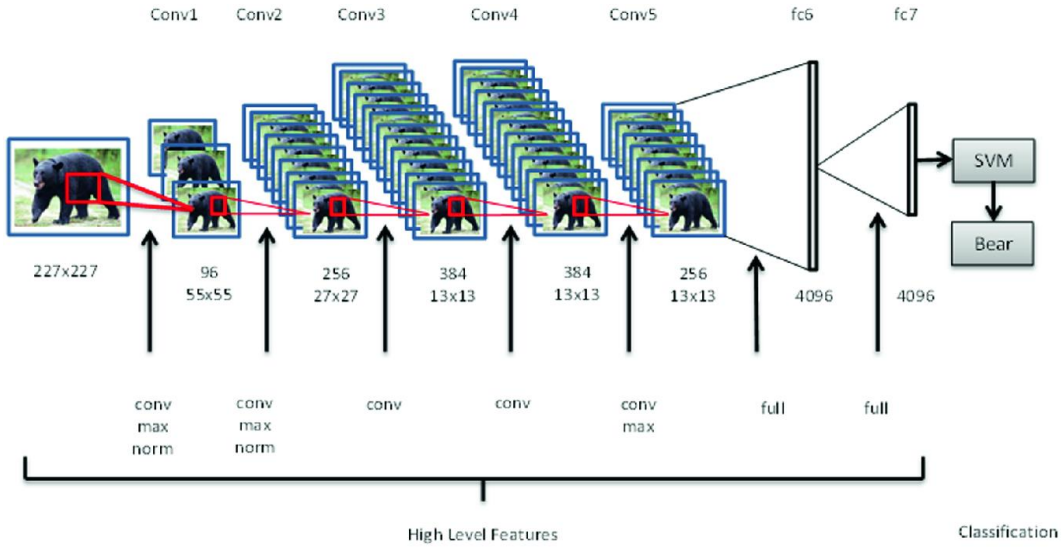
Cost Function – Average Function
(Entire Training Dataset)

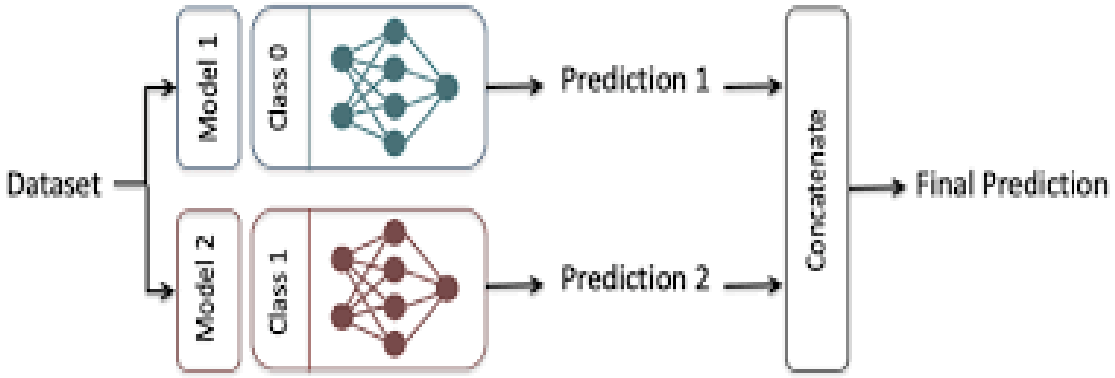


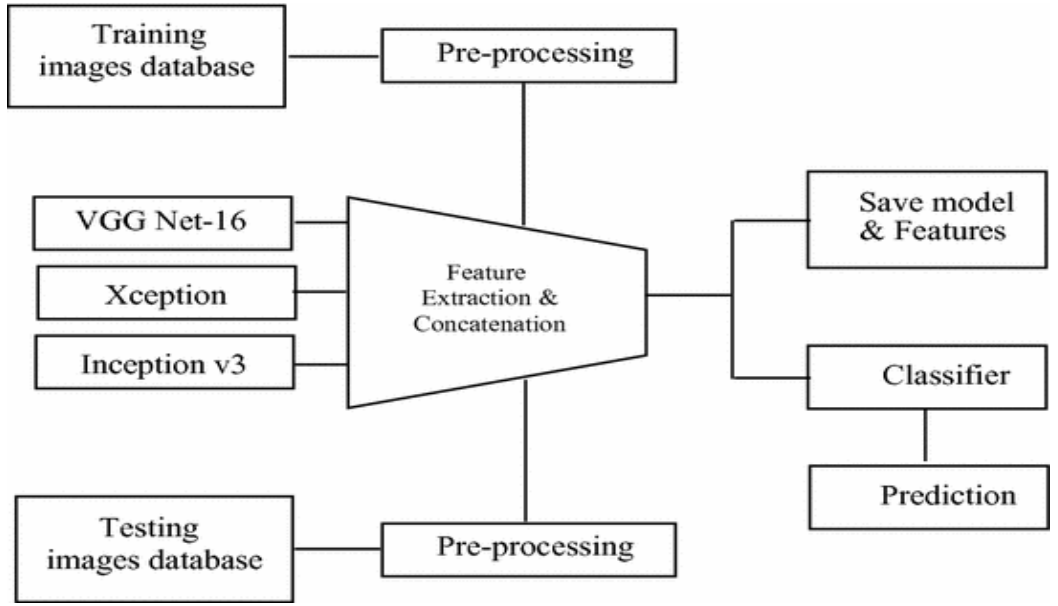


CONVNETS AS A FEATURE EXTRACTOR

- ConvNet as a feature extractor. The final classification layer is ignored and only the layer used to extract the deep features need to be defined.
- Extract the deep features and use them combined with some machine learning techniques.
- It automatically detects the important features without any human supervision



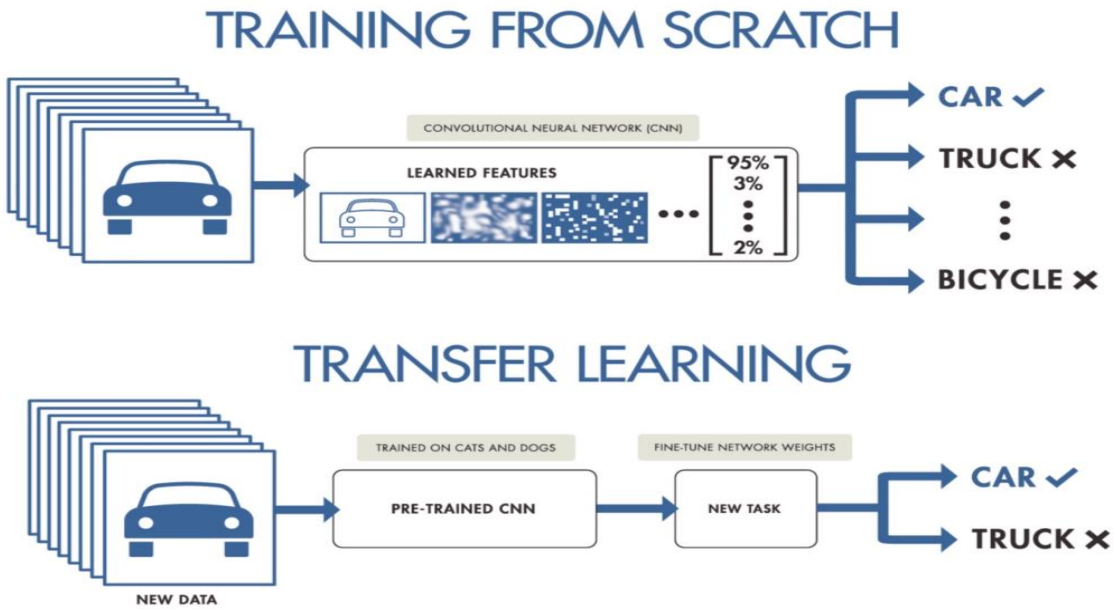


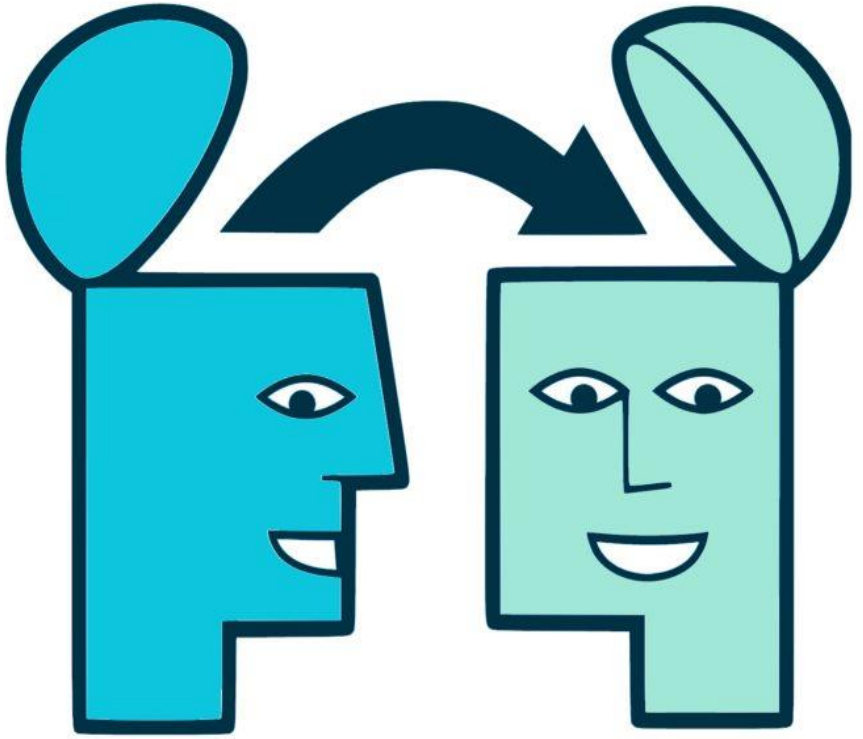


TRANSFER LEARNING IN DL

The reuse of a previously learned model on a new problem is known as transfer learning. It can train deep neural networks with a small amount of data.

- A machine uses the knowledge learned from a prior assignment to increase prediction about a new task in transfer learning.





You know how to drive two-wheeler
→ you can learn four-wheeler

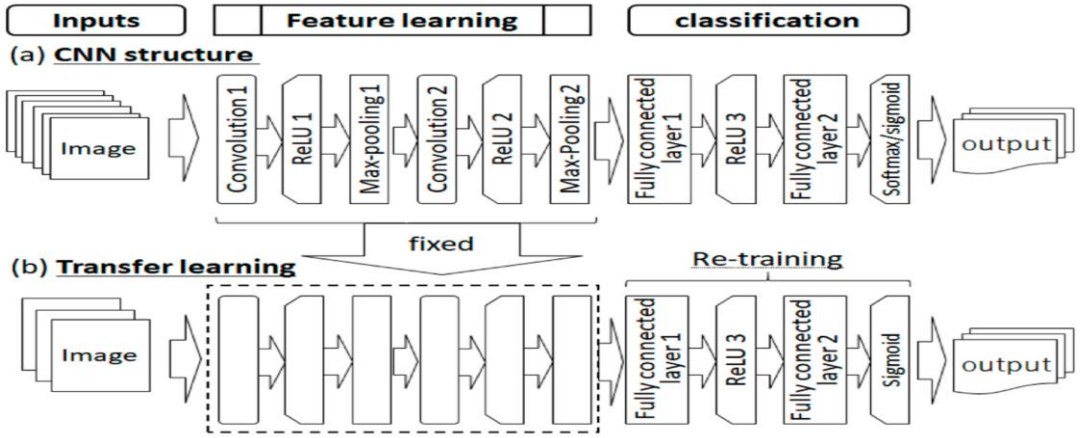
Why should we use transfer learning?

- Reduced training time
- Improved neural network performance (in most circumstances), and
- Absence of a large amount of data.

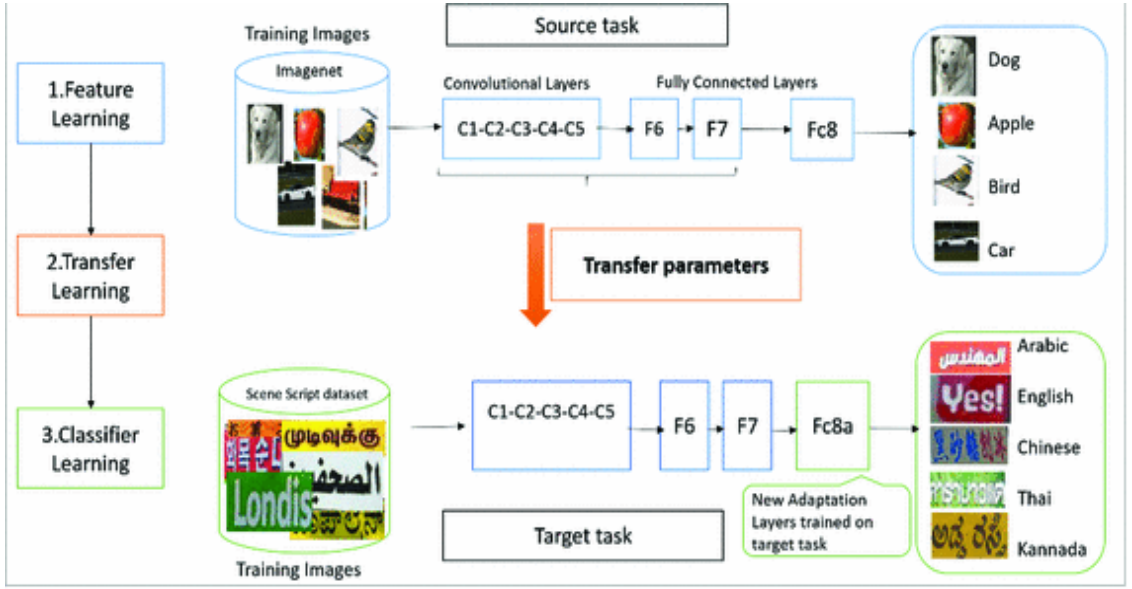
When to Use Transfer Learning?

- When we don't have enough annotated data to train our model
- When there is a pre-trained model that has been trained on similar data and tasks.

- AlexNet
- GoogLeNet
- ResNet-18
- VGG-16
- LeNet
- ResNet-34
- ResNet-50
- ResNet-101
- Inception-v3

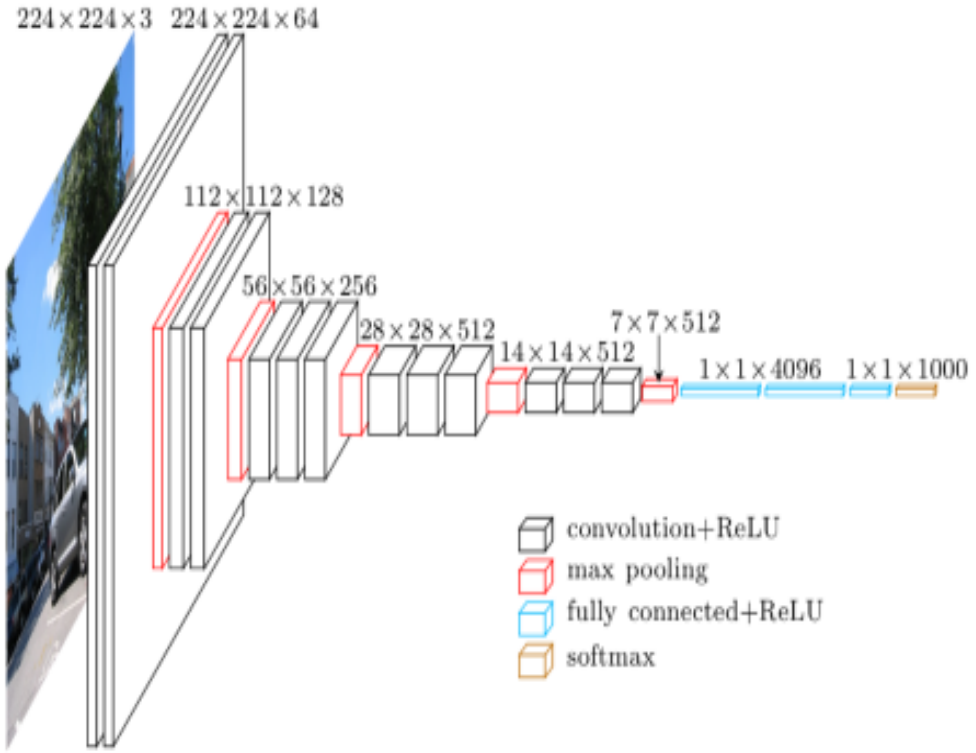


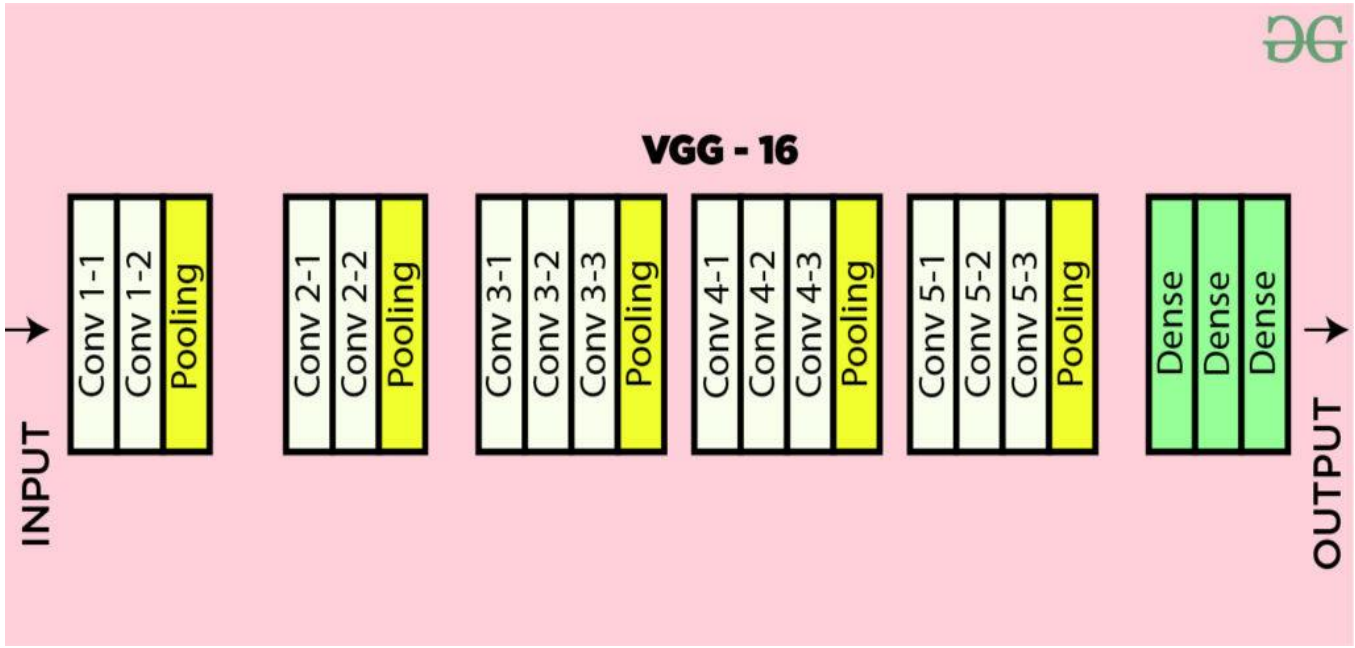
Freeze FE and Re-train C



VGG-16

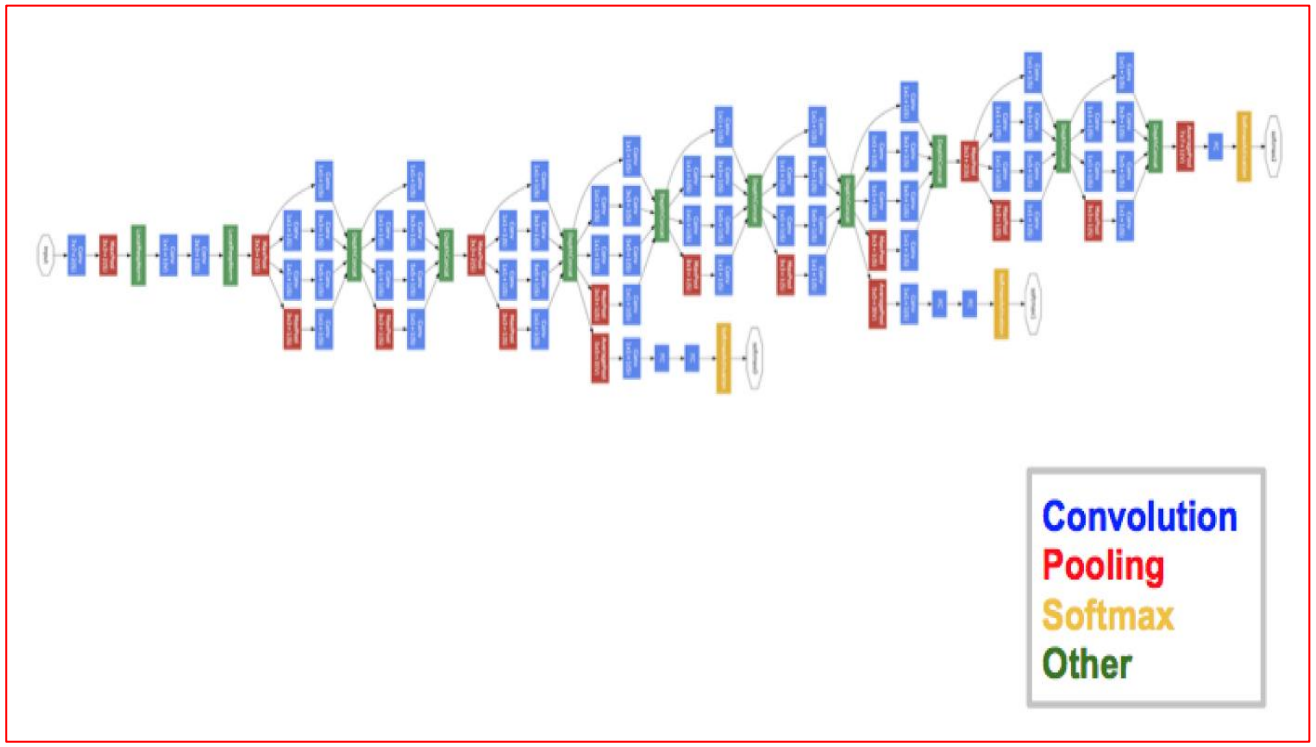
- Used to win ILSVR (ImageNet Large Scale Visual Recognition Challenge) competition in 2014.
- It is considered to be one of the excellent vision model architecture to date.
- VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION”.

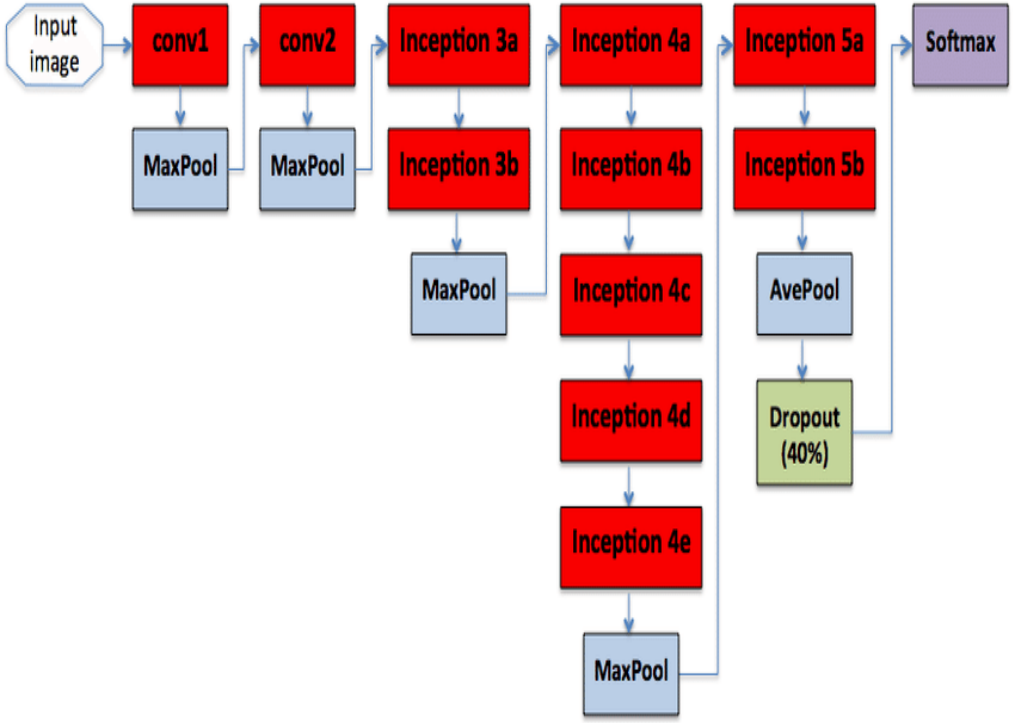




GOOGLE NET

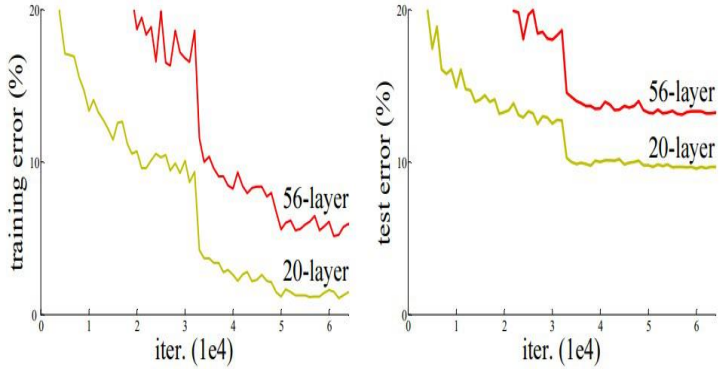
- Google Net (or Inception V1) was proposed by research at Google (with the collaboration of various universities) in 2014 in the research paper titled “Going Deeper with Convolutions”.
- Winner at the ILSVRC 2014 image classification challenge.
- It has provided a significant decrease in error rate as compared to previous winners AlexNet (Winner of ILSVRC 2012) and ZF-Net (Winner of ILSVRC 2013) and significantly less error rate than VGG (2014 runner up).
- *1×1* convolutions in the middle of the architecture and global average pooling.





RES NET (Residual Networks (ResNet))

- Every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate.

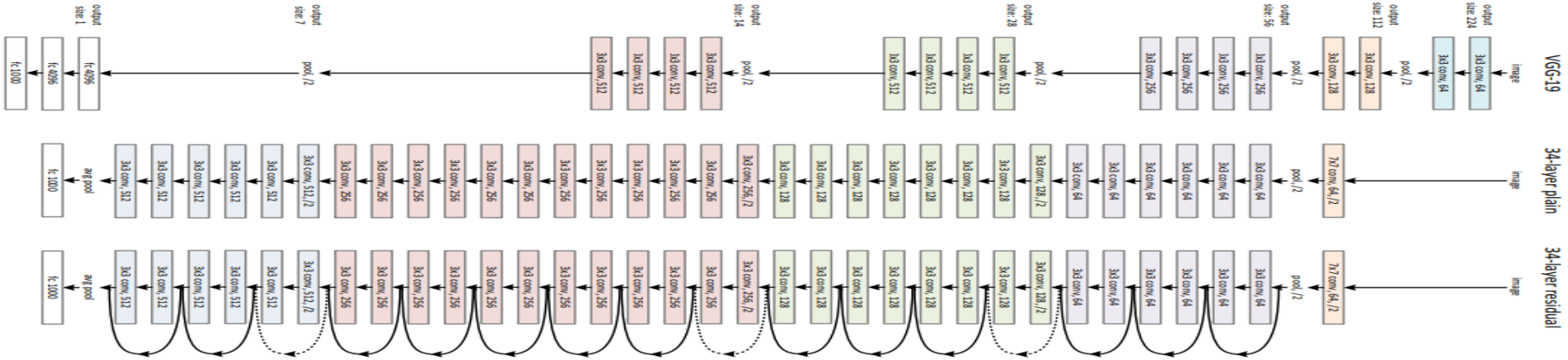


Vanishing/Exploding gradient

- Gradient to become 0 or too large.
- Increases number of layers, the training and test error rate also increases.

- ResNet, which was proposed in 2015 by researchers at Microsoft Research introduced a new architecture called Residual Network.
- In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Network. In this network, we use a technique called skip ***connections***.
- The skip connection skips training from a few layers and connects directly to the output.

CIFAR-10 dataset
60,000 – 10 classes

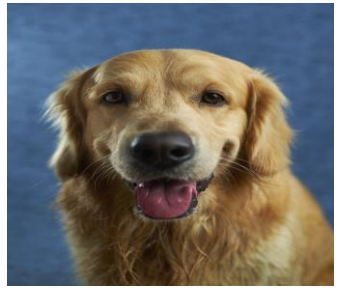


VGG-16 TRANSFER LEARNING

Import the model

```
from keras.applications.vgg16 import VGG16
model = VGG16(weights='imagenet')
print(model.summary())
```

Golden_retriever



Load sample Image

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input,decode_predictions
import numpy as np

img_path = '/kaggle/input/images/dog.jpg'
img = image.load_img(img_path,color_mode='rgb', target_size=(224, 224))
display(img)
```

Making the image size compatible with VGG16 input

```
x = image.img_to_array(img)  x.shape
x = np.expand_dims(x, axis=0)
```

Making the prediction

```
x = preprocess_input(x)
features = model.predict(x)
p = decode_predictions(features)
```

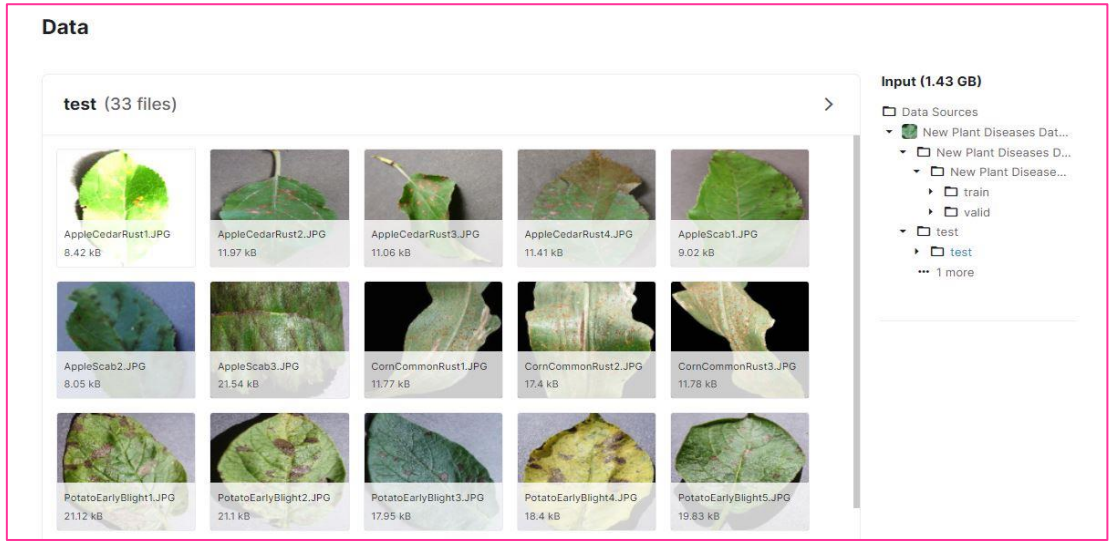
Output

```
[[('n02099601', 'golden_retriever', 0.8579672),
('n02099267', 'flat-coated_retriever', 0.018425034),
('n04409515', 'tennis_ball', 0.01615624),
('n02099712', 'Labrador_retriever', 0.015078514),
('n02099849', 'Chesapeake_Bay_retriever',
0.012522769)]]
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544

PLANT DISEASE DETECTION (VGG-16)



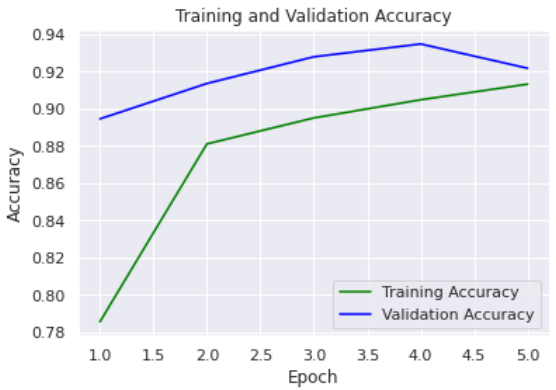
Model Preparation

```
base_model=VGG16(include_top=False,input_shape=(224,224,3))
base_model.trainable=False

classifier=keras.models.Sequential()
classifier.add(base_model)
classifier.add(Flatten())
classifier.add(Dense(38,activation='softmax'))
classifier.summary()

classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Freezing Layers

Train the Model

```
start = datetime.now()
history = classifier.fit (training_set, steps_per_epoch=train_num//batch_size,
validation_data=valid_set, epochs=5, validation_steps=valid_num//batch_size,)

duration = datetime.now() - start
print("Training completed in time: ", duration)
```



TL VS FINE-TUNING

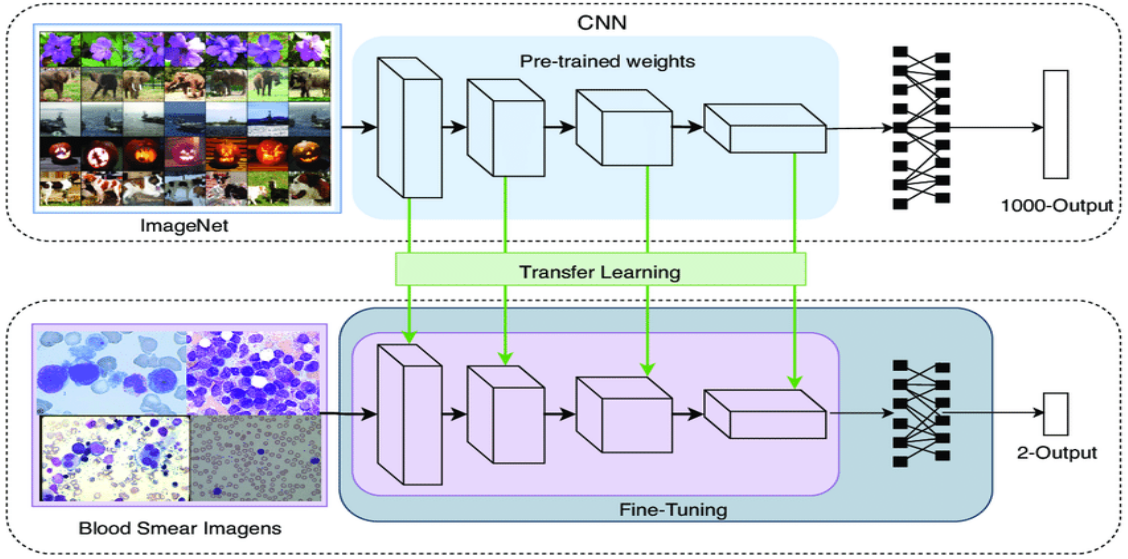
- Fine-tuning, in general, means making small adjustments to a process to achieve the desired output or performance.
- Fine-tuning is a process that takes a model that has already been trained for one given task and then tunes or tweaks the model to make it perform a second similar task.
- Here retrain not only the classifier layer but also retrain the feature extraction stage (the convolutional and pooling layers)

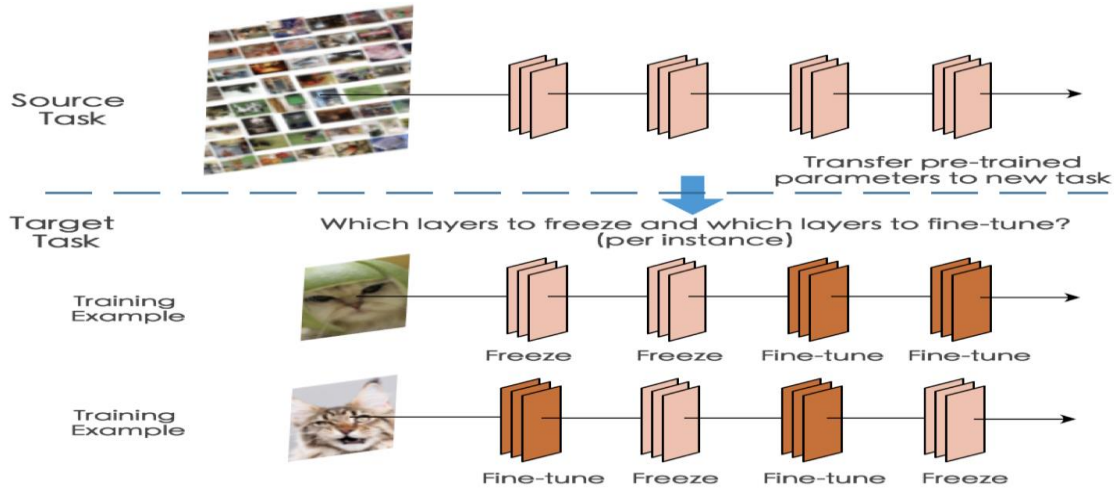
Two options:

- fine-tune all layers, and
- fine-tune only higher-level layers keeping some of the earlier layers fixed

So? which is one to choose?

- Start with transfer learning, if needed move to FT





FINE-TUNING

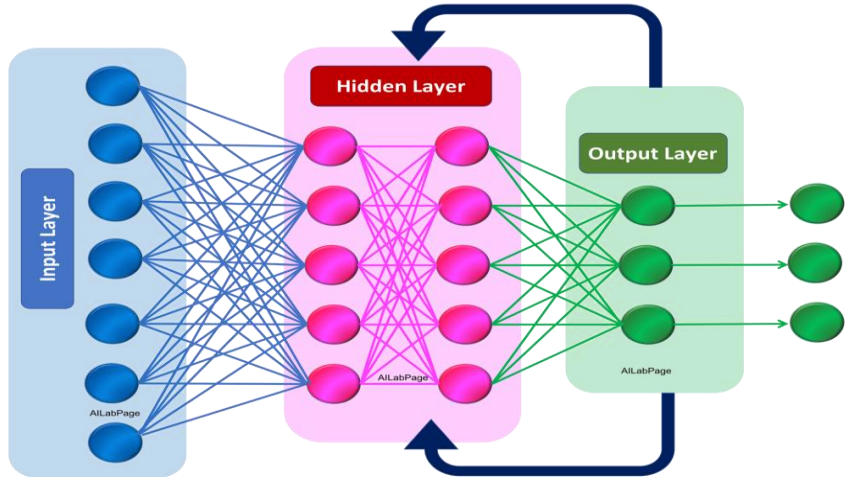
```
from keras.models
import Sequential
from keras import layers, applications, preprocessing. image
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation,GlobalMaxPooling2D
import ImageDataGenerator
from keras import optimizers , applications
import VGG16 from keras.models
import Model

image_size = 224
input_shape = (image_size, image_size, 3)
epochs = 5
batch_size = 16
pre_trained_model = VGG16(input_shape=input_shape, include_top=False, weights="imagenet")
for layer in pre_trained_model.layers[:15]:
        layer.trainable = False
for layer in pre_trained_model.layers[15:]:
        layer.trainable = True
last_layer = pre_trained_model.get_layer('block5_pool')
last_output = last_layer.output
```

```
# Flatten the output layer to 1 dimension
x = GlobalMaxPooling2D() (last_output)
# Add a fully connected layer with 512 hidden units and ReLU activation
x = Dense(512, activation='relu')(x)
# Add a dropout rate of 0.5
x = Dropout(0.5)(x)
# Add a final sigmoid layer for classification
x = layers.Dense (1, activation='sigmoid')(x)
model = Model (pre_trained_model.input, x)
model.compile (loss='binary_crossentropy',
optimizer=optimizers.SGD (lr=1e-4, momentum=0.9),
metrics=['accuracy'])
model.summary()
```

https://github.com/spmallick/learnopencv/blob/master/Keras-Fine-Tuning/Keras-finetune-vgg.ipynb

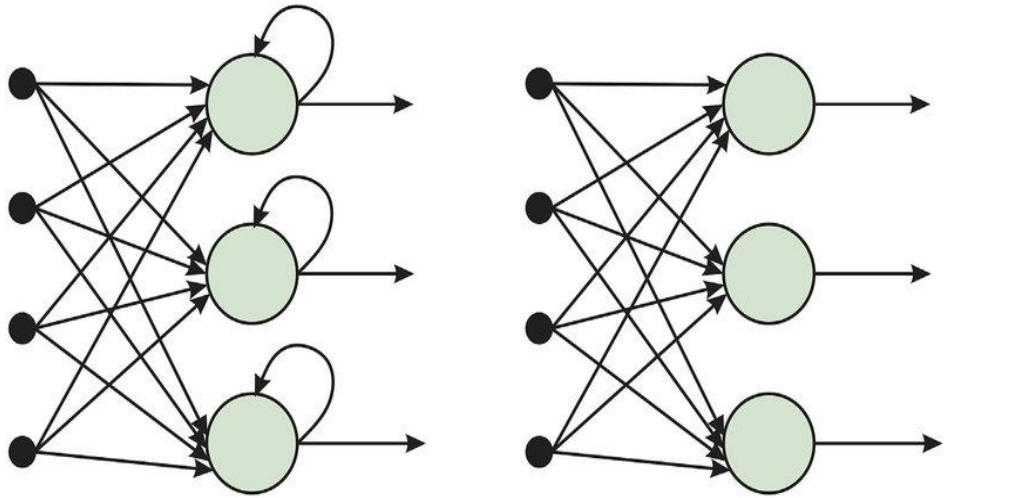
RECURRENT NEURAL NETWORK (RNN)

- Recurrent Networks are a type of artificial neural network designed to recognize patterns in sequences of data, such as auto-complete text, language translation, the spoken word, sentiment analysis.
- It is useful for NLP.
- Recurrent Neural Networks use a **backpropagation** algorithm for training.
- Because of their **internal memory**, RNNs are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next.



- An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighbourhood.

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or relu as an activation function.



(a) Recurrent Neural Network (b) Feed-Forward Neural Network

GOOGLE STOCK PRICE PREDICTION - RNN

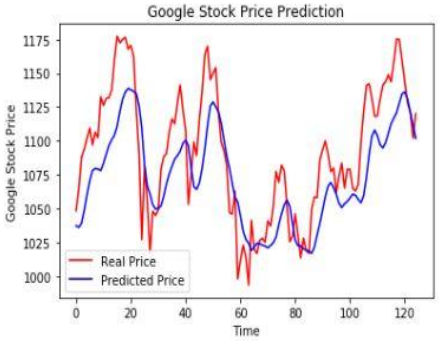
	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-01-02	357.385559	361.151062	355.959839	359.288177	359.288177	5115500
1	2013-01-03	360.122742	363.600128	358.031342	359.496826	359.496826	4666500
2	2013-01-04	362.313507	368.339294	361.488861	366.600616	366.600616	5562800
3	2013-01-07	365.348755	367.301056	362.929504	365.001007	365.001007	3332900
4	2013-01-08	365.393463	365.771027	359.874359	364.280701	364.280701	3373900
5	2013-01-09	363.769043	366.789398	361.945892	366.675140	366.675140	4075700
6	2013-01-10	369.014923	370.092896	364.380066	368.344269	368.344269	3695100
7	2013-01-11	368.602600	368.816193	365.771027	367.604095	367.604095	2587000
8	2013-01-14	366.118744	368.701935	358.841095	359.288177	359.288177	5765000
9	2013-01-15	357.340851	365.125214	353.749207	360.122742	360.122742	7906300
10	2013-01-16	358.865936	359.829651	354.529144	355.284210	355.284210	4073100

Feature Scaling

```
from sklearn.preprocessing
import MinMaxScaler
sc = MinMaxScaler (feature_range = (0,1))
training_scaled = sc.fit_transform (trainset)
```

Array (
[[0.01011148], [0.01388614], [0.01690727], ...,
[0.97543954], [0.9674549], [0.96072522]])

RNN Model




```
from keras.models
import Sequential from keras.layers
import Dense from keras.layers
import LSTM from keras.layers import Dropout
regressor = Sequential()
regressor.add ( LSTM(units = 50,return_sequences = True,input_shape =
(x_train.shape[1],1)))
regressor.add ( Dropout(0.2))
regressor.add ( LSTM(units = 50,return_sequences = True))
regressor.add ( Dropout(0.2))
regressor.add ( LSTM(units = 50,return_sequences = True))
regressor.add ( Dropout(0.2))
regressor.add ( LSTM(units = 50))
regressor.add ( Dropout(0.2))

regressor.add ( Dense(units = 1))
regressor.compile (optimizer = 'adam',loss = 'mean_squared_error')
regressor.fit (x_train,y_train,epochs = 100, batch_size = 32)
```

LSTM- LONG SHORT TERM MEMORY

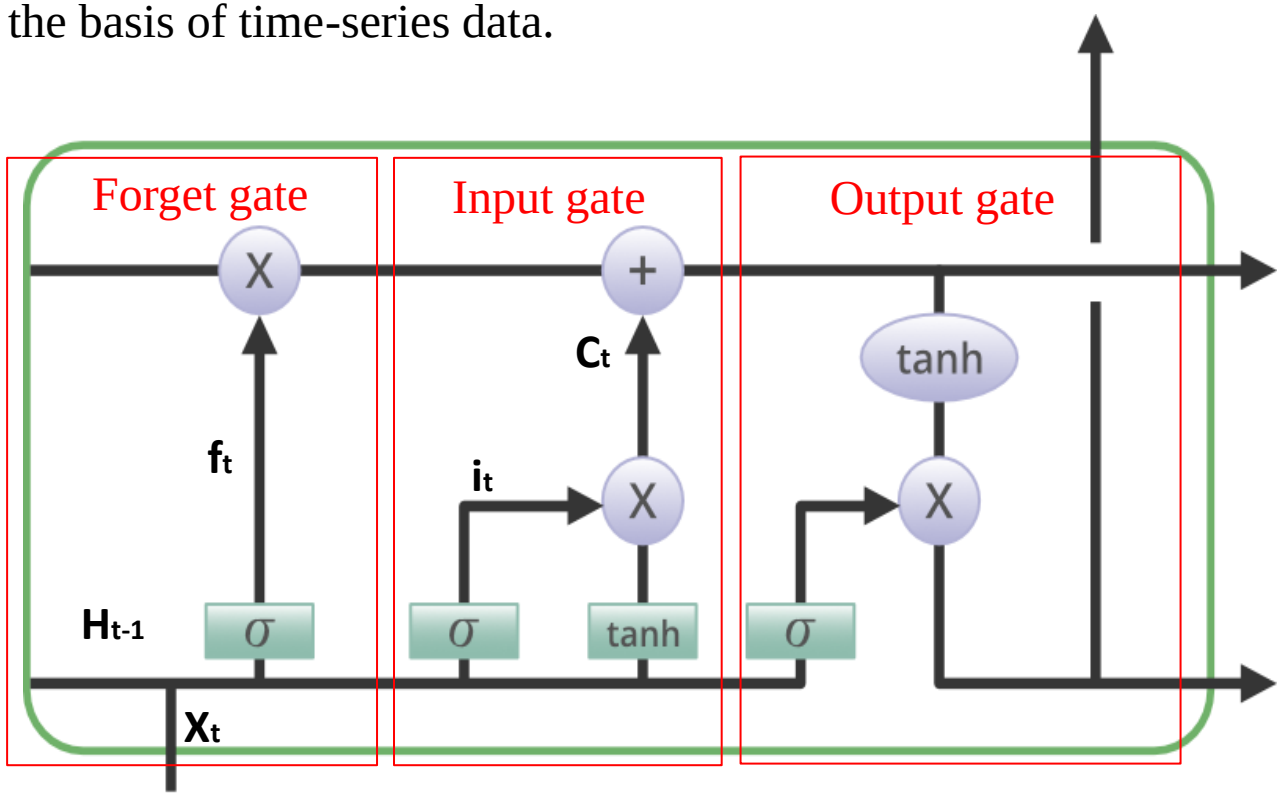
- Long Short Term Memory is a kind of recurrent neural network.
- In RNN output from the last step is fed as input in the current step.
- As the gap length increases RNN does not give an efficient performance.
- LSTM can by default retain the information for a long period of time.
- It is used for processing, predicting, and classifying on the basis of time-series data.

Combination:

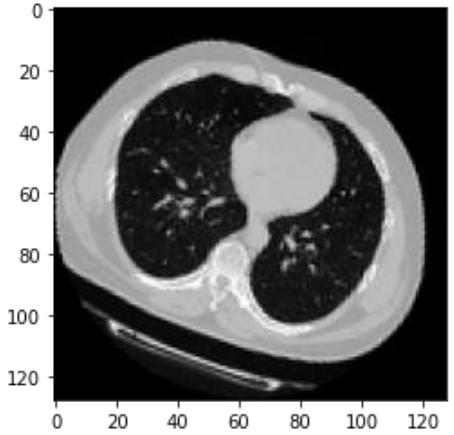
- LSTM
- CNN+LSTM
- LSTM+RNN
- Bi-Directional LSTM (Forward-Past to Future, Backward-Future to Past)

Some of the famous applications of LSTM include:

1.Language Modelling
2.Machine Translation
3.Image Captioning
4.Handwriting generation
5.Question Answering Chatbots

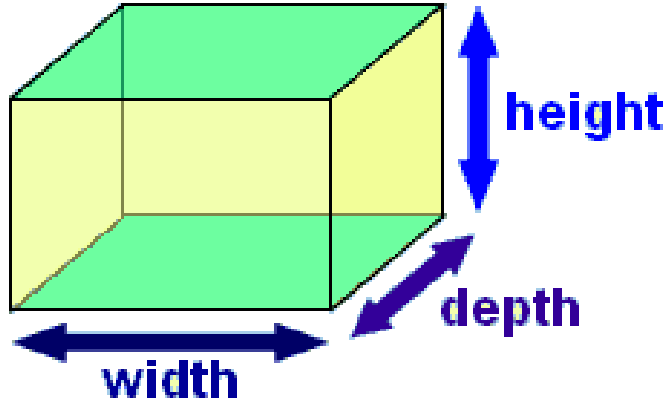


3D CNN



Lung CT Scans with COVID-19
(128,128,64,1)

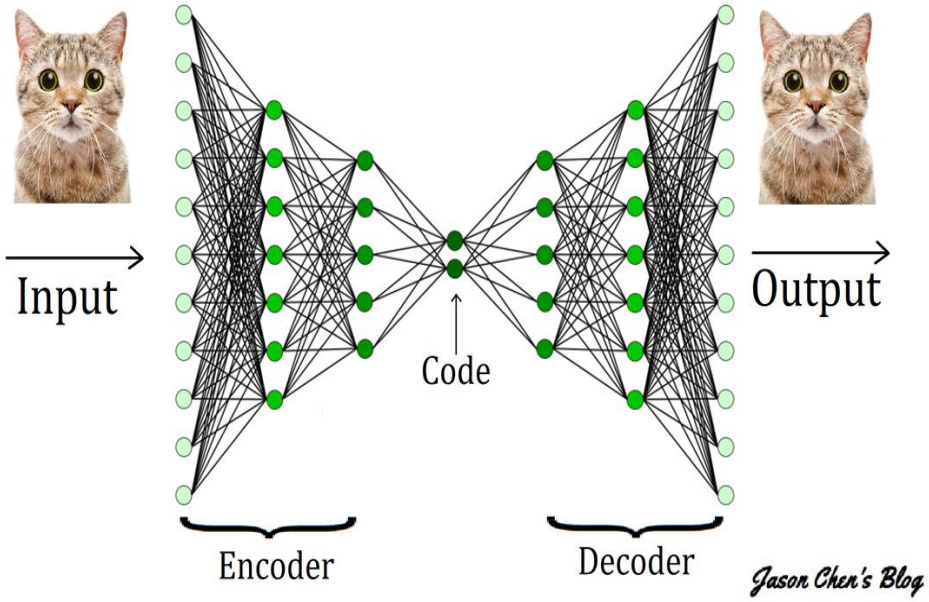
```
def get_model(width=128, height=128, depth=64):
    inputs = keras.Input((width, height, depth, 1))
    x = layers.Conv3D(filters=64, kernel_size=3, activation="relu")(inputs)
    x = layers.MaxPool3D(pool_size=2)(x) x = layers.BatchNormalization()(x)
    x = layers.Conv3D(filters=64, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x) x = layers.BatchNormalization()(x)
    x = layers.Conv3D(filters=128, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Conv3D(filters=256, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.GlobalAveragePooling3D()(x)
    x = layers.Dense(units=512, activation="relu")(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(units=1, activation="sigmoid")(x)
    # Define the model.
    model = keras.Model(inputs, outputs, name="3dcnn")
    return model
model = get_model(width=128, height=128, depth=64)
model.summary()
```



https://keras.io/examples/vision/3D_image_classification/

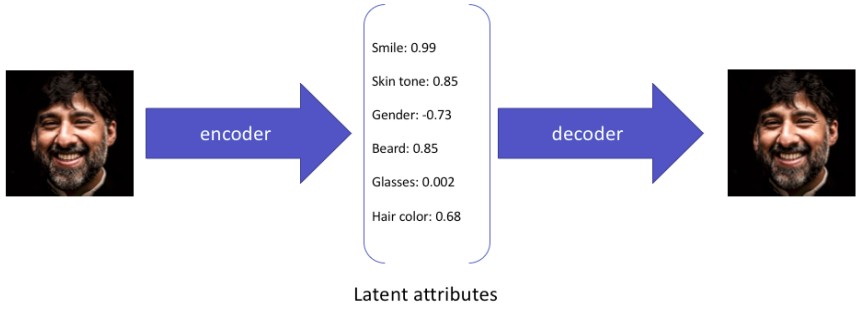
AUTOENCODERS

- An autoencoder is a type of artificial neural network used to learn data encodings in an unsupervised manner.
- Aim of an autoencoder is to learn a lower-dimensional representation (encoding) for higher-dimensional data, typically for dimensionality reduction, by training the network to capture the most important parts of the input image.
- Autoencoders can be used for image denoising, image compression, and, in some cases, even generation of image data.



Popular Auto Encoders:

- Undercomplete autoencoders
- Sparse autoencoders
- Contractive autoencoders
- Denoising autoencoders
- Variational Autoencoders (for generative modeling)



GAN – GENERATIVE ADVERSARIAL NETWORK

- It is used for unsupervised learning.
- It was developed and introduced by Ian J. Goodfellow in 2014.
- GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

Why were GANs developed in the first place?

- Neural nets can be easily fooled into misclassifying things by adding only a small amount of noise into the original data.
- Most machine learning models learn from a limited amount of data, which is a huge drawback, as it is prone to overfitting.
- The mapping between the input and the output is almost linear.

How do GANs work?

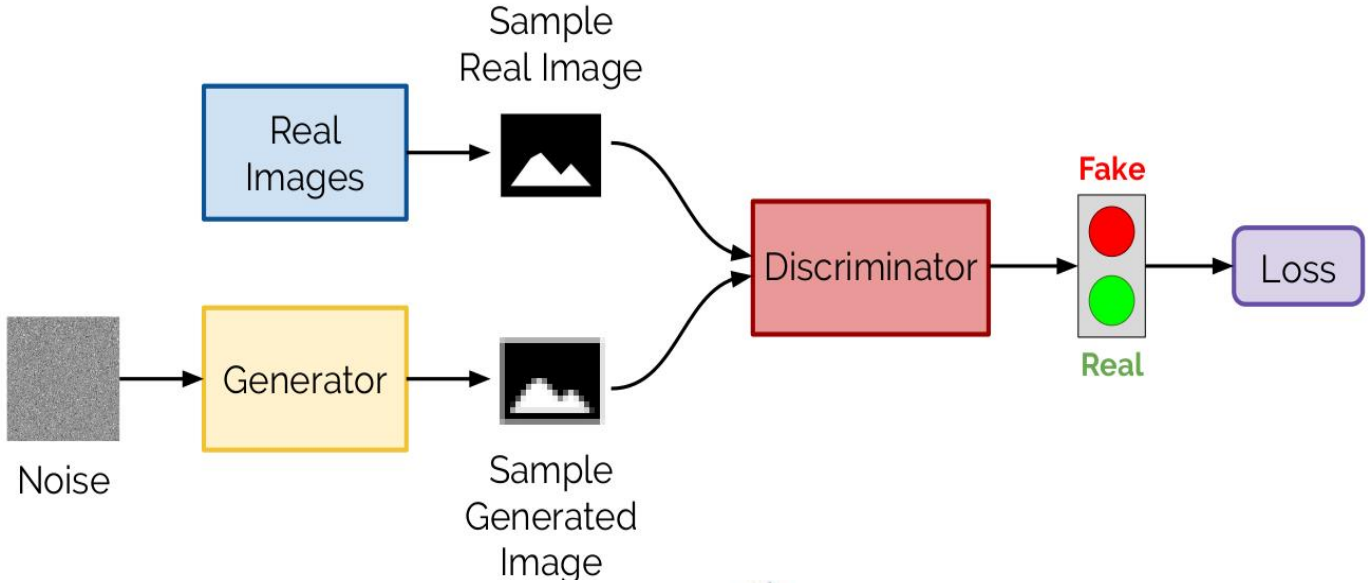
Generative Adversarial Networks (GANs) can be broken down into three parts:

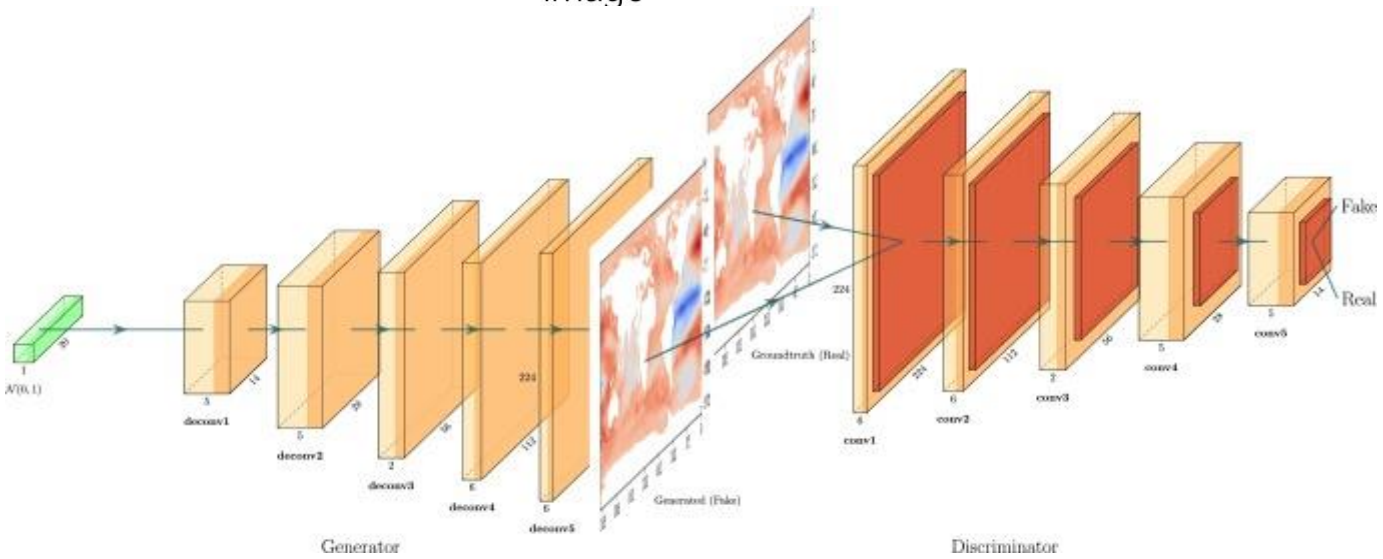
- **Generative ("the artist"):** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- **Adversarial:** The training of a model is done in an adversarial setting.
- **Networks:** Use deep neural networks as the artificial intelligence (AI) algorithms for training purposes.

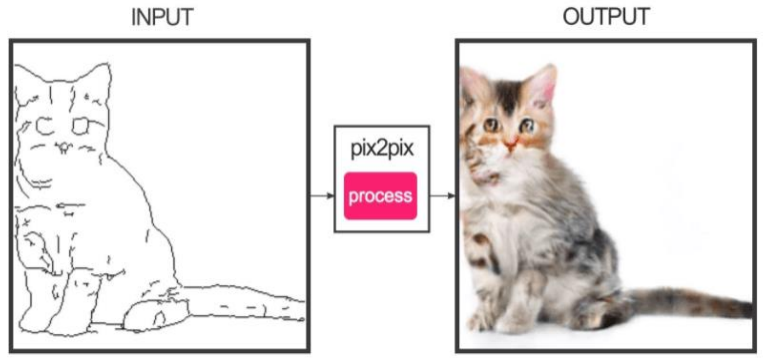
Types:

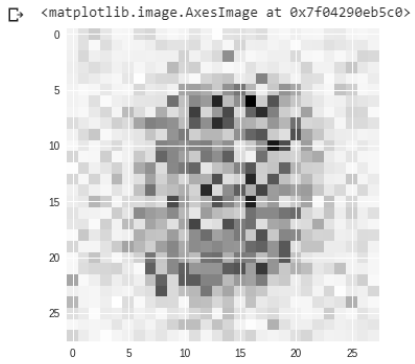
- Vanilla GAN
- Conditional GAN (CGAN)
- Deep Convolutional GAN (DCGAN)
- Laplacian Pyramid GAN (LAPGAN)
- Super Resolution GAN (SRGAN)

GAN – GENERATIVE ADVERSARIAL NETWORK









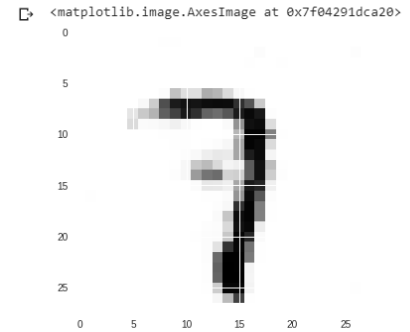
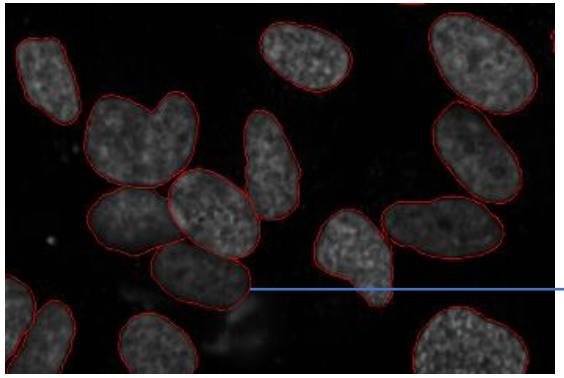


IMAGE SEGMENTATION IN DL

- Divide or partition the image into various parts called segments.
- In a nutshell, An image is a collection or set of different pixels. It groups together the pixels that have similar attributes using image segmentation.
- Object detection builds a bounding box corresponding to each class in the image. But it tells us nothing about the shape of the object.
- Image segmentation creates a pixel-wise mask for each object in the image. This technique gives us a far more granular understanding of the object(s) in the image.
- Image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance segmentation).
- Semantic segmentation performs pixel-level labeling with a set of object categories (for example, people, trees, sky, cars) for all image pixels.

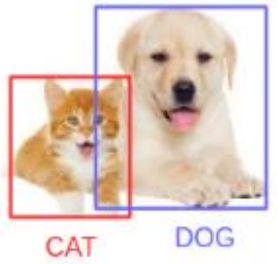


Can identify cancer from the shape of the cell

Here, Object detection only detects an object in the image, so do not use this method here.

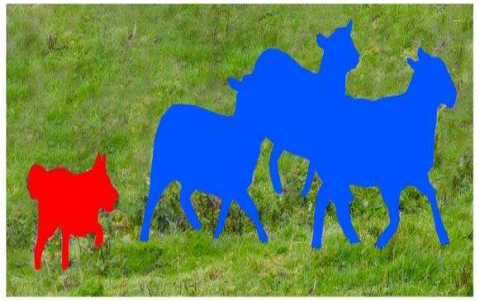
Image segmentation is done by the threshold value.

Image Localization

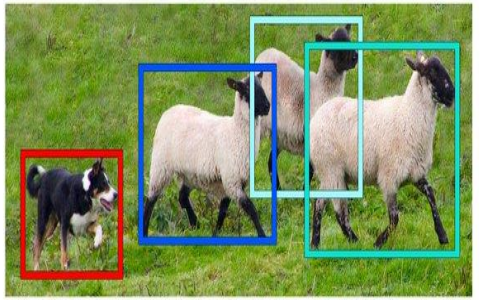


Object Detection

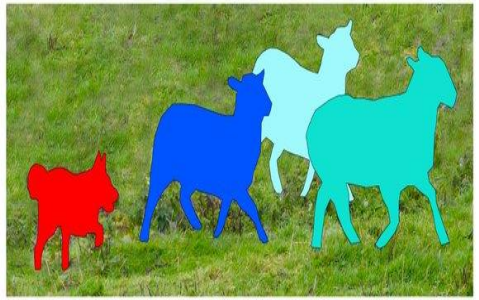
Image Recognition



Semantic Segmentation



Object Detection



Instance Segmentation

Image Segmentation Model in DL

- Fully Convolutional networks
- Convolutional Models with Graphical Models
- Encoder-Decoder Based Models
- Multi-Scale and Pyramid Network-Based Models
- R-CNN Based Models (for Instance Segmentation)
- Dilated Convolutional Models and DeepLab Family
- Recurrent Neural Network Based Models
- Attention-Based Models
- Generative Models and Adversarial Training
- CNN Models with Active Contour Models

Object Detection Model

- Fast R-CNN
- Faster R-CNN
- Histogram of Oriented Gradients (HOG)
- Region-based Convolutional Neural Networks (R-CNN)
- Region-based Fully Convolutional Network (R-FCN)
- Single Shot Detector (SSD)
- Spatial Pyramid Pooling (SPP-net)
- YOLO (You Only Look Once)

What is Ground Truth Dataset and Image Annotation?

Ground Truth

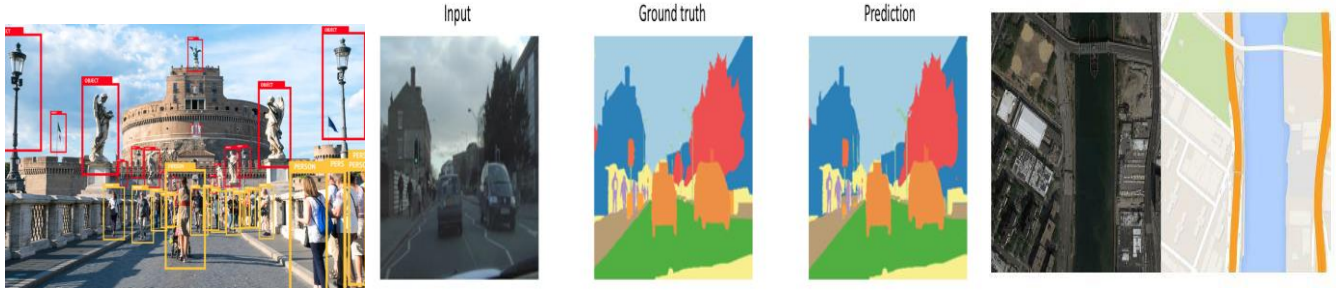
- 'Ground truth' represents the objective, humanly verifiable observation of the state of an object or information that might be considered a fact.
- A ground-truth dataset is a regular dataset, but with annotations added to it.

Why does ground truth matter

- Supervised
- Semi-supervised
- Unsupervised

Image Annotation

- Annotations can be boxes drawn over images, written text indicating samples.
- **Image annotation** is the process of labeling or classifying an image using text, annotation tools, or both, to show the data features you want your model to recognize on its own.
- Image annotation is a type of data labeling that is sometimes called tagging, transcribing, or processing.
- we also can annotate videos continuously, as a stream, or frame by frame.



Visualization tools for ML

Always understand where it works as expected and where it fails miserably.

What can we do with this?

- Evaluation Metrics
- Performance Chart
- Learning Curves to estimate Overfitting
- Model predictions on best/worst case
- Model training & inference

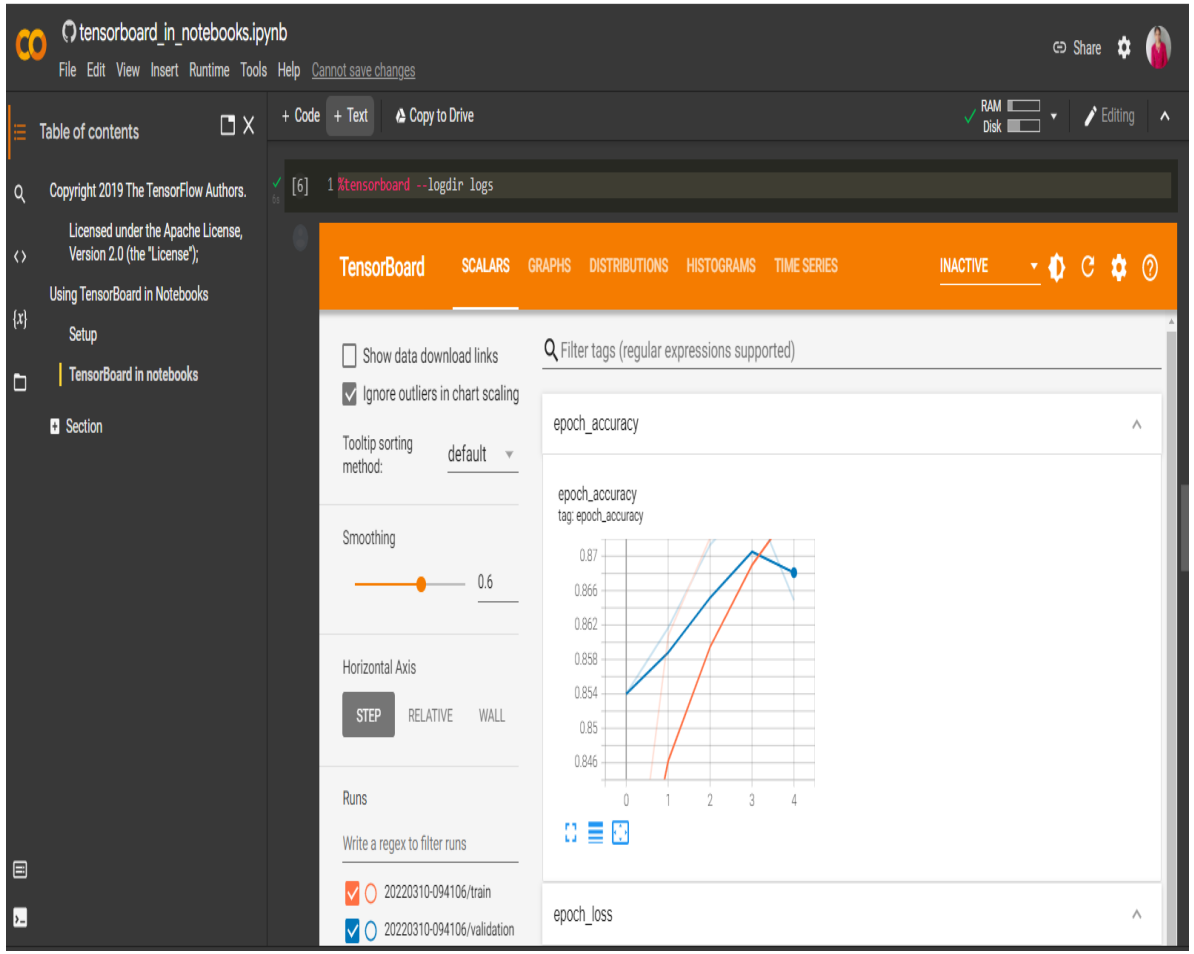
Visualization Tools

- Neptune
- WandB
- Comet.ml
- TensorBoard
- Visdom
- HiPlot

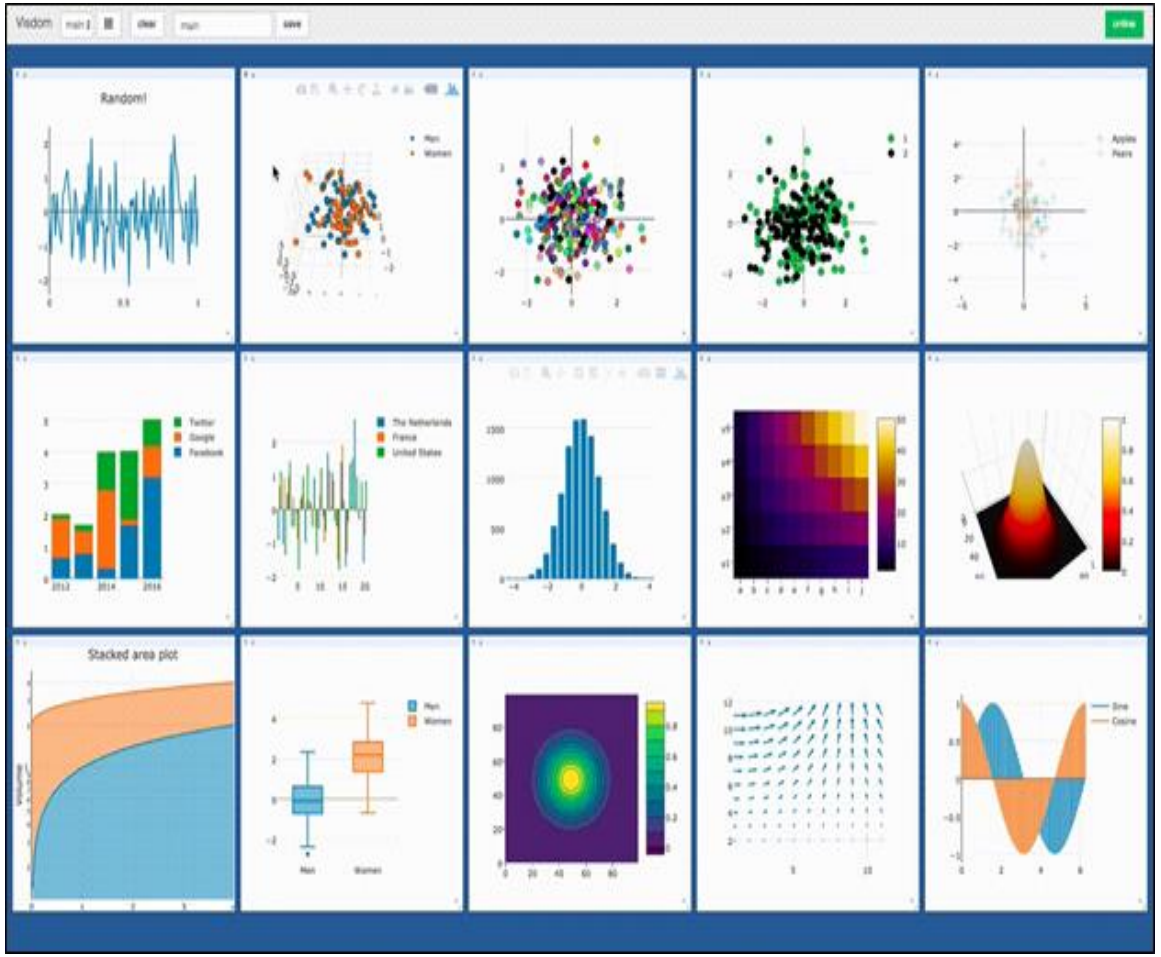
https://neptune.ai/blog/the-best-tools-for-machine-learning-model-visualization#:~:text=4. ,TensorBoard,debugging%20of%20machine%20learning%20models.

VISUALIZATION TOOLS FOR ML

TensorBoard



Visdom



Thank You...!