Data Science

- A Computer can give any answer (or) data which is already existed in storage / harddisk (or) from any place.

  But Computer Doesnot have Intelligence to give unknown data (or) Cannot predict the data.

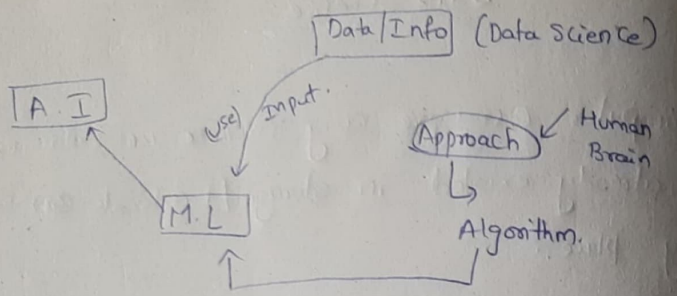
  For this we are going give a Intelligence which is artifically Create which is called as Artifical Intelligence.

  For this artifical Program we Create a Python program / Algorithm.

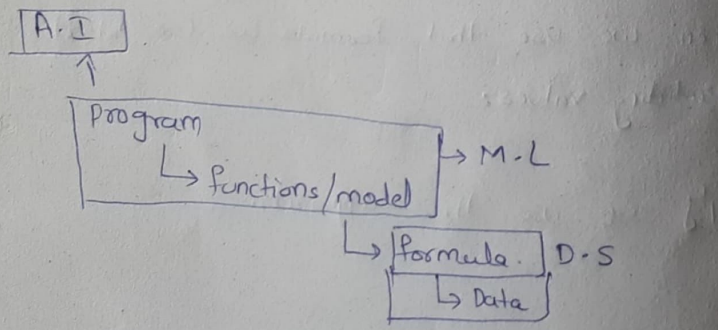
- The Human brain first take historical data and identifies a pattern and Conclude a formula.

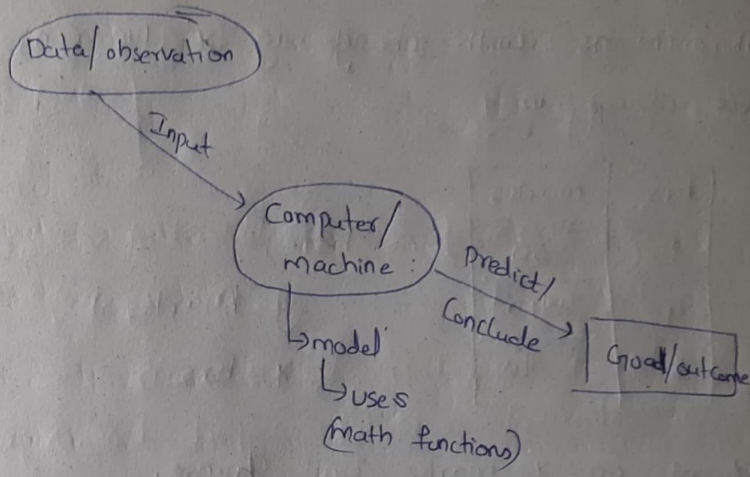
- When we use that formula we can start Predicting values.

- III'y we use this approach.



- From historical Data we Comes (or) Conclude and get a pattern which is given to machine.
- Getting this Data & analysing pattern is data Science.
- Using few steps and using the pattern and Creating an Algorithm and given to machine which takes Data as input and predicts out put which becomes machine learning.
- Indirectly one formula is a model.





- The outcome we need is called as Dependent variable (D.V) (or) target (or) outcome.
- The variables which are used to predict is called as Independent variable (I.V) (or) predictor.
- The Entire fields and records Come under (or) called as Data set.
- If one field is related (or) dependent on other field that relation is called Corelation.

- If we see any dataset like $y \propto x$ which means with change in 'x' we can see change in 'y'

Ex:-

hrs	marks
3	30
4	40
7	70

here,

y = marks

x = hrs

Based on 'x' (hrs) marks will change.

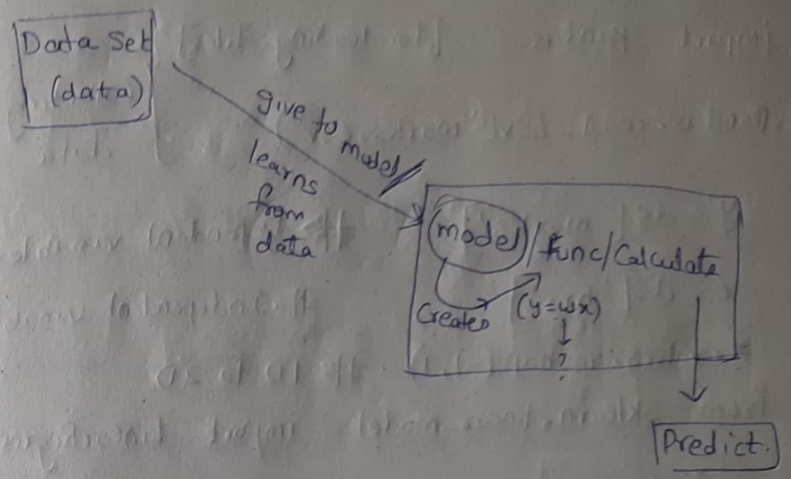
here we can identify a pattern / formula

$y = wx$

↳ Coefficent / weight

we need to identify this Coeffient / weight.

For this we are going to use simple Linear Regression Algorithm



- Give dat set to model which learns from the data and Create a function/pattern. And function identifies Coeffient / weight and predict the output.

Code :-

marks.csv → Comma Seperated Values

```
hrs, marks
3, 30
7, 70
2, 20
8, 80
```

delimeter / field seperator

SimpleML.[redacted]py

```
import pandas.   [for loading files]
ds=pandas.read_csv('marks.csv')        } Collecting data

y = ds['marks']      # Dependent variable
x = ds['hrs']        # Independent variable
x.values.reshape(-1,1)  # 1D to 2D
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(x,y) ——→ learn/find (or) create function/ create model.

model.predict([[9]]) ——→ predicts/forecast
```

- As the value of x change value of y also change this continous changing of value is called as Regression Analysis

- To identify the coefficient/weight of a model.

Ex:- model.coef_

Array function inside numpy

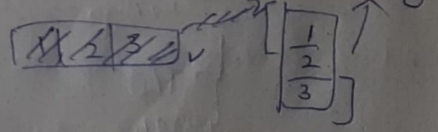
```
import numpy
a1 = numpy.array([1,2,3])
```

The main disadvantage of list is we cannot retrive column wise data in array we can achieve that.

Ex:-

```
a1 = numpy.array([1,2,3])  # 1D array/ # vector
```

```
a2 = numpy.array([[1],[2],[3]])  # 2D array
```



- numpy and pandas can be used to retrive column wise data.
- But many M.L uses numpy module because Sklearn is developed/used on numpy.
- For converting pandas to numpy data type we can use.

`X.values`

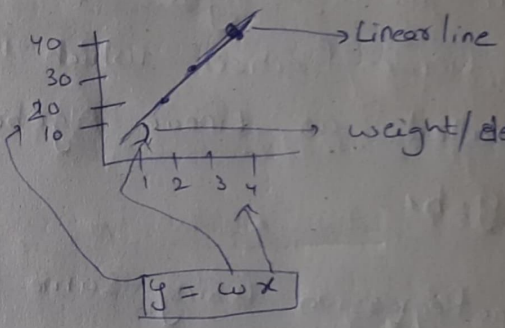
- use reshape() to change 1D to 2D

Data set (past) → Analysis (find the Pattern/ model) → Predict / Estimate

- Classification (~~Continous~~ data)
  - ↳ Algo
    - ↳ T/F
    - ↳ M/F
    - ↳ Different Classifications
- Continous (~~Classification~~)
  - ↳ Algo:
    - ↳ Data is continous we use continous Algorithms like "Regression"

The Term Linear comes for Linear Regression is when we plot graph the values (or) there comes a line of fit which passes through almost all the points.

So, that line is called linear. and Regression is continous values thats why it is called as Linear Regression.



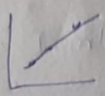
The ultimate goal is identifying that weight/angle which will help us to find the line of fit (line connects all points) and we can predict the next value.

- we have a module for plotting graphs in python.

Ex:- import matplotlib.pyplot as plt

- we have a function in matplotlib.pyplot

  ie; plt.scatter(x, y) → 

  plt.plot(x, y) → 

- In maths also for identifying a equation for a stright line we have Linear function which is also called as Linear Algebra.

  ie; $y = bx$

- In Linear Regression using same function (or) equation behind the scene.

  ie; $y = wx$

  ↓

  find this

- For finding "w" we don't use $w = \frac{y}{x}$ x

But Linear Regression use a concept of substution. i.e; keep random values into w and when $y \neq x$ happens then we use that weight.

- Data Analysis is done on historical data.
- Data Analyst is a person who predicts the data. (ML)

- Sklearn is module, linear_model is class and Linear Regression is a method.

Ex:-

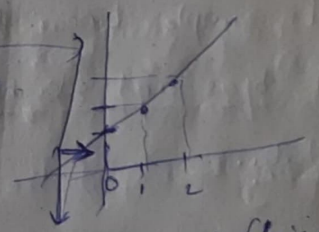
(x) years of Experience	Salary (y)
2-2	45,000
3-5	60,000
⋮	⋮

When a person has '0' years experience then $y = w * 0$ i.e; $\boxed{y = 0}$ x i.e; Salary = 0 that should not happen so that's why we add some amount that is called as bias [which will be min value]

$\boxed{y = b + wx}$

↓

final Linear Regression formula



y-Intercept (b is what intercepts y-axis)

- To get the value of intercept

Ex:- model.intercept_

Now,

we can calculate by getting weight and intercept values and multiple for specific years person's salary.

Ex:- years = 2 years

(w) weight = model.coef_

(b) intercept = model.intercept_

$y = b + w * x$

↓

we get salary.

* The goal of Simple Linear Regression Algorithm is to find bias and weight. By using historical data.

- In real life we don't have simple data.

- while finding the result. we have a weight [i.e; predicted by past data]

i.e; $\hat{y}$.

- And for next predicting value is y.

i.e; $y - \hat{y}$ = gives you the error/residual

i.e; how much difference you predicted from actual value.

- After creating a model [i.e; model.fit(x,y)] we can store that model for future purpose for prediction.

[pip install joblib]

Ex:- ~~from sklearn.externals~~

import joblib

joblib.dump(model, 'salary_model.PKL')

↓

filename where model save.

- we can give any extension for the model mostly we prefer pkl.
- Now, we can share this model/file and they can predict the output.

Now for using,

Ex:- model = joblib.load("Salary-model.pkl")

Now, model.predict([[value]])

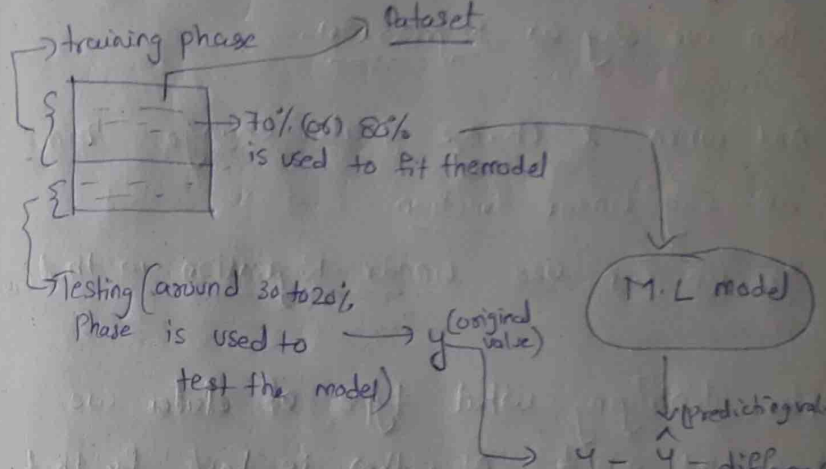
what we are doing?

First, get the dataset and identify what you want to predict. (dependent variable)

Second, Ask for Domain expert who has complete knowledge on that Domain and get information regarding features/fields/independent variables which are going to impact the predicting variable.

Now, you give these features dataset to the model and model is going to find coeffient/weight and bias.

- weight is always related to features i.e given based on how much (+ve (or) -ve) that feature is going to impact predicting variable



Syntax (or) Ex:-

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)
```

features/fields

predicted value (or) output

```
model.fit(x_train, y_train)

Predict =
model.predict(x_test)
```

- When y is a value which is continous then we say as Regression.
- And when, x change y also change then we say Linear function.

So, we can use Linear Regression in that case.

After identifying what type of data we have and what Algorithm is best for that then we use that Algorithm/function.

- And Algorithm finds bias and weight.

How Algorithm finds bias and weight?

The Algorithm does hit and trails and keep multiple values and finds the bias and weight when L.H.S is nearly equal to R.H.S ($y = b + wx$)

Now, bias and weight should be those values such that **L.H.S - R.H.S = Error**

this error should be minimum.

L.H.S - R.H.S = error

$y - \hat{y} = \text{error}$ ⟶ for 1 record.

for all records.

$$\frac{|y_1 - \hat{y}_1| + |y_2 - \hat{y}_1| + \cdots + |y_n - \hat{y}_n|}{n} \quad \text{(MAE)}$$

[Mean Absolute error]

for 'n' records the Model uses this formula. and gives minimum error/loss.

$$\frac{\sum_{x=1} y_x - \hat{y}_x}{n} \quad \text{(MAE)}$$

from sklearn import metrics

metrics.mean_absolute_error(y_test, predict)

↓

This gives error (or) predicted value is correct or wrong.

- If we have multiple features it is called as multivarient.
- If we have single feature it is called as univarient.
- want to see few records then use:-

  dataset.head()

- want information regarding dataset

  dataset.info()

- dataset.columns (gives the fields of dataset)

- when dataset has strings instead of numbers then we need to convert the strings to numbers ie; called as label encoding.

Ex:- from sklearn.preprocessing import LabelEncoder

state_le = LabelEncoder()

sf = state_le.fit_transform(state) → fieldname/feature

This line gives a unique value to specific state.

* few fields like state which has multiple states and when we give weight to it. It doesn't have any meaning as it is called as categorical variable.
- we have weight to specific state is meaningful but for entire state field which has multiple states giving that a weight is useless.
- But we should not loose data so, we are going to create new fields/features with those categorical variables/categories with all those states. (Always finite variables)

Ex:- newyork California Las Angles

0	1	0
1	0	0
2	1	0
3	0	1

we have 3 states and converting into **one-hot encoding** which will have a value for specific state in specific record.

Ex:- from Sklearn.Preprocessing import OneHotEncoder

```
state_ohe = OneHotEncoder(sf)
sf = sf.reshape(-1,1)
ohe = state_ohe.fit_transform(sf)
```

ohe.toarray() → to see values as one-hot encoding.

Dummy Variable Trap:-

Always every feature should be independent to each other.

if x_2 is changing based on x_1 then we get an issue called multicollinearity.

So, if we have like this then only use one field/feature.

its like → Ex: I have studied for 12hrs.
(or)
I haven't studied for 12hrs. } both are same

Ex:- M / F → If the person is not M then he is Female so, we can use any one field.

This concept is called removing Dummy variable Trap.

Ex:-
```
State_final = ohe[:,0:2]
```
↓
removed Las Angles state as we can identify it by newyork & California.

Computer Vision

numpy:-

import numpy

numpy.array(array/List)

CV2:-

CV2 comes from ~~python~~ opencv-python module.

so,

```
import cv2
img=cv2.imread("imagename")
type(img)
img1 = img[100:200]
cv2.imwrite("mynewimg.jpg", img1)
```

imread → reads images from current location drive

type(img) → array.

cropped image.

writing (or) storing in a file.

The above code crops image and stores image.

- If you want to capture image from camera then

Ex:- image = cv2.VideoCapture(0)

0 means in built Camera and if Camera is external then it will be 1 (or) more based on Cameras Connected

myphoto = image.read()

Camera one and clicks image.

- myphoto[0] → is True, means image got captured (or) else got some error

- myphoto[1] → has image.

cv2.imwrite("capture.jpg", myphoto[1])

stores image as capture.jpg.

- For adding two images.

img1 = cv2.imread("img1.jpg")

img2 = cv2.imread("img2.jpg")

v1 = numpy.vstack((img1, img2)) → vertically.

h1 = numpy.hstack((img1, img2)) → horizontally

tuple.

---

previous example of M.L (continuation)

now we have removed dummy variable trap and kept those values in State_final variable.

X_final = numpy.hstack((X, State_final))

where X has removed State field. as

X = dataset['R&D', 'Admin', 'Marketing']

and doesn't take state field in X.

now, use.

model.fit(X_final, y)

Before this fit we can also split the data and fit so, that we can test with test data.

model.coef_

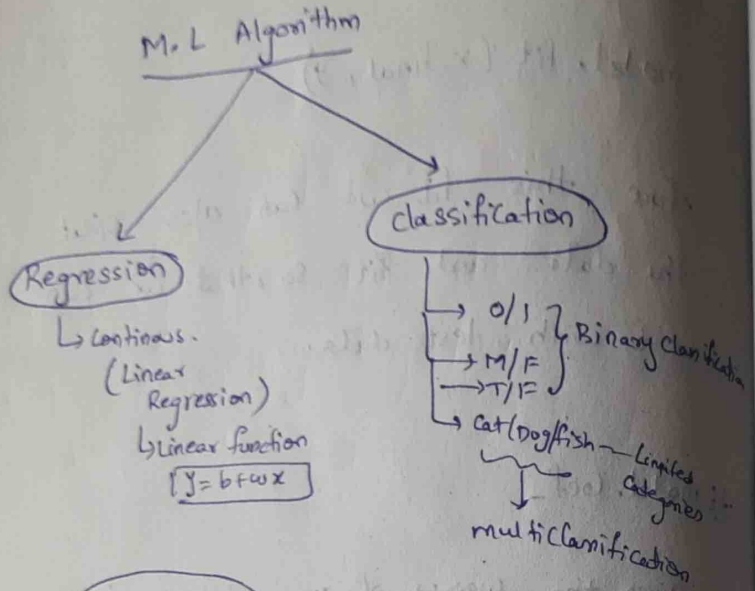
given the weight of every column. ie;

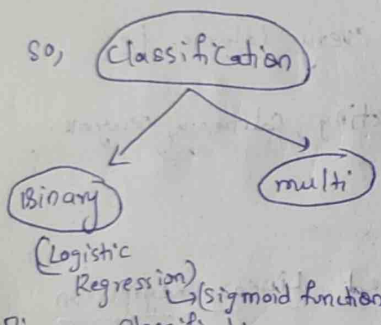
R&D, Admin, marketing, California, newyork.

Till now, we have used Linear Regression as data (or) predicting value is continous so, we used Regression.

Now, if the outcome is any one of two (or) more like M/F where we need to predict male (or) Female.

For this we use classification problem.





Binary Classification :-

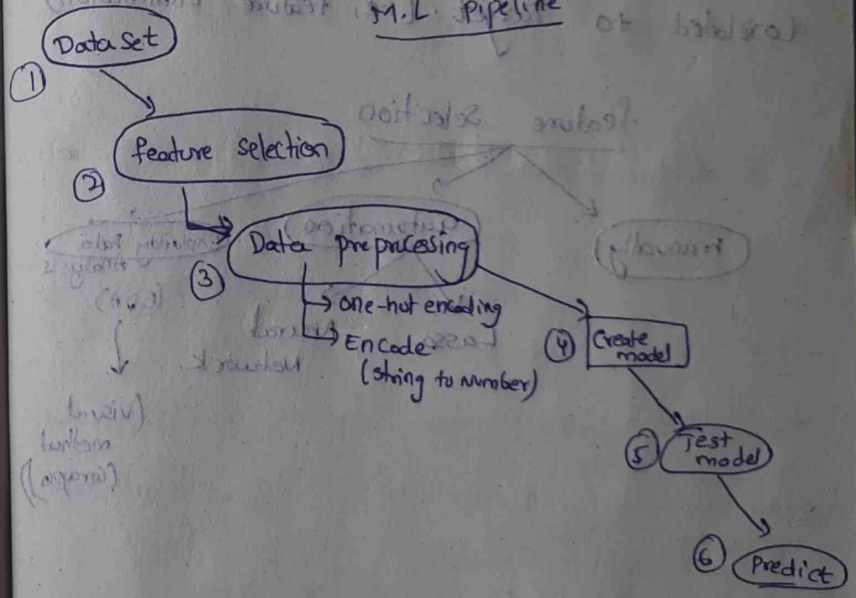
Logistic Regression internally uses Sigmoid function which is intern used for Binary Classification.

* use Kaggle.com for downloading dataset.

we are using titanic_train.csv.

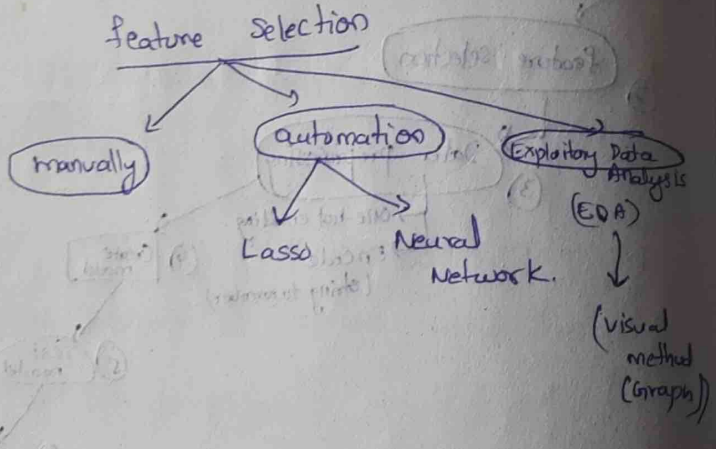
Ex:-
```
# import pandas
# dataset = pandas.read_csv("titanic_train.csv")
# dataset.info()
# dataset.columns
# dataset.head(5)
```

M.L pipeline



feature selection.

- First identify what field you want to Predict i.e; Target / Dependent variable.
- Now identify all fields/columns/features which are impacting Target/D.V. i.e; when field change Target changes. i.e; field is Corelated to Target (or) not.
- Removes those features which is not Corelated to Target. (i.e; feature elimination)



After selecting features manually.

```
#y= dataset['survived']
```

we removed name, Ticket, Fare which Doesnot impact surviving.

So,

```
# x = dataset[['pclass', 'sex', 'Age', '.....']]
```

In few cases identifying manually is harder

So, now lets use EDA for feature selection.
↳(graphs)

for graph's we can use seaborn for graphs

```
# import seaborn as sns
```

we need to analyse from previous data that Male (or) Female is it impacting survival (or) not

```
# gender = dataset['sex']
```

sns.countplot(gender)

first it finds all Categories [like M/F] and gives there Count as plot/graph

sns.countplot(y)

gives how many survived and not survived based on Category i.e; [1 and 0]
1 → survived, 0 → not survived.

(or) we can use dataset['survived'] instead of y.

sns.countplot(dataset['survived'], hue = gender)

'hue' means count the gender who has survived.

(or)

sns.countplot(dataset['survived'], hue = 'Sex', data = dataset)

now, if Sex is changing survived then we can select that feature.

while selecting feature we have few records are null.

so, for identifying we can use isnull() from pandas.

dataset.isnull()

and for visualizing this null values we can use, headmap() from seaborn.

sns.headmap(dataset.isnull())

when we see black then that field doesnot have missing values. and vice versa.

- If we a lot of missing values we can eliminate feature.

- If values are missing and feature is important then we can keep near by value (or) average value this is called as imputation.

Ex:
```
def fillAge(cols):
    age = cols[0]
    Pclass = cols[1]
    if pandas.isnull(age):
        Print("Avg"); // substitute logic here.
    else
```

Some logic for substituting null values by Average.

Age=
```
# x[['Age', 'Pclass']].apply(fillAge, axis=1)
```

~~changed Age = x['Age'] Age~~

```
x['Age'] = Age
```

overides Age data in the dataset of x variable

- If only one record is missing and that is not much useful/important then we can remove that record.
- But like Bank Dataset in real world we should not do so.
- So, The process of doing (or removing null values is called Data cleaning.

08/11/21

- In stead of using oneHotEncoder and slicing the columns from sklearn.preprocessing library.

we can use get_dummies() from pandas module

Ex: x = ['sex']

```
sex = Pandas.get_dummies(x['sex'], drop_first = True)
```

here, we removered first column from male, female (or) female, male we remove first column. and give you one column to remove dummy variable trap.

i.e; To make all columns independent i.e; removing multicolinearity.

Now, convert all the categorical variables (columns) into non-categorical variables / removing dummy variable trap and make it single dataset.

- we can use concat() like hstack()

ds = pandas.concat([sex, pclass ...], axis = 1)

↓

concat horizontally.

By default concat does vertically.

Now, we have done

→ feature selection

→ Data preprocessing

- → Data Cleaning (removing null values)
  - → Data imputation
- → Label Encoding (option)
- → One-Hot Encoding for Categorical data. and removed dummy variables.

Now, we can use for creating model.

Now, we are finding a person is gona live (or) die i.e; 1/0 i.e; binary classification

So, for this we use Logistic Regression.

Ex: from sklearn.linear_model import LogisticRegression

Now, we create a model.

model = LogisticRegression()

Now, always split data into Training and testing data [as it goes randomly]

Ex:

from sklearn.model_selection import train_test_split.

X_train, x_test, y_train, y_test =

train_test_split(ds, y, test_size = 0.30)

Now, 70% goes to training and 30% we use for testing.

Now, fit the model.

model = fit (x_train, y_train)

- If we want to drop any one row if it is not important which has Null value (Nan)

use, ds.dropna()

Now, Check whether model is working properly (or) not

ypred = model.predict(x_test)

we can predict how accurate is the model by comparing y_pred to y_test and store that in count (after for loop) then take total number and divide them i.e; (count/total records) then you get the percentage of accuracy of model.

- If you want it to be done using function

Ex: from sklearn.metrics import confusion_matrix

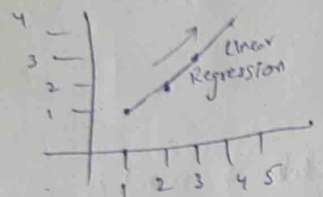
confusion_matrix(y_test, y_pred)

[TP FP
FN TN] → very dangerous value

True +ve False +ve
False -ve True -ve

10/11/21

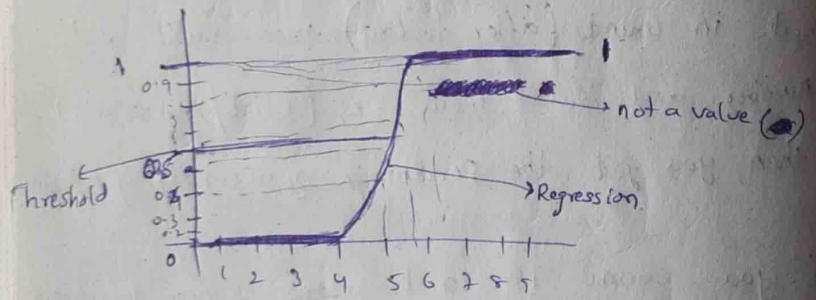
Linear Regression graph



formula → Linear function/equation ⇒ $y = b + \omega x$

$y \propto x$

Logistic Regression graph



So, for identifying this type of graph we use sigmoid function. (Range is 0 to 1)

above 0.5 can be considered as 1

below 0.5 " " " as 0

The formula is $\frac{1}{1+e^{-z}}$

where $z = b + \omega x$

- For Data Cleaning we can use ETL (Extract Transform Load) Tools also.

Computer vision

Image/video processing

library ⟷ openCV / opencv-python / python-opencv

module ← cv2 → functions

↳ videoCapture() (Connects to webcam)

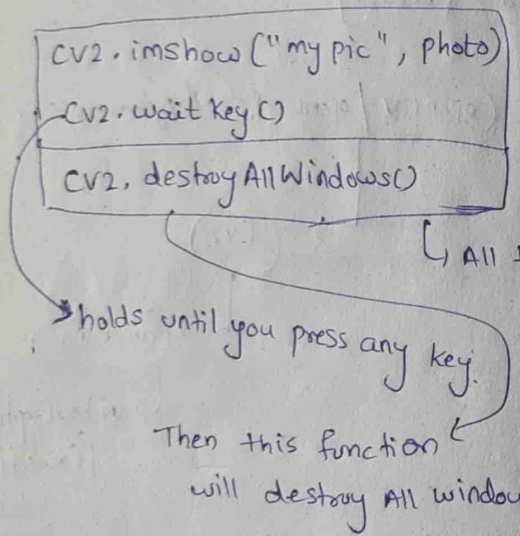
Ex: import cv2

Camera = cv2.VideoCapture(0) # selecting Camera

return, Photo
= Camera.read() # click Photo

Print(return) → return type • (True / False)

cv2.imwrite("Pic.png", Photo) # Saves Photo in harddisk.



we can specify time all

cv2.waitKey(5000)
↓
Time in seconds.

• Image → is a 3D array which holds [R,G,B] values within range (0-255)

You can create your custom image easily, by manupilating the array of 3D. by changing values.

These array of one [RGB] is called as single Pixel.

• [Camera.release()] use this when you want to release Camera so that other program can use. (light of camera goes)

Video :-

Continous images is called video.

Video Code :- (sample code)

```
while True:
    return, photo = camera.read()
    cv2.imshow("my video", photo)
    if cv2.waitkey(100) == 13:
        break
cv2.destroyAllWindows()
```

- 13 is enter key and the window ~~breaks~~ will close.

we can crop video also after capturing

i.e;

```
photo = photo[100:400, 150:450]
```

for making video as ~~black~~ Grey

```
photo = cv2.CvtColor(photo, cv2.COLOR_BGR2GRAY)
```

- In CV2 module it store array as RGB [B,G,R]

12	11	21

Computer vision

- For Computer to work like human we have given intelligence (A.I) by using Machine learning.
- Illy human eye. Captures continas images and sends to brain and immediately it identifies every object within image.
- But in Computer webcam acts like Camera but it cannot identify object for that we need to provide intelligent (or) give a vision of identifying objects.

Then that is called as Computer vision where we give brain to Camera and identify objects.

i.e; brain is called a model. (M.L)

which identifies objects from the images.

C.V → Pic → Object detection for this create a model.

- For capturing a video we have done already
- import cv2

```
camera = cv2.VideoCapture(0)
return, photo = camera.read()
cv2.imshow("pic", photo)
cv2.waitkey()
cv2.destroyAllWindows()
```

This is for capturing any image.

- For downloading precreated models use haar cascade frontal face.
- how to load a model after downloading, use.

Ex:- `model = cv2.CascadeClassifier("filename.xml")`

↓ any type as we can give to model

`pos = model.detectMultiScale(photo)`

↓ we provide our captured pic and now this function identify face is there (or) not.

- This function returns the co-ordinates of the face inside image.
- now you want to highlight the face we can you

Ex:-

`cv2.rectangle(photo, (50,50), (150,150), [0,255,0], 5)`

(50,50) → x_1, y_1 ; (150,150) → x_1+x_2, y_1+y_2

- (50,50) → Co-ordinates of rectangle i.e; (x_1, y_1) and (x_1+x_2, y_1+y_2) These two points.
- [0,255,0] → Colouring the rectangle
- 5 → Thickness of rectangle

Now, use imshow() to see the changes.

15/11/21

- When video camera is ON. ie; Camera is capturing images fastly.
- And every image is internally stored as arrays and those arrays are stored as binary values and those binary numbers are loaded on RAM as electric signals (Power ON and OFF)
- But here machine Cannot identify the objects in image for this we used M.L model for making it intelligent and to identify objects.

- Every image is 3D ~~array~~ array.

photo[: , : , :]
↓ rows ↓ clows ↓ specific columns.

Photo [: , : , 0/1/2]
↓
any one is used then the result will be in 2D array

so, if you need a photo to be in 3D then use

Photo [: , : , 2:3]
↓ second columns but o/p in 3D array.

↓

This get output as

[[[G1, G2, G3, -----]]]

- if you need last (3rd) & first Column inside 3D array

Ex:-

pic= Photo [: , : , [0, 2]]
rows, columns
↓
given you first & 3rd column

now, we can change photo into complete green

Ex:-

Photo [: , : , [0, 2]] = 0

ie; removed Blue and green values in photo.

- Instead of retriving B, G, R values from array operation we can use :-

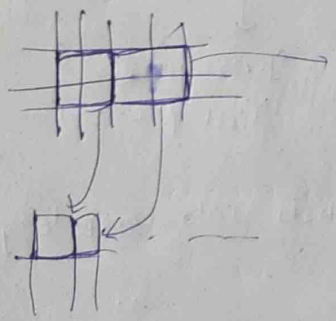
Ex:
```
B,G,R=
cv2.split(photo)
```

↓

which return 3 columns.

Making a picture blur

- here the concept is we use (or) merge multiple pixels by taking average of those pixels. and combine it and make a image.



how many pixels you combine that much intense the blur will be.

- This process of converting image is called Convolution.

- Even we can use blur()

Ex:
```
pic = cv2.blur(photo, (3,3))
```

↓

Combining 3 by 3 (3×3) pixels into single pixel.

```
import cv2

camera = cv2.VideoCapture(0)

ret, photo = cv2.read()
model =
cv2.CascadeClassifer("modelname.xml")

fdetect = model.detectMultiScale(photo)
```

detects face in above code.

- Capturing face in video/real time code

```
while True:
    ret, photo = Camera.read()
    fdetect = model.detectMultiScale(photo)
    if len(fdetect)==0
        x1= fdetect[0][0]
    if len(fdetect)==0:
        Print("face not found")
    else:
        x1 = fdetect[0][0]
        y1 = fdetect[0][1]
        x2 = fdetect[0][2] + x1
        y2 = fdetect[0][3] + y1
        rphoto = cv2.rectangle(photo, (x1,y1),
                               (x2,y2),
                               [0,255,0], 2)
        cv2.imshow('Pic', rphoto)
        if cv2.waitKey(100)==13:
            break
cv2.destroyAllWindows()
```

17/11/21

- How to use mouse on top of images (or) video that can be achieved using Callbacks (or) events.

- In python we know every variable assignment is a reference type

So, if you need some of data from variable we can share it to other.

Ex:- variable = photo[:,:,:]

↓

here we are passing data but not reference.

i.e, variable & photo has different addresses.

- we can also use copy(). ~~also~~.

Ex:

```
variable = photo.copy()
```

- imshow()

↓

In imshow internally it creates a empty window first and then inserts image in it.

ie; Ex:- cv2.imshow("new", photo)

This is equal to

```
cv2.namedWindow(winname = "new")
cv2.imshow("new", photo)
```

↓

This option first checks any window is there with this name (or) not if not it creates if yes then it uses that window.

- For handling mouse events, use:-

```
cv2.setMouseCallback("new", lw)
```

↓ its reference/name of function.

- indirectly setMouseCallback will call lw with 5 arguments.

```
def lw(x, y, z, P, Q):
    Print(x) → to get values of events on that window.
```

- In cv2 we have many pre created variables for events.

Ex:- cv2.EVENT_LBUTTONDOWN

- In setMouseCallback() first argument it calls using lw() is event and next two are co-ordinates. [x, y]

- In cv2 we have circle() for drawing circle

Ex:-

```
Pic = cv2.circle(photo, (150, 150), 50,
                 [0, 255, 0], 2)
```

(150, 150) ↓ Co-ordinates of circle; 50 ↓ radius; [0, 255, 0] ↓ Color of circle; 2 ↓ thickness of circle

- If you want to fill the circle then use thickness as -1.

One sample code for keeping a circle at clicked position on window/photo :-

```
def fw(event, x, y, i, j):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(photo, (x, y), 50,
                   [0, 255, 0], -1)

cv2.namedWindow(winname="new")
cv2.setMouseCallback("new", fw)
while True:
    cv2.imshow("new", photo)
    if cv2.waitKey(100) == 13:
        break
cv2.destroyAllWindows()
```

22/11/21

- CV2 treats colourful image as 3D image i.e; it stores [[[B,G,R,]...[]]]
- CV2 treats All Black & white images as 2D.
- All R, G, B values should be between [0-255] only i.e; 1 byte space.

  ↓

  $2^8 \rightarrow$ 8 bits.

So, its taking integer value and positive values only so, we use unsigned integer i.e; uint

- And we use 8 bits for every R, G, B so, in CV2 normal we can see datatype as uint8

  ↓

  uint8 → 8bit

  ↓ ↓

  unsigned integer

- import numpy as np

- arr = np.zeros((10,10))
  - Creates array with zero values.
  - 2D array (10,10).

- arr1 = np.zeros((10,10,3))
  - 3D array

type(arr)
- numpy.ndarray
- datatype

and inside values are by default float values. [ici arr.dtype]

So, for changing them,

- arr2 = np.zeros((10,10,3), dtype=np.uint8)

type(arr2)
- numpy.ndarray but values inside are integers

now, see

- import cv2
- cv2.imshow("img", arr2)
  cv2.waitKey()
  cv2.destroyAllWindows()

- arr2 has all zero's so, it shows black color. [R=0, G=0, B=0]
  - no color at all

- This output is shown in separate window and on top of window image is shown.
- instead of this try matplotlib.

Ex- import matplotlib.pyplot as plt

plt.imshow(arr2)

- Indirectly as cv2 converts image as numpy.ndarray and values as uint8.

Ex:-
```
import cv2
pic=cv2.imread("image1")
type(pic)
```
↓
numpy.ndarray
```
pic.dtype
```
↓
dtype('uint8')

- So while creating array using numpy always provide dtype=np.uint8

- when we read image using cv2 module it internally stores data as BGR format
so, if we use
```
# plt.imshow(pic)
```
then it shows differently. i.e., matplotlib stores array/pic as RGB format.

So, for this we have to convert BGR to RGB

So, use:-
```
# pic=cv2.cvtColor(Pic, cv2.COLOR_BGR2RGB)
```

- overriding image with another image

i.e., if we have Pic1 and Pic2
```
CPic1 = pic1[0:100, 0:100]
Pic2[400:500, 400:500] = CPic1
```
here, we took/cropped image with 0 to 99 rows and columns and we saved cropped image as Cpic1.
with that we overrided Pic2 image from 400 to 500 i.e., 99 rows and columns
(both rows and columns should match while overriding)

• Blending of two images (on top of another)

for this both pictures should be equal in shape.

• So for resphape we use

```
pic1 = cv2.resize(pic1, (1024,1024))
pic2 = cv2.resize(pic2, (1024,1024))
```

after this picture might distorted (squeeze)

• while placing on top of each other we can set how much percentage of image should be used.

i.e. pic1 80% (0.8) (alpha) + pic2 20% (0.2) (beta) + 0% (Gamma)

```
# blendimage = cv2.addWeighted(src1=pic1, alpha=0.8,
                               src2=pic2, beta=0.2, gamma=0)
```

Bitwise operation

Image → 3D

[R, G, B]

[127, 205, 37]

0111111 ---- ----

1byte 1byte 1byte

* So, every pixel takes 3byte on RAM.

Bitwise AND	OR	XOR
0 & 0 = 0	0 | 0 = 0	0 ^ 0 = 0
0 & 1 = 0	1 | 0 = 1	1 ^ 0 = 1
1 & 0 = 0	0 | 1 = 1	0 ^ 1 = 1
1 & 1 = 1	1 | 1 = 1	1 ^ 1 = 0

* So, now we can apply bitwise operations on those RGB then if we use 'not' operation.

lighter colors become darker,

i.e (white → black, light blue → dark blue)

```
pic_rev =
# cv2.bitwise_not(pic)
```

- 'not' means every Color will be converted opposite (ie; R, G, B)

- If you need to do and operation

picture = cv2.bitwise_and (pic1, pic2)

- Indirectly when we see the operations are done on 0's & 1's

Masking :-

When you want to change Color we can use this.
(ie; in medical field to identify disease we use this).

- Code :-

```
# import numpy as np
    low_green = np.array([50, 50, 50])
    high_green = np.array([25, 255, 255])
```

```
import cv2
camera = cv2.VideoCapture (1)
while True:
    ret, photo = camera.read()
    photo = cv2.cvtColor(photo, cv2.COLOR_
                         BGR2HSV)
    mask = cv2.inRange(photo, low_green,
                       high_green)
    cv2.imshow ("hi", mask)
    if cv2.waitKey(100) == 13:
        break
cv2.destroyAllWindows()
```

This can be helpful in object detection.

24/11/21

Image/video processing + AI (ML/DL) = Computer vision.

- In grey photo the array will be 2D array i.e; white & black. Combination number will be a single pixel.

- For making as grey we use

```
# grey_pic = cv2.cvtColor(photo, cv2.COLOR_BGR2GRAY)
```

and to plot grey in matplotlib use

```
# plt.imshow(grey_pic, cmap='gray')
```

- how to give filters to images

$$\begin{bmatrix} 160 & 175 & 145 \\ 170 & 150 & 175 \\ 165 & 170 & 100 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & -2 \end{bmatrix}$$

↓

3×3 matrix is multiplied with some values of matrix then we get some filters

- That matrix we use is called as Kernel matrix.

- This matrix will be having a specific pattern (or) values to add filters to original image by multiplying 3×3 values of image pixels with 3×3 kernel matrix. This process is called convolution.

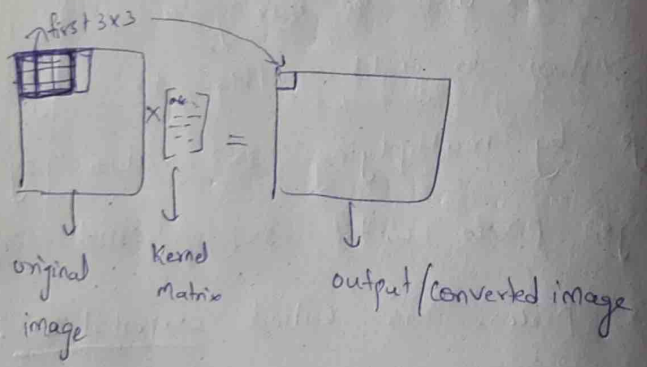
website to play with this process

https://setosa.io/ev/image-kernels/

- If we need a Kernel matrix and multiply with our image we get blur
- That values of kernel matrix is :-

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

- i.e; after multiplying that result image one pixel will be equalient to 3x3 pixels of the original image.



- Kernel matrix values goes to 0 it will become darker and close to 255 will be lighter.

- mostly we use 3x3 kernel matrix.
- And while choose 3x3 in original image and next 3x3 pixel will be picked.

So, that is called stride.

[i.e; we can decide next 3x3 (or) next-next 3x3 pixel to be taken for multiplication]

- After multiplying those values are added and become one pixel.

Ex: Code :-

```
import numpy as np

Kernel1= np.ones((3,3))/25

ret, Photo = Cam.imread camera.read()
                                  ↳ make it 2D by using grey.
Pic = Cv2.filter2D(src=photo, ddepth=-1,
                   Kernel=Kernel1)
```

- So in cv2 we have blur() internally it multiplies with kernel matrix and sum the all 3x3 pixels and make it as single pixel and so. on---

Ex: cv2.blur(photo, (3,3))
↓
kernel size

- while you increase the kernel matrix size the more blur will photo become.

Ex: after multiplying we can do median instead of sum

Ex: cv2.medianBlur(src=photo, ksize=3)

Threshold

- Converts image into black & white (or) gray
- we use cvtColor() to convert color instead we can convert directly by

photo = cv2.imread('pic.jpg', 0)
↓
flag value for gray as 2D array.

- So, from
  0-128 will be mostly blackish color
  129-255 will be mostly whitish color

we need to remove white for reading black text from image.

for converting,

cv2.threshold(photo, 128, 255, cv2.THRESH_BINARY)
↓ ↓
Threshold maxval

29/11/21

Lets See Invisible cloak (or) Lets See the concept of morphological.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

blank_img = np.zeros((600,600))

plt.imshow(black_img, "grey")
```

- want to keep text on top of that image

```
black_img1=
    cv2.putText(blackimg, text='ABCDE', org=(50,300),
                fontFace = cv2.FONT_HERSHEY_
                SIMPLEX,
                fontScale=5, Color=(255,255,255),
                thickness=25)
```

(org: co-ordinates; Color: white color)

- detecting objects in a picture is morphological (i.e overlapped objects)

- For removing smaller noises we uses smaller Kernel side.

```
Kernel = np.ones((5,5), dtype=np.uint8)
```

- For making boundaries apart in image (blank_img specifically) we use erosioning. (characters)

```
erode_img = cv2.erode(black_img1, kernel)
plt.imshow(erode_img)
```

- In erode function when we have 5x5 pixel with black it doesnot do any thing but if we have some part with white and some part is black then it gives more prefrence to black.

- If we want to do eroding again and again that much white and black pixels gets removed.

Ex:-
```
erode_img = cv2.erode(img, kernel, iterations=3)
```
(black; 3 times it does)

- In dilution it does opp. to erosioning.

Ex:-
```
dilute_
Pic1 = cv2.dilate(img, kernel, iterations=4)
```

- In dilution it gives more preference to white and less to black.

- Similar type of operations done on image can be used using:-

```
morph_img =
    cv2.morphologyEx(img, cv2.MORPH_OPEN,
                     kernel)
```

- does erose and dilute.
- similar type of variables can be used for removing noisy data.

```
close_img = cv2.morphologEx(img, cv2.MORPH_CLOSE,
                            kernel)
```

- does dilute and erosion.

Similary, we can use MORPH_GRADIENT for checking the boundaries of Image.

- For making invisible,

- First take background image without you present on that image.

- And now you will be taking a live Cappra you will be on the top of the background image.

So, when you try to remove the entire body with replacing the background image pixels the we become invisible.
this can be achieved by masking.

- So, here we need to identify the entire outline of our body is very difficult.

So, what we can do here is we can you take help of cloth (with single color) So, we can defect that single color cloth and then substitute cloth color pixel with background image pixel.

Then we get invisible.

- check code in google drive.

01/12/21

Deep Learning

- Machine learning is older way.
- Newer way is Deep Learning.
- All the M.L problems can be solved using Deep learning but not vice versa.
- Algorithm behind Deep learning is Neural Network.

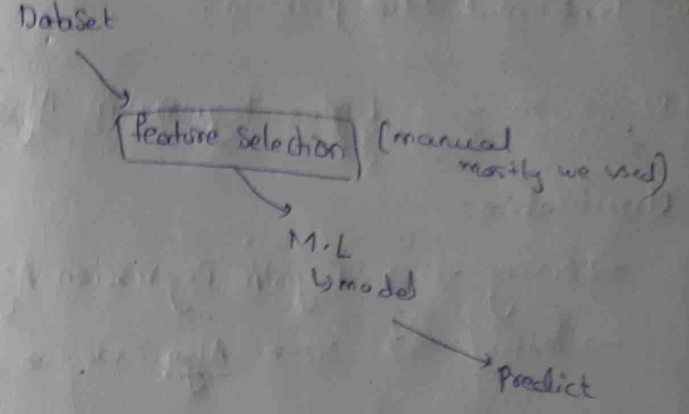
Machine Trained
↳ M.L (model) → uses function
↳ finds bias, weight

old
(~~Traditional~~ to M.L)
(Traditional M.L)
↳ Algorithms are mostly statistical (or) mathematical way.

new
↳ Deep Learning
↳ Algorithms are mostly use similar to human brains. (how brain Learns)

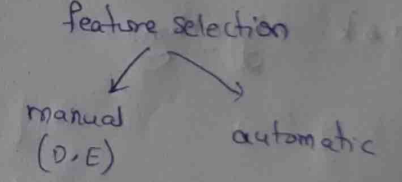
Ex: $y = mx + b$
↓
Just finds values.

Traditional M.L



- Feature selection is one of the most important phase.

→ mostly done by Domain expert but if there are billions of features/fields identifying is very difficult.



→ so, we need to use auto selection by using Neural network Algorithm (NN).

- In Traditional M.L problem:-

→ Regression ⟶ Linear Regression Algo

→ Classification:-

→ Binary ⟶ Logistic Regression Algo.

→ multi ⟶ Some Algo we can use

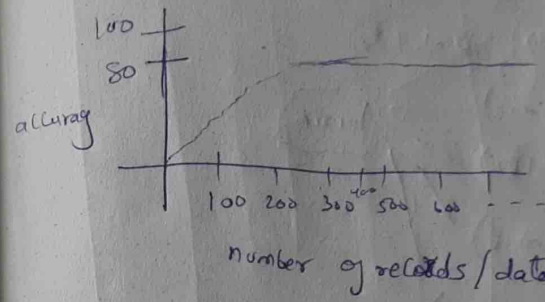
- In Traditional we use multiple Algo's for multiple useCases.

But in NN Algo. it supports for many problems.

- one Algorithm for many problems ie; Neural Network Algorithm.

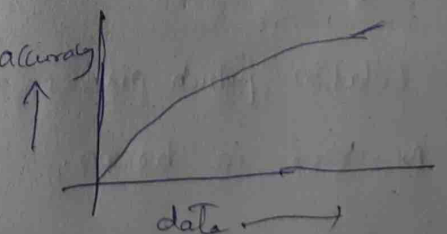
- In Traditional M.L when we increase data it increases its accuracy; but at certain point the model accuracy becomes constant.

This is another disadvantage.



The data given to model increased but the accuracy stopped at 80 and gives almost constant accuracy.

* But in N.N - when we give huge data / big data the model gets more and more accurate.



Neural Network

- It works exactly / similar to human.

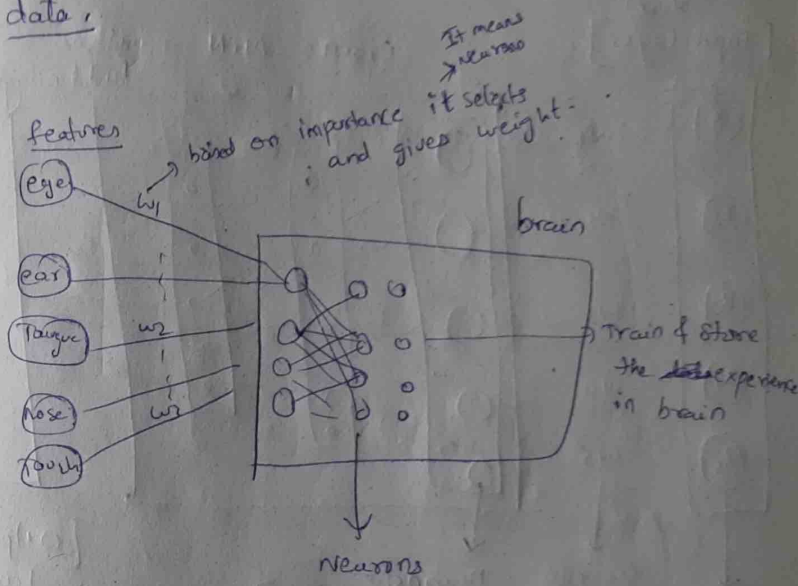
- how human gets data.
  i.e; 5 Sensory organs.

  → eyes (seen) — Sends data
  → ears (hears)
  → Tounge (Talks) — brain
  → nose (smell)
  → Touch (Touch/feel)

- now, data comes to brain, but brain doesn't store every thing,
  It takes very few features [few pixels will be taken but not every pixel] which is use to identify any object.

- This feature selection [which pixel is import] is done by neurons in brain.

- So, after selecting features with help of neurons those data is given to brain.
  i.e; brain only stores experience, but not data.



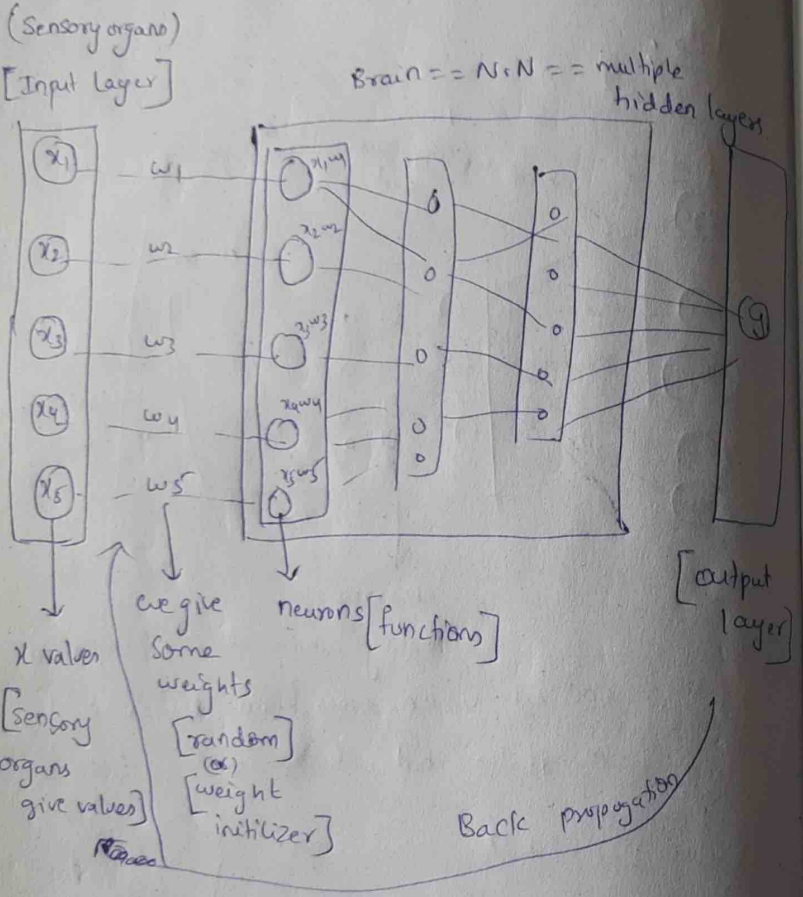
identify which data is important (or) not which comes from senses and eleminates that features

- For this we can use a library called Keras.

/> pip install keras.

03/12/21

N.N Algo.



The over all think is we need to find the weight.

So, for finding weight we Initially give random weight (or) weight/kernel Initilizer with different features (serces)

- Those weight and features are given to a function [neurons].

and after some operation and then we check with real answer.

if it is okay then no problem but if it is not same [checking with real output] we need go again and change the weight. and do again same prosn.

ie; Called back propogation.

[weight reinitilization].

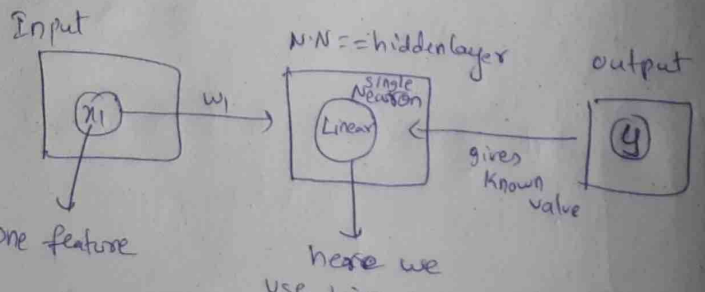
- we always check the value with original value i.e; finding error..

- as long as minimizing error means we are selecting right weights and approaching to accurate output/prediction.

- Here while going back propogation again and again changing the weights will be very difficult for large data.

So, we use a concept of optimizer and change the weights accordingly.

- what (or) how Neuron function has?



if data is like

x_1	y
1	10
2	20

→ The above single neuron function will use linear Regression model.

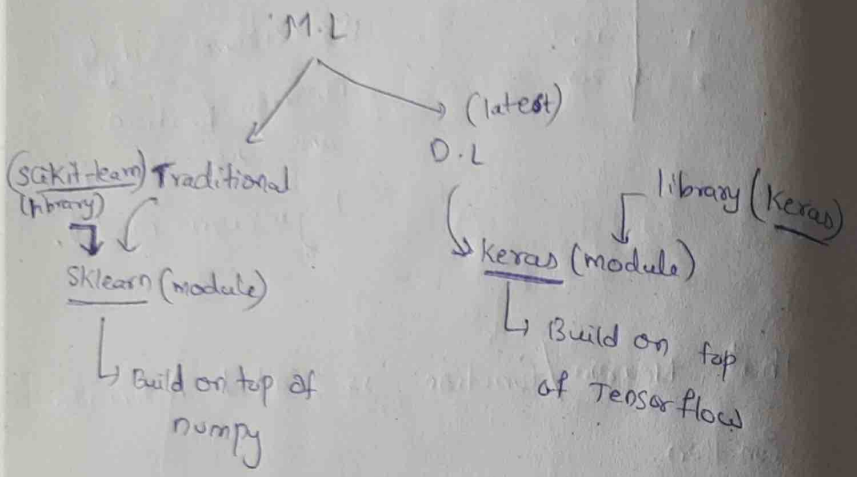
- That Neuron/function is called Activation function.

- III'ly, we can use sigmoid function, Logistic function for single neuron.

- we do same traditional M.L model/function in a single neuron function.

- And for back propogation we use error/cost/ Distance can be finded by Mean Absolute error.

06/12/21



- When we have less data [i.e, around thoushands or lakhs of records] (or) less fields/features use Traditional M.L model.

(or) else use Neural network algorithm.

- for now, humans have 85 billion neurons i.e, each neuron has some function.

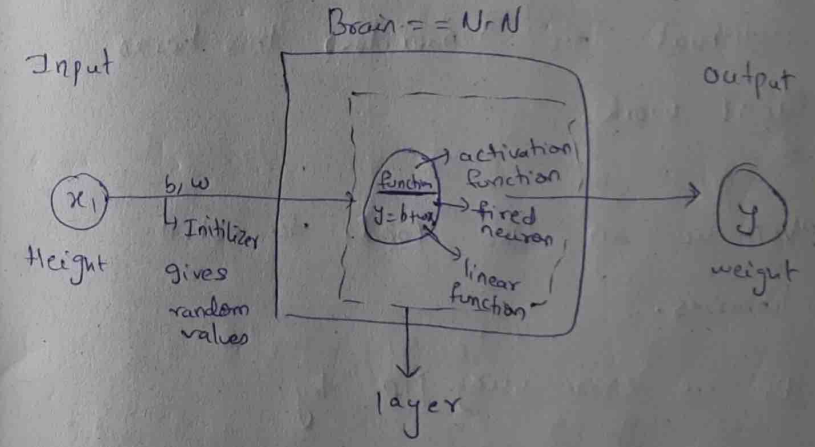
- Based on what functionality that respective neuron automatically activates.

- normally also we say a neuron fired i.e also called as activated a neuron.

may be that's why a single neuron is called activation function.

- So, now we need to decide how many neurons/functions to be created and how many unit of functions/neurons does some operations. [which model to be used in function]

Example of weight dataset



- mostly we use same/similar type of model for most of the N.N problems.

- import pandas as pd

```
y = dataset['weight']
x = dataset['height']
```

from keras

- pip install tensorflow

without this tensorflow this Keras wont work,

- even we can use 'conda' to install libraries.

If we error using pip try

- conda install tensorflow.

- from keras.models import Sequential

- model = Sequential()

↓

This is for saying sequential neuron layers are used in our model.

- from keras.layers import Dense

- The more layers the more accurate. So we use Dense.

- To check how many layers we have

  - model.get_config()

- Now lets add layers. So, we use Dense function to provide how many layers, activation needed and what function using add()

→ model.add(

single neuron → Dense(units=1, activation="linear", with linear function

Single Layer

kernel_initializer = "zeros",

weights initializers for features

bias_initializer = "zeros",

input_shape = (1,)

how many features r we are giving as input.

))

• model created now lets train.

→ model.summary()

↳ See summary.

• now, we need to give model the error function from which the model gets trained and finds/predict again and again and gives less error output.

→ model.compile(loss = "mean_squared_error")

↓

we can see in online on keras loss functions.

So, even for identifying which/how much weight to be given for next weight is done by optimizer.

→ model.compile(loss = "mean_squared_error", optimizer = "Adam")

↓

we have many optimizers.

now, train the model.

↓

→ model.fit(x, y)

and check the weight and bias

→ model.get_weights() ←

- we can predict like

  ⇒ model.predict([100])

- we can share model using

  ⇒ model.save("first_model.h5")

  ↓ any extension

- we can check how many input and output for single layer

  ⇒ model.get_layer("dense").input

  ⇒ model.get_layer("dense").output

10/12/21

- The main problem is identifying features for this we use Neural Network. which identifies similarly like human. so, in computer world we say Artifically so, we call as Artifical Neural Network.

- for this we are using pima diabetes dataset and build a ANN model.

Ex:

```
import pandas as pd
dataset = pd.read_csv('name of dataset.csv').
```

↓ Pima-indians-diabetes.data

//Pandas by default takes first row as column names.

for this we dataset we don't have column names so, we can use

```
dataset = pd.read_csv('file.csv', header = None)
```

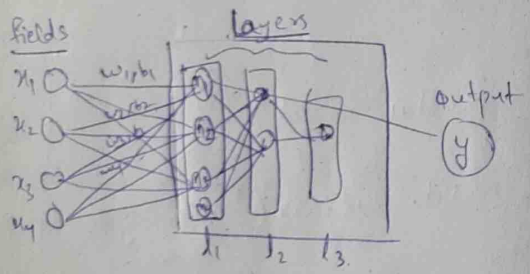
↓ This treates first column as record

- This gives column names as 0, 1, ---- as many as columns

- Check from where dataset is downloaded and identify the names of columns which represent 0, 1, 2, 3 - - - columns.

→ y = dataset[8]

x = dataset [[0, 1, 2, 3, 4, 5, 6, 7]]



mostly its better to have same number of neurons as same as fields/features.

and we don't use 1 neuron because few fields may have zero weights for best (or) least cost function/error but in real world it might be useful

So, we will be not considering single neuron instead we use multiple neurons.

- The number of layers we increase (or) how deeply we learn (or) train model by increasing layers.

i.e, indirectly increasing random values of weight and bias understanding that features based on those values which become Deep Learning.

→ which gives more accuracy.

→ from keras.model import Sequential

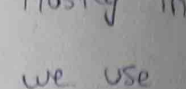
model = Sequential() → creates empty brain [within this we need to add layers]

model.get_config() → which shows how many layers are these.

```
from keras.layers import Dense
```

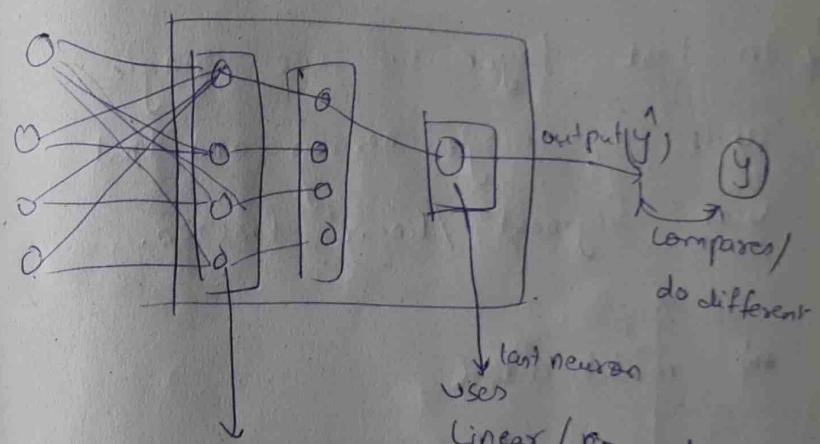
for adding layers we use dense function.

```
model.add( Dense(
    units = 5,                  → neurons are '5'
    input_shape = (8, ),        → dataset has 8 columns
    kernel_initializer = "zeros",   → at first the weight will be zero i.e;
    bias_initializer = "zeros",     → even at first we use zero for bias values
    activation = "relu"
    )
)
```



- Mostly in the last neuron / layer we use only single neuron and that neuron will use two functions mostly i.e;

Linear Regression (or) Binary classification.



(This does one thing value which is less than zero is treated as '0' and greater values are treated as them self).

→ model.summary()

* here we used only single Dense() which means we have single layer.

we can add() multiple layers to the model.

* In last layer we use single units ie; single neuron and that uses "sigmoid" / 'linear' function.

→ model.add(
    Dense(
        units=1,
        kernel_initializer="zeros",
        bias_initializer="zeros",
        activation="sigmoid"
)
)

- DO model.summary()

and check layers, weights & bias/parameters.

* Mean square error is used in Regression (loss function)

In Binary classification we use Binary Crossentropy class (loss function.

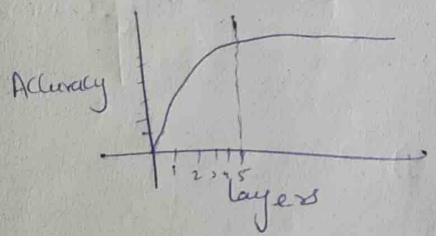
→ model.compile(optimizer="adam",
    loss="binary_crossentropy")

→ model.fit(x, y)

15/12/21

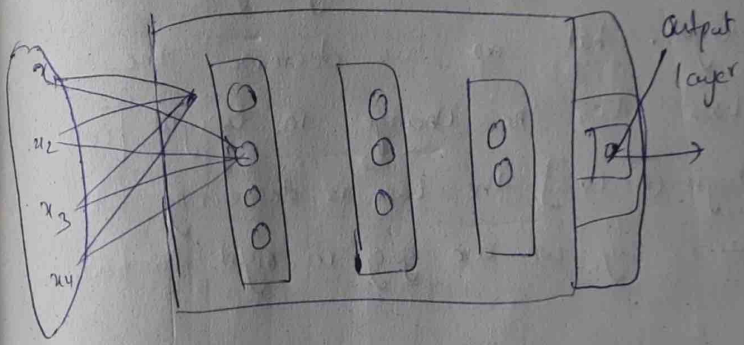
- The more layers and neurons more accuracy will be there but it takes more computing time.
So, make sure to be as much as we need that make more accurate and try to use less layers and neurons.
[Don't believe in formulas in google to how many we have to use]

- The place when accuracy become almost constant at that time we can stop creating new layers.



- So, here the Data scientist come in role and identify at what layer the accuracy will be constant.

- So, here we can use Devops for automating of identifying the layers so, we can says MLOPS (Datascience + Devops)



we mostly use same neurons ~~for~~ (or) less than same neurons from layers to layer.

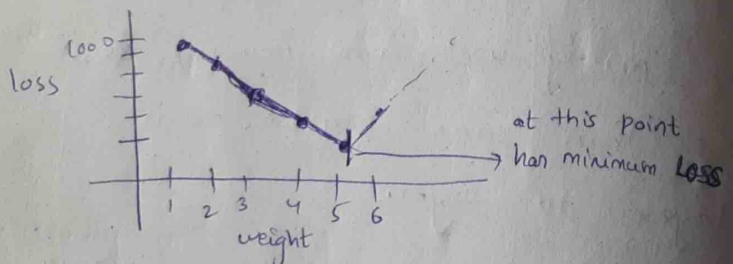
- And at last i.e; output layer we mostly use Binary classification i.e; single neuron. [sigmoid / linear regression Activation record]

- we know L.R function

$y = b + wx$

we identify these two values based of Loss/error function

for initial value we might give 1 (or) 0 weight. but as we decrease the Loss i.e; the change in weight [i.e; high (or) Loss] the Loss is decreasing we can say we are going in right direction.



- Direction is called as Gradient and when we come down we say Descent i.e; [Gradient Descent] G.D

In above graph when weight increase loss decreased so we can say [G.D] Gradient Descent.

- when we are changing weight & bias based on the error/Loss function.

- The rate (or) changing those values i.e; [0, 1, 2, --- (or) 0.1, 0.2, 0.3 (or) 10, 20, 30 ---]

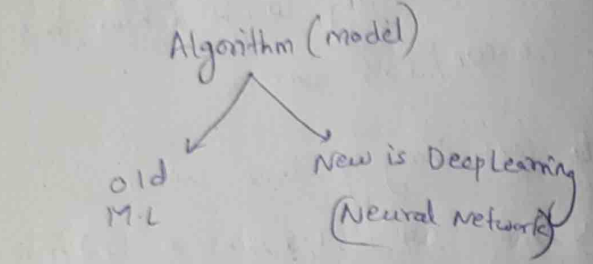
The steps you are changing and see the Loss function this is called as Learning rate.

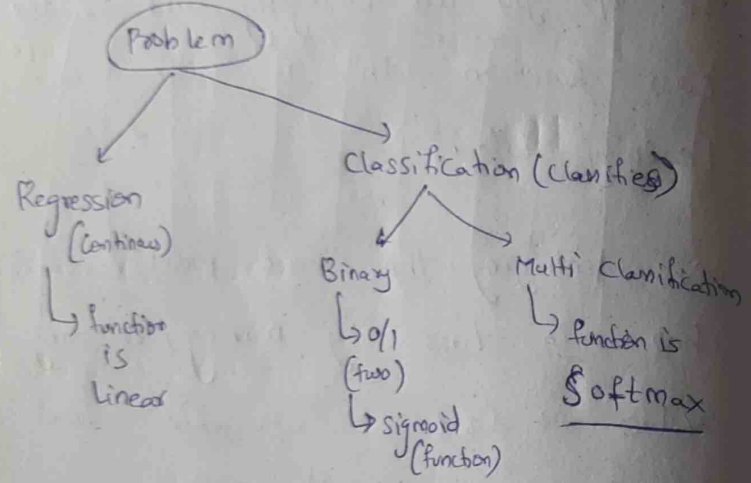
- i.e; identifying the exact position which has min. Loss by moving/selecting values as fast as possible that is said as learning rate.

- when we do high/Longer Jumps to go faster to min. Loss we get lesser accurate min. Loss/error/output

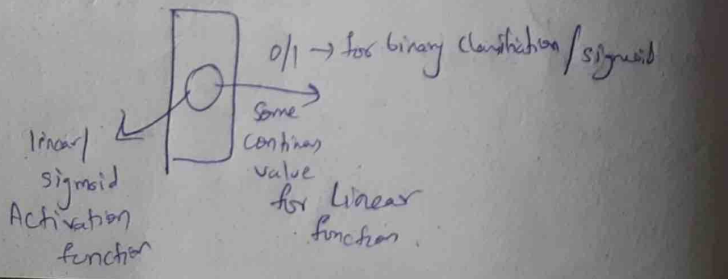
So, Learning rate should be normal. not faster (or) Slower.

17/12/21

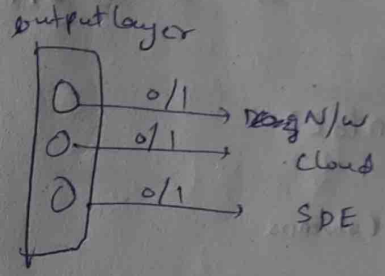




- Mostly in linear & Sigmoid function we get a single output for respective input. in the last output layer



But in Multi classification we have multiple outputs so we need to have multiple neurons, which specify the specific classifier and gives output in range (0, 1) i.e; if a specific neuron gives 0 i.e; that classifier has nothing. and vice versa.



So, here if we get SDE as 1 then we can say there is more probability of getting that Job.

- For multi classification sir used "wines.csv" data set.

→ import pandas as pd

→ dataset = pd.read_csv('wines.csv')

- here y.column is fixed but it has more values to be calculate

→ dataset.info()

→ dataset.columns

→ y = dataset['class']

→ x = dataset[[,]]
rest columns

In pandas we have

→ y.value_counts()

which counts how many categories are there.

Till now we did dummies variables for x,

now we have to do for y,

So,

→ y = pd.get_dummies(y)

In x we have dummy variable trap and here we don't have because x are independent variables but in y one variable cannot find another variable.

So, we dont have dummy variable trap.

- for specifying how many input neurons should be used ie; number of columns. ie;

→ in_shape = x.shape[1]

- for checking number classes in visual way use seaborn and compare y with class column.

now, let build model

```
# from keras.models import Sequential

# model = Sequential()

# from keras.layers import Dense

# model.add(Dense(
        unit = 6,
        input_shape=(in_shape,),
        activation = 'relu',
        kernel_initializer = "he_normal",
    )
)
```

by default it uses Glorot/xavier kernel initializer even if we don't specify

```
# model.get_config()

# lets create another layer

model.add(Dense(
        units = 4,
        input_shape = (in_shape,),
        activation = 'relu',
        kernel_initializer = 'he_normal',
    )
)
```

input takes from previous layers

Copy paste for new layer.

```
# output layer

model.add(Dense(
        units = 3,
        activation = 'softmax',
    )
)
```

we have 3 classifier in our dataset of y

- model.summary()

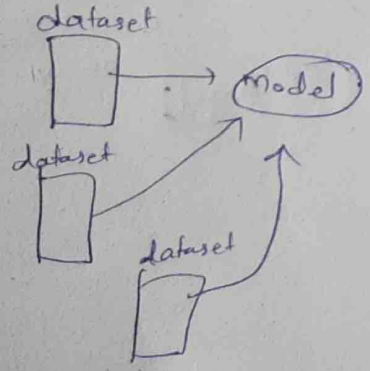
- model.compile(optimizer="adam", loss="categorical_crossentropy")

- model.fit(x, y)

- As human can learn more when we read again and again

Similary when we get dataset to model again and again it ~~gets~~ trained more accurately.

This way of giving data how many times i.e; epoch Iteration



So, here epoch = 3

- model.fit(x, y, epochs = 100)

↓

See output at which epoch value the loss is constant.

So, Change epoch accordingly.

- All the loss is stored in

myloss =

- model.history.history['loss']

- type(myloss) → list

- myloss_final = Pd.DataFrame(myloss)

- type(myloss_final) → Pandas.Core.frame.DataFrame

↳ it converts into table

- myloss_final.plot()

- If we need accuracy also within compile() then,

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

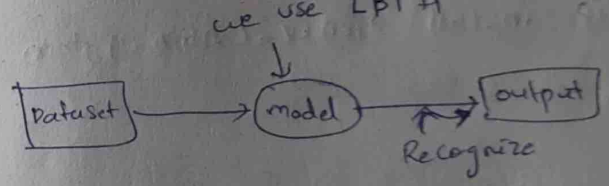
- now after fit we can see the model accuracy

22/12/21

CV

- Till now we have detected face now, lets see how to Recognize face.

- For this we need to give the exact Cropped part of our face with multiple angles and this Cropped face will be removing noisy data around you so, model will be accurate.



LBPH (Local binary patterns Histogram) which will Recognize the person.

- First we Capture lot of pictures of Our's and convert those pictures into Black & white [which will take less space]

and those picture are used to identify face and crop that part and store in a folder.

These images became input to the LBPH model.

- See code in sir drive

- For cv2.face_LBPHFaceRecognizer.create() to work install library.

ie; pip install opencv-contrib-python

24/12/21

Advance Deep Learning Concepts

→ Tuning
→ Regularization
  → Dropout
  → L1/L2
  → (Lasso/Ridge)
→ K-fold
→ Cross-validation
→ over fitting.
→ Generalization.
→ Batch
  ↳ SGD (Stocastic Gradient Descent)

- There is one challenge when we split the data as training and testing data. at that time when training data is almost same and the W, b will be almost same but if we have totally different testing data we will have different b, w this issue is called as Over fitting.

Ex:- hrs marks

1	10	} → training data
5	50	
4	40	
2	60	} testing data
3	90	

here

$w = 10$

$y = b + w x$

$10 = 0 + 10(1)$

In this case we have totally different data so, we face issues.

This is issue is called as overfitting.

- Any model which is Generalized then it might not be accurate but it is more reliable. [this will resolve overfitting]

- when we have issue with the data ie; overfitting then we won't have more accuracy of the model.

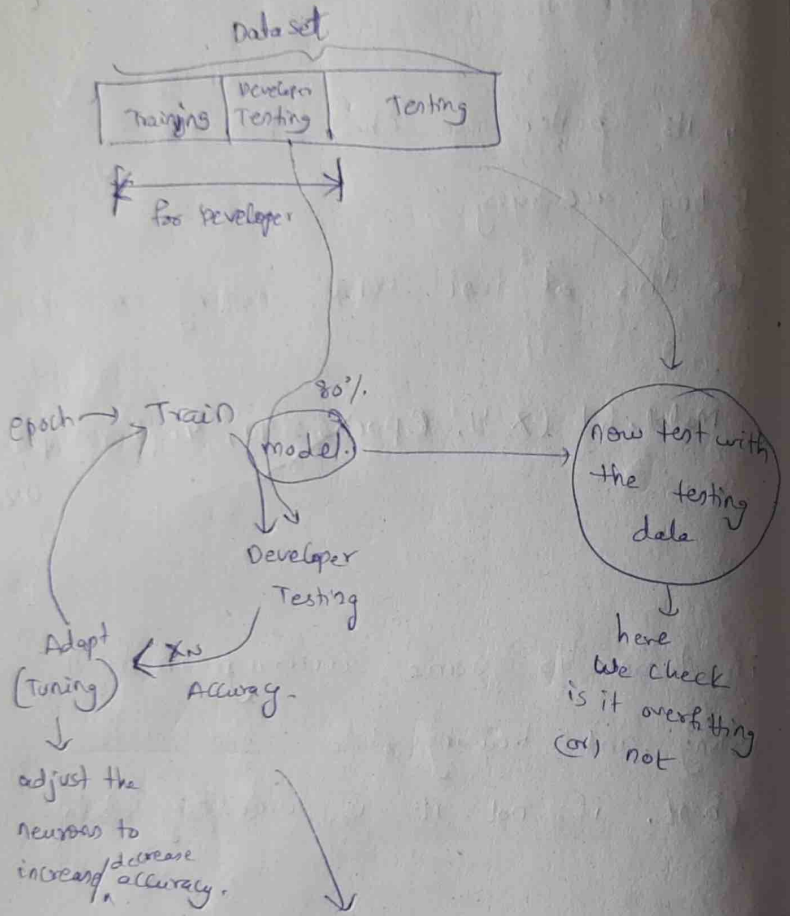
so, its better to check both training and testing accuracy.

for this at last while fitting the model

model.fit (x, y, epochs = 90, validation_split = 0.20)

if there is same accuracy in both testing and training data then it is best. if not it is overfitting data.

* In real world we do multiple testings.

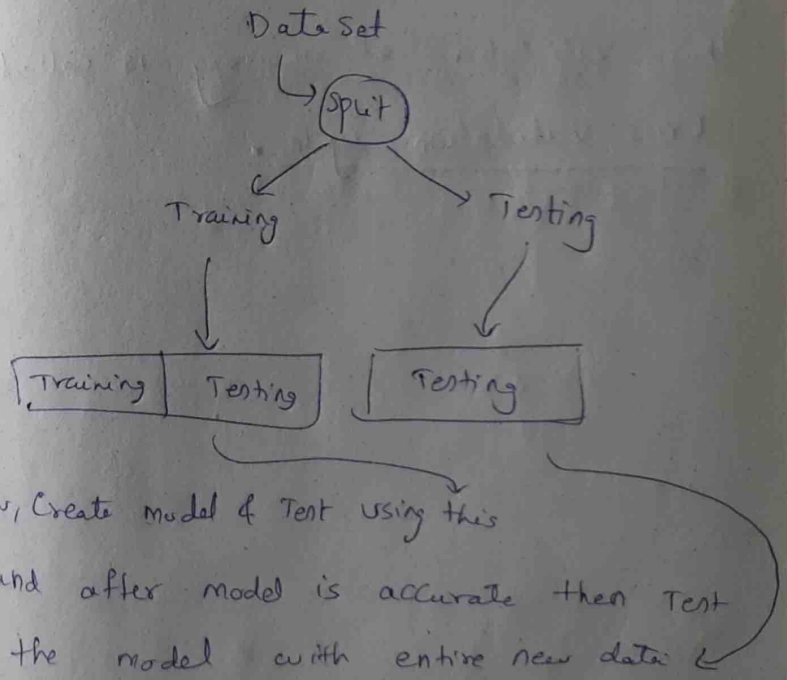


. The Developer Testing data is Called as "validation - spit"

. we always/mostly divide dataset into 3 parts. [Training, Dev Testing, Testing]

Till now indirectly we did first two only.

But for real data what we have to do is:-



Now, Create model & Test using this and after model is accurate then Test the model with entire new data

- That Dev Testing is like validation we if accuracy is not good we train again and again. and we increase accuracy.

- And after it has best accuracy with the Dev Testing data we check with actual Testing data and if model doesn't overfit with the new accuracy then the model is great.

- This validation of accuracy is called Cross-validation cycle.

27/12/21

- When we have lot of difference in accuracy of training testing data and testing data then we can say we have overfitting

- To overcome this we can use:
  - → Dropout
  - → Regularization.

- For this lets see iris data set.
  (we can download (or) we can import from sklearn.datasets module)

```
from sklearn.datasets import load_iris
iris = load_iris()
iris  # see all the dataset
iris['target_names']  # types of iris flowers
iris['data'] → complete dataset
```

→ iris.feature_names # gives column names

Lets store this dataset in the format of table (DataFrame) which is provided by pandas.

→ import pandas as pd

→ df = pd.DataFrame(data = iris.data,
Columns = iris.feature_names)

→ Lets add "y" to the dataset so that we get a clear idea.

→ df["label"] = iris.target

↓

we can give "y" also

now, df is complete dataset and now, we can divide X and y.

[we can directly also use instead of adding to dataset]

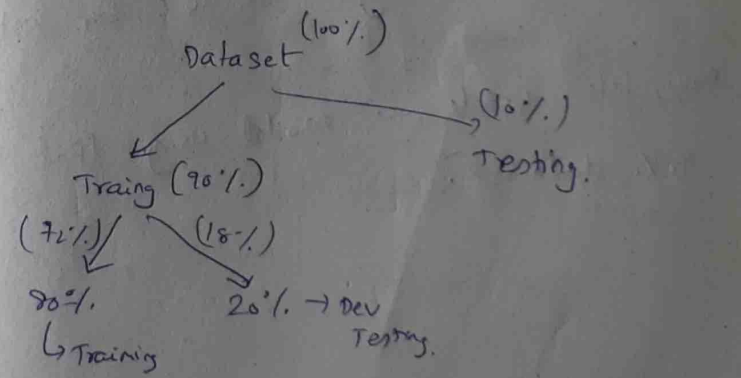
→ y = df['label']

→ X = df[['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)']]

'y' has multiple values so, we have to do one-hot encoding.

This is y so, we don't remove any value / field. [all are independent]

→ y_final = pd.get_dummies(y)



now, lets split,

```
from sklearn.model_selection import
                        train_test_split

x_train, x_test, y_train, y_test =
    train_test_split(x, y_final,
                     test_size=0.10,
                     random_state=42)
                        ↓
                     Taken values/rows
                     randomly.
```

now, we have done
90% for training and 10% for testing

now, lets create a model using that 90% data set.

```
from keras.models import sequential
model = Sequential() # instead of this we can use
from keras.layers import Dense
model = Sequential
model.add([ → we can add all layers as one using list.
    Dense(units=32, activation='relu',
          input_shape=(4,)),
    Dense(units=32, activation='relu'),
    Dense(units=16, activation='relu'),
    Dense(units=16, activation='relu'),
    Dense(units=8, activation='relu'),
    Dense(units=8, activation='relu'),
    Dense(units=3, activation='softmax')])
```

we can keep this model in function also.

- model.summary()

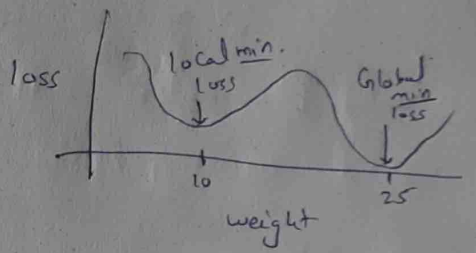
all layers got created.

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

history=
model.fit(x_train, y_train, epochs=200,
          validation_split=0.20
          batch_size=20
          )

history.history['loss']
```

- Selecting single record at one point of time and giving it to the model and it does forward propogation and backward propogation until you find Global min. loss. using all records.



This Global min loss can be achieved only at single batch approach.
(selecting single record/row at one point of time)

But it takes a lot of CPU Time.

- This way of optimizing is Called SGD (Sarcastic Gradient Descent)

- To Reduce Time / CPU Time we Can use min. batch of records (ex: 5) and give to the model.

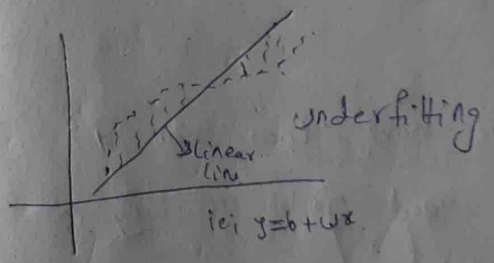
30/12/21

- As we have done testing/finding accuracy in training test as well as testing data.

So, when the model doesn't give nice accuracy in training Data then we say High Bias: (It is just a term not related to $y=b+wx$)

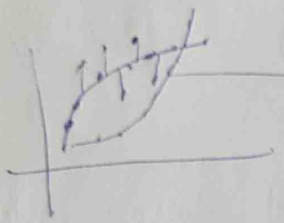
And when accuracy is not good in Testing Data we say high variance.

we can see in fig.



Mostly we don't have linear model as it will not fit with all data points and model will not be accurate so, we use non-linear (polynomial) Model.

- when we use this non-linear Model we get Low Bias



any one of the line is possible. (non-linear)

- we change a lot of values ie; specifying the no. of layers, optimizer and many parameters these are called as hyperparameter / metaparameter / tunning parameters.

- when we have Low Bias ie; we get good model accuracy in Training data.

But if the same model accuracy is not good at Testing data we get high variance and called as overfitting.

- Totally we can say that when accuracy is good at Training data we say low Bias. and if accuracy is good at testing data we say as low variance.

- So, if any model has low Bias and low variance then we can say it is not overfitted.

- And if accuracy is good at Training data then we can say it is not underfitted.

- Good accuracy at training data → not underfitted

- Good accuracy at traing data and not at testing data → overfitted

- Good accuracy at both then → It is not overfitted and underfitted (Good model)

- To Solve over fitting we use Regularization → L1 & L2 (Lasso) (Ridge) and Dropout

- Dropout:-

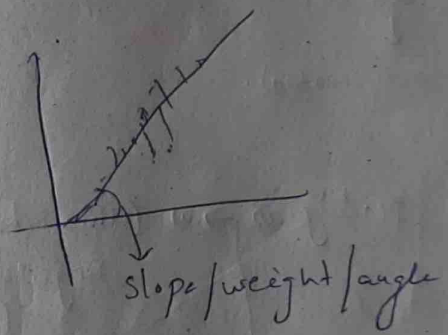
Every layers has neurons based on value of neurons the model is trained so, that first in training data accuracy is good as it remembers and when coming to testing data they fail.

So, for this we can limit the number of neurons to be used in a single layer/layers and for every epoch (cycle) backpropagation) we can change/use other neurons.

This dropping of neurons is called Dropout.

- Even model will be trained faster and removes overfitting and gets generalized model.

- L1 Regularization has one of feature is over fitting and even has/does feature selection (now we are not using)



we should decrease this weight for this we always take the help of loss function ; i.e; mean absolute error.

- But this work for only training data and for testing data it might not work so, accurately so, it again comes to overfitting.

So, to reduce this we are going to add penalty to the MAE (or) Loss function so, that we can reduce the overfitting problem.

$$\text{Loss} = \text{MAE} + \lambda \sum w_i \rightarrow \text{penalty}$$

λ ↓ Some constant.

when we add penalty as → w (This is L1 Regularization (Lasso))

→ w^2 → (This is L2 Regularization (Ridge))

- For code see "Regularization in keras" file in sir drive.

- In this code we have imported dropout and l2 Regularization.

In single layer we have kernel_regularizer parameter.

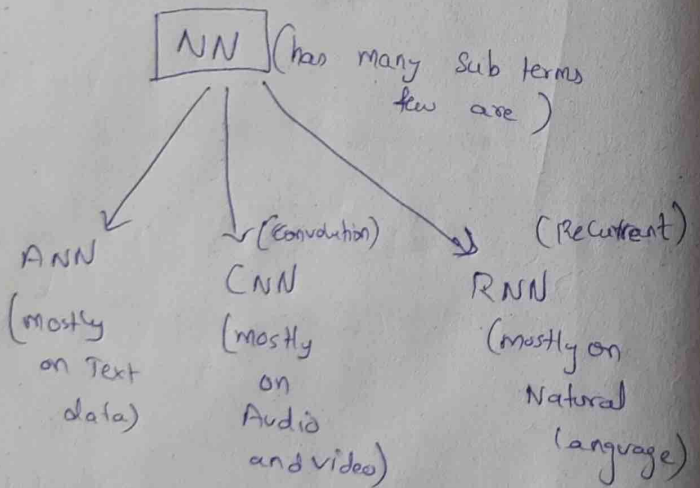
and after every layer we can use Dropout.

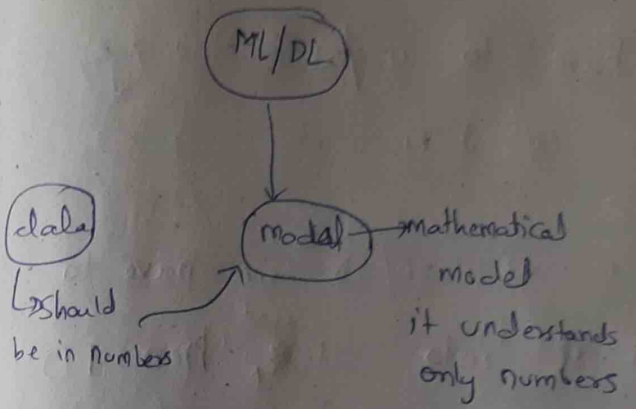
- for l2 we give factor variable i.e; value of λ (lamda).

03/01/22 CNN

- Till now we use Artificial Neural Network which on Data as Text (excel, csv)

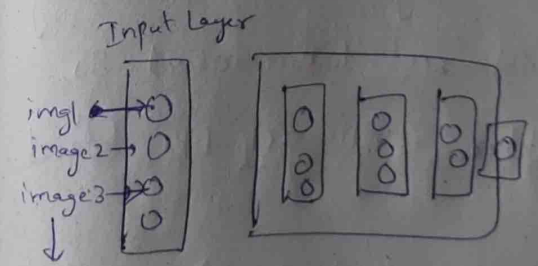
In which we have use many layers and Neurons and for Generalizing (or) to optimize (or) to get more accuracy we have done tuning on top of model by using Dropout, L1 & L2 Regularization.





- As model is Mathematical function which understand only numbers so for most of the data which we get will be in raw numbers [Image, audio ----]

So, we can direct use and give to Neural Network.



every image is converted as pixel and gives to model.

When we compare with older dataset to this CNN here each image acts like each record in ANN.

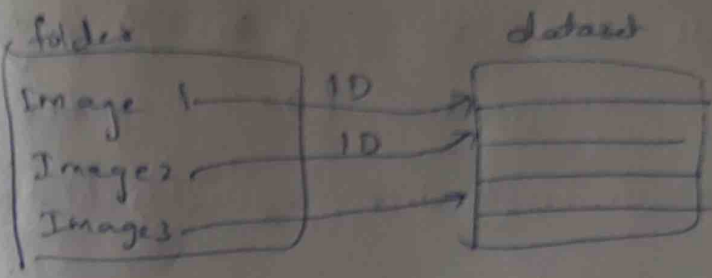
So, we have to use/send each image to the Neural Network.

* image is of 3D/2D we have to convert it into 2D and then 1D so, that it acts like a single record/vector.

So, here we can say that every Data set ~~set~~ has number of rows/records.

when come to image we said every image should act like record so, entire folder will act like dataset.

- Now, lets convert each image to 1D and store in a dataset file (csv) so, that we can use normally



Lets see a Example which was close by open community MNIST.

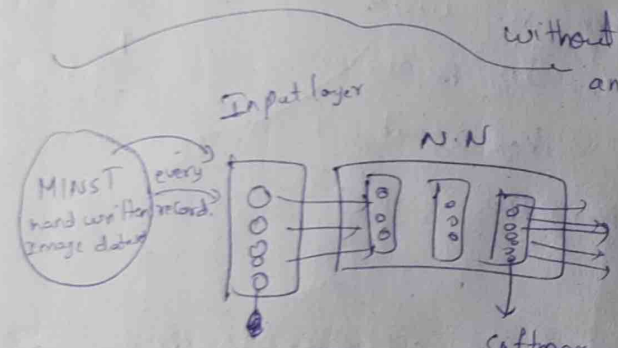
Where when we write a character i.e; example on paint we write 1 then it takes that picture and identifies what character it is.

* For this we take images in as a records and each column/feature will act like a single pixel of image.

- Lets download a dataset / get it from pip command of this MNIST hand written dataset.

Which has dataset of most of the hand written images of Numbers and converted in a dataset file.

- In this dataset we are going to identify/detect (0-9) numbers. So, it acts like multi classification for output. (it output layer we need 10 neurons). (use softmax)

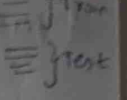


Code :- [This is not CNN]

```
from keras.datasets import mnist
dataset = mnist.load_data('mnist.db')
```

In this dataset we have both training and testing data.
So, we can use splited parts.

train, test ← dataset



↓

In this we have (2D) Images and there output is;

[[[20 X Image], [Value of Y Image]]]

- even in test also we have same format data.

```
x_train, y_train = train   (train data)
```


```
x_test, y_test = test   (test data)
```


```
img = x_train[0].shape
```

↓

Takes first image and shows the shape as (28, 28)

```
import matplotlib.pyplot as plt
plt.imshow(img, cmap="gray")
```

for testing/ the image.
seeing

- Now we have to convert the image from 2D to 1D.
for sending/using in the model.

→ img.reshape(784) [# 28*28 = 784]

here single image ie; x_train[0] is converted in single [1D] array.

So, lets convert all the images.

→ x_train.shape

(60,000, 28, 28)

↑

for not Changing this all records ~~first and~~ and change only the next 2D image. we use -1.

x_train_1D=

→ x_train.reshape(-1, 28*28)

↓

The 2D is converted into 1D

→ x_test_1D = x_test.reshape(-1, 28*28)

[This is for testing data].
in future we use.

→ y_train is a 1D and Categorial variable

So, it is output (or) y so, we need convert into Dummy variables.

- here instead of pandas we also use another way ie; Keras. [anything is fine]

- from keras.utils.np_utils import to_categorical

- y_train_cat = to_categorical(y_train)

↓ dummy variables

This creates like

	0	1	2	3	4	5	6	7	8	9
for 0 value →	1	0	0	0	0	0	0	0	0	0
for 1 value →	0	1	0	0	.	.	.	.	.	.

#
Now, we can create model

- from keras.models import Sequential

- model = Sequential()

- from keras.layers import Dense

- model.add(
    Dense(units=512, input_shape=(784,),
    activation='relu')
)

- model.add(
    Dense(units=256, activation='relu')
)

; 128,
64,

last layer

model.add(
    Dense(units=10, activation='softmax')
)

model.compile(

optimizer = RMSprop() (for optimizer)

↑ for this we need to import

we can use adam also

optimizer='adam',

loss='categorical_crossentropy',

metrics=['accuracy']

↑ for just seeing the accuracy when training

)

model.fit(

X_train_1D,

y_train_cat,

epochs=10)

img_test = X_test[15]

img_test_1D = img_test.reshape(28*28)

↓

taking one test image and converted into 1D.

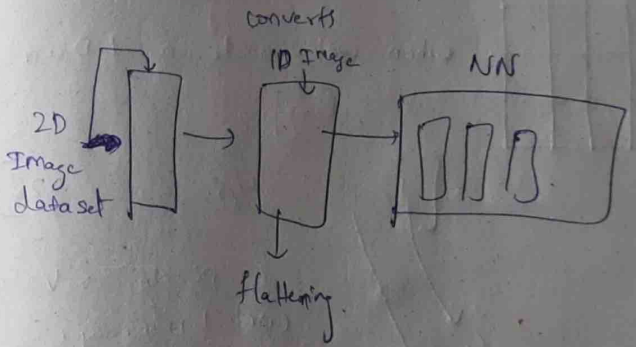
model.predict(img_test_1D)

Showing some errors but it has to work debug it. [Check sir code]

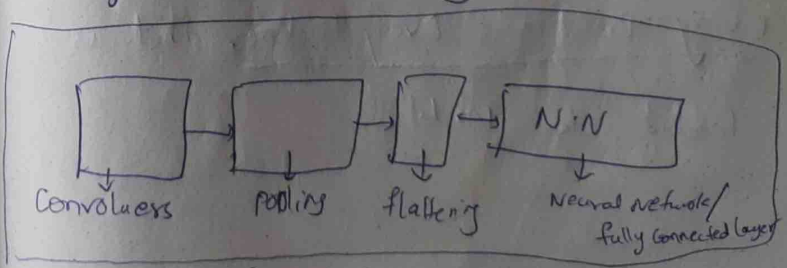
05/01/22

- Before we took Data of Image as it was in 2D and we have converted in 1D.

For this conversion we have a concept called Flattening which will help us for converting 2D image data to 1D image/Array/dataset



we have more two concepts, Convolution, Polling before flattening.



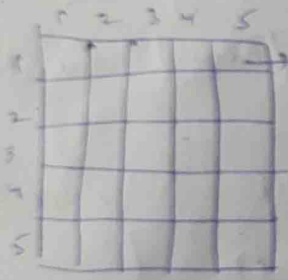
CNN (convolution Neural Network)

Those 3 combine together we call it as convolution neural network (CNN)

These 3 can be used directly instead of separetly. ie using CNN.

CNN ⟶ used for Image classification

Image



5×5 = 25 pixels.

when we create a 1D array we get

[1, 2, 3, 4, 5, - - - - 25]

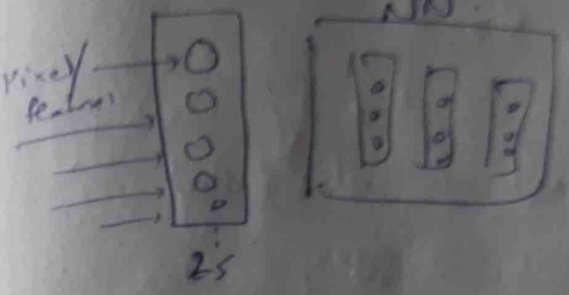
25 pixels as vector (or) 1D array.

x_1 x_2 - - - -

features.

Every pixel = = Every feature, that is feeded to model.

. here we have 25 features which we give to model.



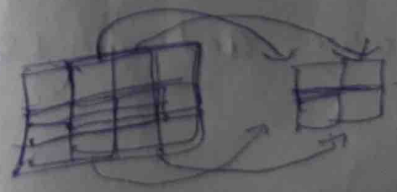
So, if we go to Big image (or) having pixels 1024*1024 then, we have many features that takes much more time to calculate the weight.

For this we are going to reduce the size of image without loosing the data/content by using/taking few pixels [Ex:- 9 [3×3]] into a single pixel.

By using mean/average.

This is called as Pooling / mean pooling.

Eg:



Now image is decreased by pixels

- we can make changes that how much (or) how many pixels have continuity and important data and reduce the poding size.

- Neural Network is also called as fully connected layer.

Convolve :-

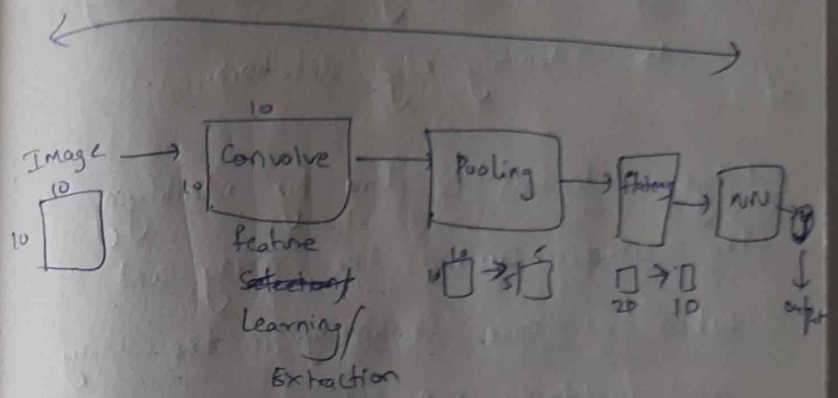
- In Image we have many features [like, face, eyes, hair- - - -]

  So, this feature learning is done by convolve.

- This mainly does is identifying edges [edge detection].

- So, the convolution does both first edge detection then identifies features in that edge detection and identifies what is that object.

CNN (Convolution Neural Network)



07/01/22

- The main functionality of Convolve layers is to Identify edge detection:

- Edge detection is done by identifying the values within the array and when we have Continuity in the array and suddenly we get Change in the value of array then we can identify that there was an object at that position.

single row of the Image example.

Ex: [0, 0, 0, 0, ---- 1, 1, 1, 1, ---- 0, 0]

here the object is

even Convolve does many operations on top of this vector/array and identifies the change in value and detects the objects.

. This uses Kernel for operations.

After this operation we get the edges of the object.

Ex: [0, 0, 0, 0, ---- 1, 0, 0, 0 -- -1, 0, 0, 0]

here the object is.

This value has come for one of the Kernel But in real Image Convolve will use multiple Kernels. and identify multiple feature map/detection.

i.e Ex: Kernel units = 64
then feature maps = 64

If we use Kernel as 3x3 then if we have Image 50 * 50 pixels then after operation we loose two raw pixels & colum pixels and become 48 * 48 pixels.

Code:-

Lets try for one real Image

```
from scipy import misc        → has some images
img = misc.ascent()
```

After importing matplotlib.

```
plt.imshow(img, cmap='gray')
f = numpy.array([
    [0, 0, 0],
    [0, -4, 0],
    [0, 0, 0]
])
```

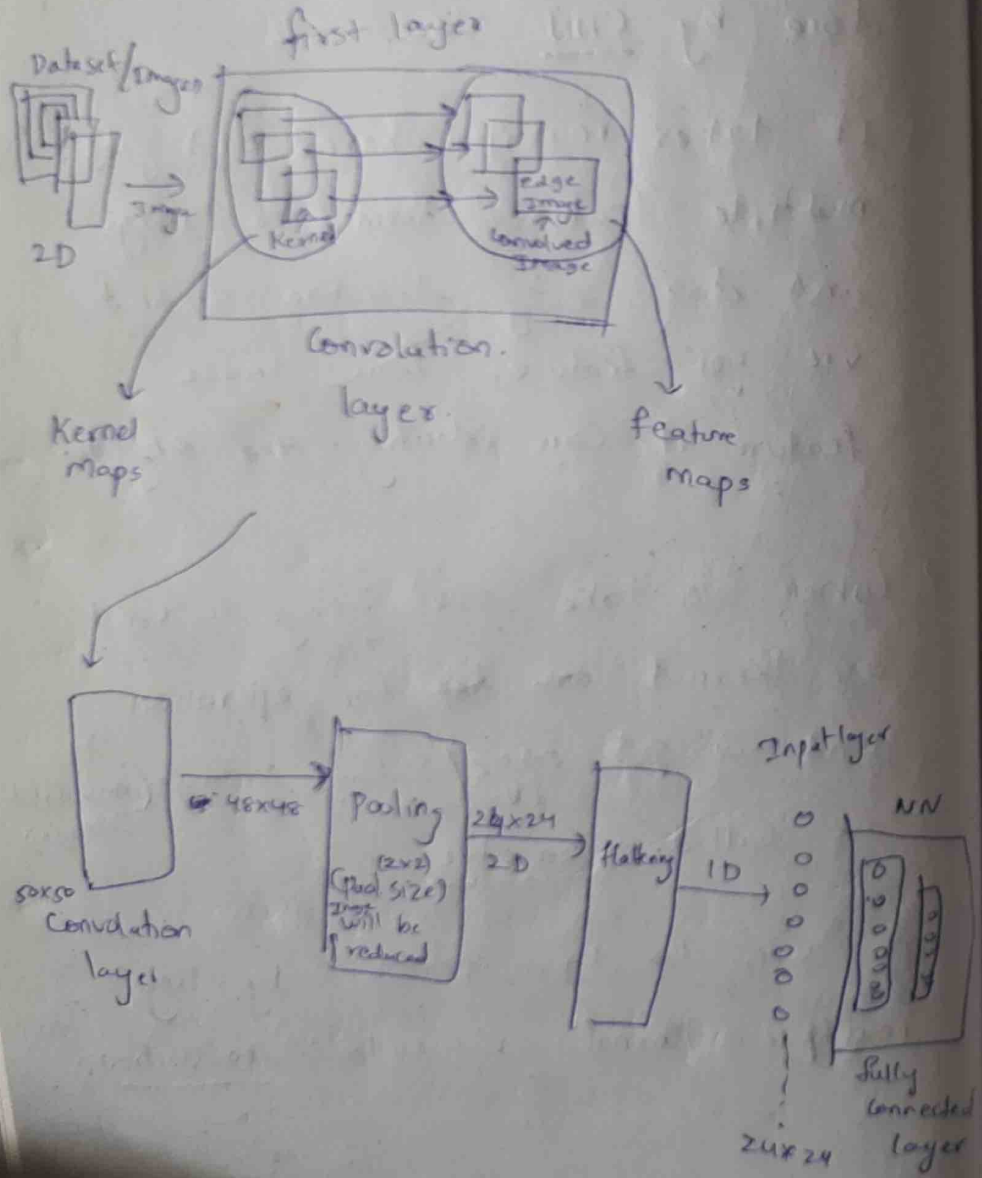
} this is kernel make changes according to for edge detection

```
from scipy.signal import convolve2d
cimg = convolve2d(img, f)
plt.imshow(cimg, cmap='gray')
```

- Now, this change in kernel for doing operations on every pixel of Image is done by us but it is done by CNN also.
- It takes multiple Kernel and gets multiple features from the Image and does edge detection so, that we get features from those feature it can identify the object.

- When we take one image a used one Kernal ~~on~~ ~~top~~ for operation and we get edge/featured Image (convolved Image) is called Convolve.
- If we do this operations by using multiple Kernel is called convolution.

CNN Architecture :-



For output layer we need to classify the objects so, we use Softmax and then we identify the loss by comparing with y and $\hat{y}$ as it is supervised learning and to reduce loss we backpropagate to again from convolution layer and change Kernel and we go on.

12/01/22

→ DataSet = = folder
we have (we have
→ Image = = Records

• Lets take one dataset of images which has dogs and cats.

Lets Create CNN model:

- from keras.models import Sequential
- model = Sequential()

we know, first layer is convolve layer and identifies the feature/edge detection.

- from keras.layers import Convolution2D

and now instead of Dense layer we are using this Convolution2D layer.

- model.add(

Convolution2D(

filters=32, → This will create 32 Kernel units which different values of Kernel and identifies edges

Kernel_size=(3,3), → Kernel size is 3×3

input_shape=(64,64,3),

)

)

we have converted (or) going to convert all our images to 64*64*3 shape so, that all images will be of same size i.e; 64×64 pixel with 3D Image.

Now, the first layer/Input layer is created.

Then Lets Create pooling layer.

This acts like second layer.

```
from keras.layers import MaxPooling2D

model.add(
    MaxPooling2D(pool_size = (2,2))
)
```

This will be used to convert (or) reduce the image by 2*2 pixels.
[Image gets half]
ie; 31*31 pixels

now, lets use can Create flattening layer.

```
from keras.layers import Flatten

model.add(
    Flatten()
)
```

```
model.summary()
```

from flattening layers how many records comes that many records will act like features to Neural network.

Now, lets Create N.N

```
from keras.layers import Dense

model.add(
    Dense(units=128, activation='relu')
)
```

```
{
    64,
    32
```

This all layers are fully connected with convolve, pooling, flattening layers.

so, for last layer it is Binary Classification (Dog/cat) so, we use Sigmoid activation function.

```
model.add(
    Dense(units=1, activation='sigmoid')
)
```

Now, Entire CNN is Created.

17/01/22

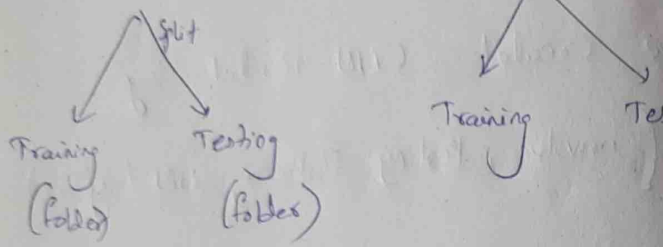
- we have created CNN model by using (convolve, Pooling, flattening, NN layers)

So, for increasing accuracy we can change number of layers, (or) number of neurons within the layers, we can change number of Kernels/filters and even we can change the size of compression of image ie; pooling.

This way of improving accuracy is called tunning.

- Now, Lets fit/train the data into the model.

folder (dataset) == • CSV (dataset)



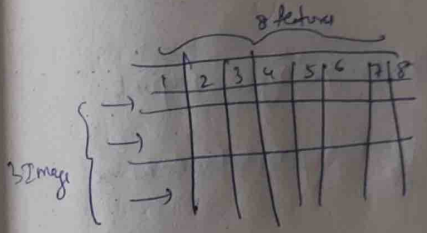
we have output as 'y' as in binary classification we have 2 types of values classified in y output

Here we have two folders i.e; Cats and dogs folder where we have data set / Images.

These folders acts like the output.

Normally we have data set and we have records and features in it.

- When we have lot of features and less number of records / Images then identifying weights will be very difficult and will not be accurate



For this we can do one thing that is we have 4000 Images and we have many features maybe around 32K So, we need to find 32K weights.

- So, we can collect extra Images (or) using 4K Images we can generate new Images (ex: flip (or) crop Image --)

This is called data / Image generation.

- So, for this process of generating new Images is called pre-processing.

- Keras has pre-built Image preprocessing.

```
from keras_preprocessing.image import
                    ImageDataGenerator
```

// This is Sample see sir code/online

```
train_datagen = ImageDataGenerator(
        featurewise_center = True
        featurewise_std_normalization=True
        rotation_range = 20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip = True
        )
*/
```

- Change the directory of training folder ie; train_generator parameter & validation_generator for testing folder

- Even our image we are getting is (64,64) so, we need to change target_size parameter to (64,64) as CNN is taking 64 Neurons.

- Now, lets train the model.

it uses train_generator & validation_generator which is very useful and saves RAM space and on the fly it generates multiple Images.

```
model.fit( train_generator,
        steps_per_epoch=8000,
        epochs = 50
        validation_data = validation_generator,
        validation_steps=800)
```

** before this try (compiling)

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

copy paste the model.fit() Code here

(try epochs = 2 in fit() as it takes lot of time

Save the model now,

→ model.save("myimage.hs")

Lets predict it by converting our test data in 4D as our Image/trained data converted into 4D.

('SSD - single shoot detection box model search in gagle for researching)

26/01/22 Advance CNN

- As we know convolve layers identifies the features.
- And when we create multiple convolve layers connected to each other then we can get much accurate feature detection.
- So, there are many models which uses many convolve (or) pooling layers and makes the model more accurate based on Images.
- Even they tried many CNN Networks to make (or) increase accuracy.
- Ex:- LeNet (or) AlexNet are some examples.

- For downloading Image dataset (many datasets examples)

  in google :- Image net

  (or)

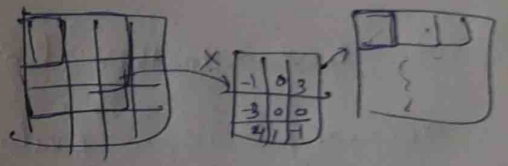
  image-net.org/download.php

- So, many communities / researchers used many combinations of layers (convolve, pulling, ANN..) and did published. Where research papers.

- Sir showed level model/network and used MNIST dataset and created a model.

- In Convolve layer we used Kernel and made operations and get after opertation we get one pixel.

That output some times can becams negative as Kernel might have negative values.

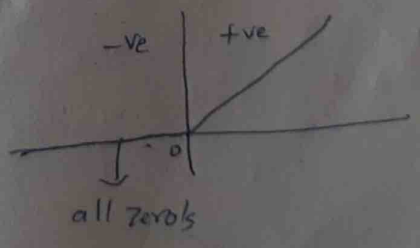


So, for this we have a function which removes negative values/pixels and replace with Zero.

(as we never have negative pixel color)

That function is called Relu.

This is also called as Relu activation function (rectified linear unit)



So, we do Convolve, Relu, Pooling

These three are called (CRP)

- Another name for pooling is subsampling.

- we have another function for seeing accuracy ie; evaluate()

- Verbose is a parameter at fitting the data which shows output to us.
  (verbose = 0 will not show output while traing)

28/01/22

Mostly we use

opencv (library) → cv2 (module) → some functions to perform operations on Image.

- Instead of using this opencv ~~module~~ library from now, we are going to use

Pillow (library) → PIL (module)

Behind the Keras also we used in building CNN model it uses Pillow only.

/> pip install pillow

- Small Code

- from PIL import Image
  (PIL: module inside pillow library)

- image = Image.open('cat_or_dog.jpg')
  → used to load & open that Image. (even we can store)

- type(image)
  → show in which data type it is.

- image.show()
  → It will open in new window.

- image.crop((100, 100, 300, 300))
  → This will crop the Image

- image.rotate(30)
  → rotates the image 30 degrees

- image.rotate(45, expand=True)
  → This will not cut the Image and just rotate complete Image

- image.resize((200, 200))
  → resizes the Image

- Image.ROTATE_90
  → There are few constant variables used to transpose/flip the Image

- image.transpose(Image.ROTATE_90).resize((100, 100))
  → Image gets flip/Rotate and gets resized.

- image.size
  - The size (or) shape of image
- image.height
  - height of image
- image.width
  - width of image
- image.format
  - is it jpg (or) jpeg
- image.transpose(Image.ROTATE_90).save('new.png')
  - Saving as new Image

So, we can use Keras directly,

- from Keras.Preprocessing import image
- Kim = image.load_img('Cat_or_dog_1.jpg')
  - loads image
- type(Kim)
  - This tells internally Keras also uses Pillow

- we mainly use this for preprocessing of Image

- Visual Geometry Group has Created a model (VGG model) which has used ImageNet (Saw yesterday) dataset and they have Created around 92% accuracy.
- There are 2 types
  - → VGG-16 (ie; 16 layers) of diff. CRP(Convolve, Relu, pooling)
    - Mainly used Convolve as layer

- we can also Create many model by Changing different parameters (Convolve layer, Multiple CNN...)
- And for automatic change of this parameters then come MLOPS which automatically substitute different number of parameters and gets best model.

- Even Aws Sagemaker which will Automatic & Build a model.

- So, if you want to you/create a model (or) CNN model which identifies basic dog(or) cat then we can use the VGG model. which is already trained & saved.

- Even we have VGG 19 network

- So, we can see the Architecture and create our own model but it takes a lot of time.

Code :-

```
from keras.applications import vgg16

model =
vgg16.VGG16(weights='imagenet')
```
↳ providing weights of that imagenet data set/ model

```
model.summary()
```

```
model.layers
```
↳ we can see what all layers are used.

```
model.layers[0].input
```
↳ what is the first layer input which will be 4-dimension.

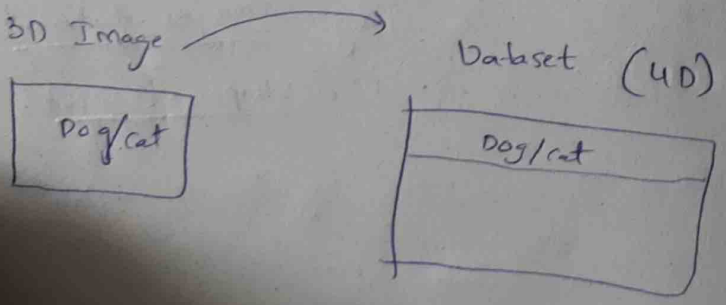
(rows, Image height, Image column, RGB color)

- rows → no. of Images.
- Image height, Image column, RGB color → This is total Images dataset.

i.e; shape of input

- from Keras.Preprocessing import image
- img = image.load_img('Cat.jpg', target_size = (224, 224))
  - we have change Image Size as much as the size of Image given to the VGG model (ie; got trained by that size)
- img.size
- Now, image is in 3D but there model is in 4D, so, for predicting we need to convert our 3D to 4D image

ie; 3D ⟶ 4D



- type (img)
- img_np = image.img_to_array(img)
- type (img_np)
- img_np.shape
- import numpy as np
- final_img = np.expand_dims(img_np, axis=0)
- final_img.shape
  - ↳ now it gets 4D.

now, we can give to model for prediction.

- model.predict(final_img)
  - ↳ This gives all the categories and their percentages by change it has 1000 categories at last layer so, identifying is difficult

- from keras.applications.vgg16 import
  decode_predictions
- from keras.applications.vgg16 import
  preprocess_input
- fimg = preprocess_input(final_img)
  - It converts into right format.
- pred = model.predict(fimg)
- decode_predictions(pred, top=3)
  - top 3 predictions it shows

31/01/22

- Now, we have used vgg16 Network for prediction.

we even have

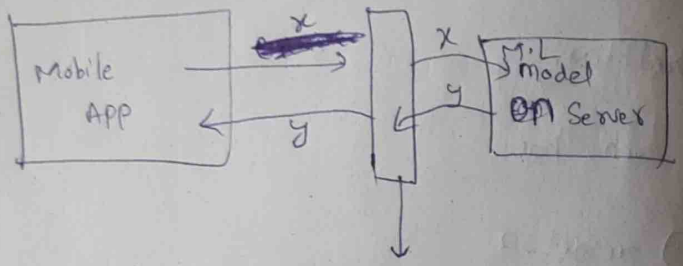
1. resnet50
   - In this we have 50 layers.
2. inception v3

and we have many more.

So, even we can see those Network layers and can create our model.

- Change respective parameters / layers / CRP (or) NN and make the model accurate.

- Integrating mobile app with M.L model



Web application

↳ Rest API

↳ python-cgi (flask)

(or)

Node

- Lets use Google Colab now, as it runs/uses on GCP.

- And even we can use Gpu

↓

Graphical processing unit

which in cpu it runs one instruction at a time but in Gpu we can run multiple/parallel instructions at once.

- There a option called Connect and use Connect to hosted runtime.

- Runtime is option on top panel and select change runtime type and select hardware acclerator as Gpu, and click save.

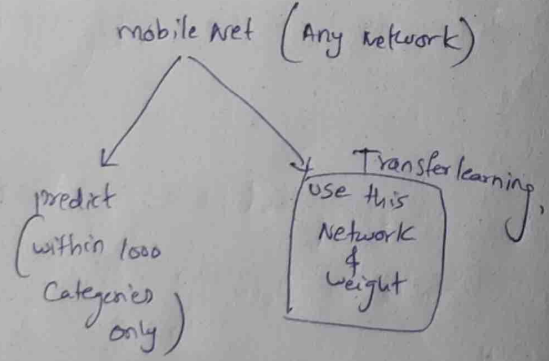
- Tpu is Tenser processing unit which is like combination of Gpu's (more optimized) (It's almost same as Gpu only)

02/02/22

Transfer Learning & fine Tuning

↳ mostly it is used in CNN (we can use in even RNN & ANN)

- Transfer learning is done by using existing model, (or) Network and its weights and Creating new model which can predict our new categories also.



we can say, we are Transfering the existing network on model's learning and Creating new model.

Code :-

```
from keras.applications import mobilenet
model = mobilenet.MobileNet(weights = 'imagenet')
```

This is a dataset and has 1000 Categories

So, we get last layer as 1000 Categories

```
model.layers
```

↓

Shows all layers.

- So, here we known last layer has 1000 Categories and which is what going to predict.

So, Lets remove last layer and add our own layer and with our own categories.

But remaining layers will be as it is.

- here we can directly fit and it doesn't touch already trained layers and it trains last layer only.

model.layers[0].trainable

↓

first layer is trained (or) not

If shows True i.e; that layer is already trained so, don't train me again. It says.

- But if we use fit() then all layers goes back to normal.

So, we are going to make every layer as false then it doesn't train again that layer.

```
for i in model.layers:
    i.trainable = False
```

↘ all layers got locked.

So, that no one can train model again

- The above layers known how to extract features from the image.

```
from keras.models import Sequential
from keras.layers import Dense
```

↓ all layers are in Sequential way so, we use this

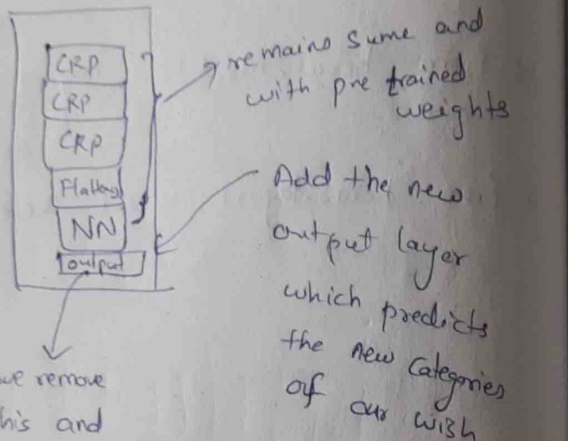
- The output layer is also called as Top layer.

- So, for getting last layer.

```
model.output
top_model = model.output
```

- Even we can add full connected layer also by removing the output layer. (our own model (previous model we created))

07/02/22



- Now, we can use mobile net.

```
from keras.applications import mobilenet
model = mobilenet.MobileNet(weights="imagenet")
model.summary
```

- There things we did yesterday. and now, lets substitute the output layer.

for removing the output layer directly while loading the network we can do.

```
model = mobilenet.MobileNet(weights='imagenet',
        input_shape=(224, 224, 3), include_top=False)
```

specifying this because the layer are trained by this shape

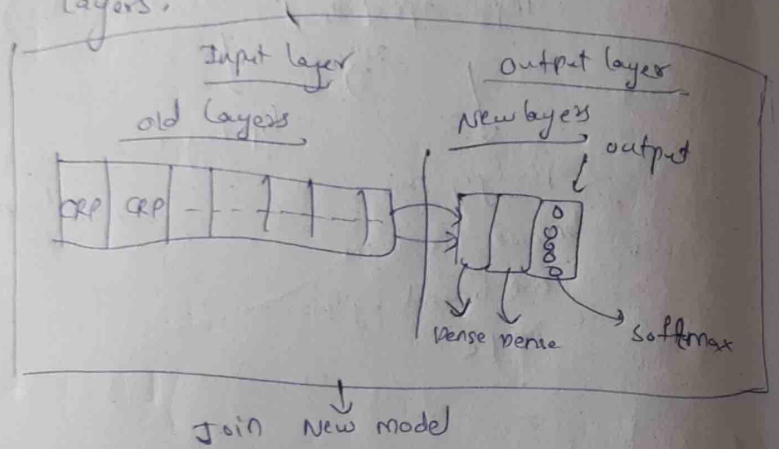
top layer (or) output layer

- And we made already trained layers as False i.e; we don't need to train them again.

- use yesterday code now.

And now, lets create new layers and add it.

- Instead of creating a single output layer as the existing model created / trained on already existing images so, its better to create multiple layers.



- So, from yesterday we known that top_model has ~~target~~ last layer of existing layers.

and now, lets add new layer to that last layer

```
top_model =
    Dense (units = 1024, activation = 'relu') (top_model)
```

adds to the last layer

and overrides this last layer.

- Now, the layer i.e. last layer is our own layer.

```
top_model ~~output~~
```

```
top_model = Dense (units = 512, activation = 'relu') (top_model)
```

```
top_model = Dense (units = 3, activation = 'softmax') (top_model)
```

This is last layer now,

- Now, we need to combine them we just added to each other / Connected. by we need to create new model so, lets combine.

```
from keras.models import Model
```

normally,

- model.input
  ↓
  This knows where the model's first layer is.

- top_model
  ↓
  This knows the last layer now

So,

new_model =

- Model(inputs = model.input, outputs = top_model)

- new_model.summary()

now, lets train newly added layers.

- now, can fit the model directly

Sir, showed flowers and monkeys example models.
(check in Sir's google drive)