

Human brain is build of 1000's of neurons.

These neurons have nerve cells.

In deep learning we r trying to create a
brain for machine so that it mimics human behaviour.

So from the idea of neurons , nn is created.

There are many types of nn,

perceptron, ANN,CNN,Transformers,RNN

ANN- Artificial NN- this is used for tabular data.

CNN- use on image/video data

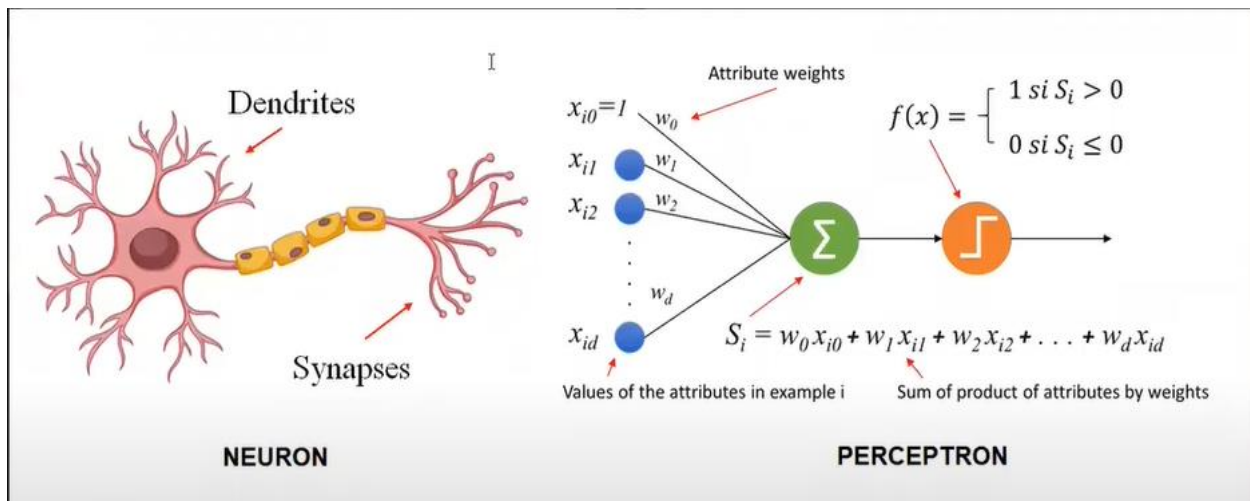
RNN- for text/timeseries data/speech

Now, perceptron- it is the building block
of all the neurons.

So, 1st understanding neuron.

it has O-0

O-dendrites,-axon,0 synapses

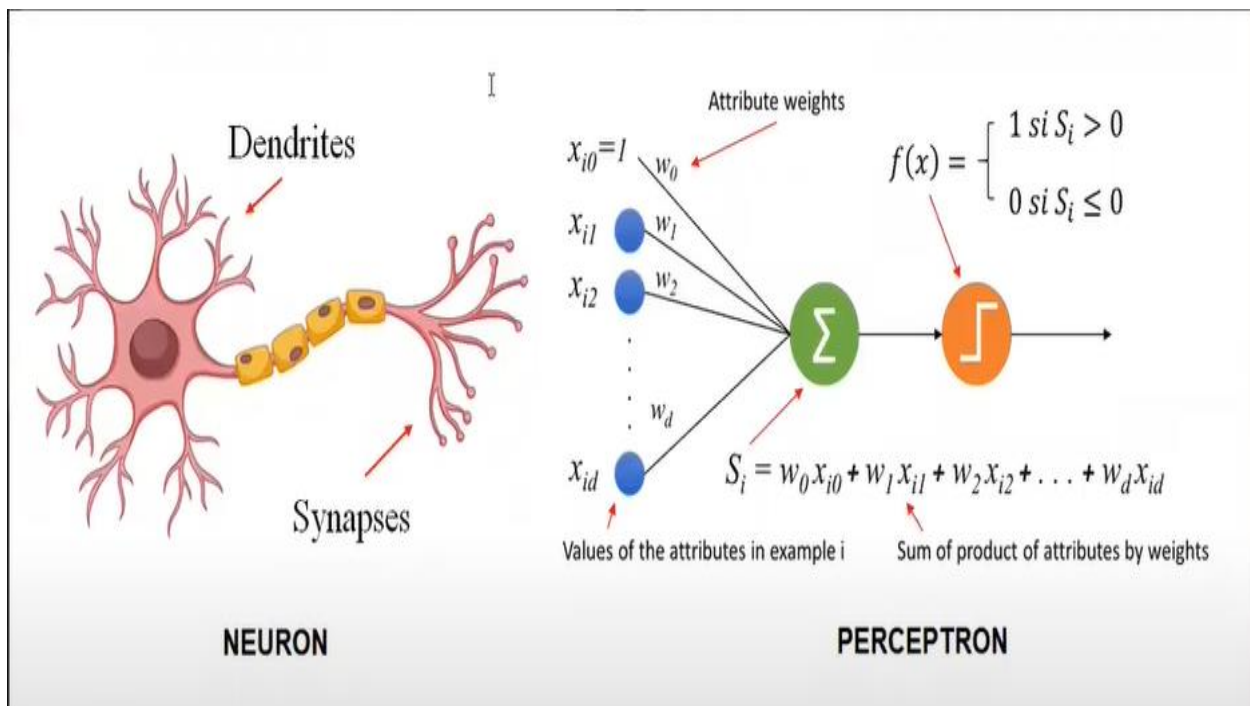


now this is a single neuron of of brain.

Our brain has network of these neurons.

Perceptron

A Perceptron is an algorithm for supervised learning of binary classifiers, it can be used for regression as well. This algorithm enables neurons to learn and processes elements in the training set one at a time.



Understanding the working of Perceptron,

For eg, in Linear Regression we have, $y = m_1x_1 + m_2x_2 + m_3x_3 + b$

Now, number of x is equal to number of attributes .



Each input has some weight associated with it which are multiplied with the input.

These inputs are given to a node , which has 2 operations inside.

1 is to sum those inputs and then apply activation function on this summation.

And the output is given to the neurons further.

Now in perceptron , these activation functions can be threshold some threshold.

But these activation functions can be of variety depending on the work in hand.

Activation_functions:

What is an Activation Function?

Activation functions introduce **non-linearity** in a neural network.

They help the network learn complex patterns in data.

Without activation functions, a neural network would just be a **linear regression model!**

Mathematical Representation:

For a neuron with input **xxx** and weight **www**:

$$z = w \cdot x + b$$

The activation function **$f(z)$** transforms z before passing it to the next layer.

1. Linear Activation (Identity Function)

Equation: $f(x)=x$

Problem: Doesn't introduce non-linearity → Can't learn complex patterns.

2. Non-Linear Activation Functions

These are widely used because they introduce non-linearity.

1. Sigmoid (Logistic Function)

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

Properties:

Smooth, differentiable.

Outputs in (0,1) → Useful for probability-based tasks (e.g., classification).

Problems:

- Vanishing Gradient (small derivatives slow learning).
- Not zero-centered (gradient updates are unbalanced).

Use Case: Binary classification : (e.g., Spam Detection).

2. Tanh (Hyperbolic Tangent)

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Properties:

Range: (-1,1) → Zero-centered! (helps balance positive & negative values).

Stronger gradients than sigmoid.

Still suffers from vanishing gradient for large values of x. **Use Case:** Recurrent Neural Networks (RNNs).

3. ReLU (Rectified Linear Unit)

Equation:

$$f(x) = \max(0, x)$$

Properties:

Fast & efficient (simple operations).

Reduces vanishing gradient issues.

Problem: Dying ReLU → If neurons output 0, they never recover (gradient becomes 0).

Use Case: Convolutional Neural Networks (CNNs) for image recognition.

4. Leaky ReLU

Equation:

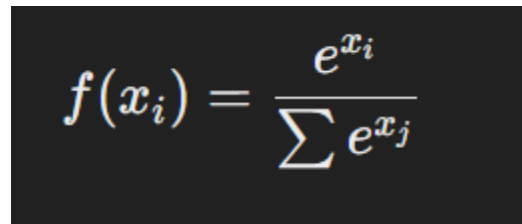
$$f(x) = \max(0.01x, x)$$

Avoids dying neurons by allowing small negative values.

Use Case: Deep networks where ReLU causes neuron death.

5. Softmax – For Multi-Class Classification


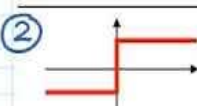
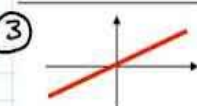

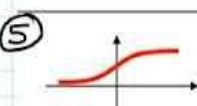
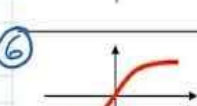
Formula:








$$f(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$$

Use When: Multi-class classification output

Avoid When: Binary classification (use Sigmoid instead).

Commonly Used Activation Functions

1. Step function: $f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$	① 	Range $\{0, 1\}$
2. Signum function: $f(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	② 	$\{-1, 1\}$
3. Linear function: $f(z) = x$	③ 	$(-\infty, \infty)$
4. ReLU function: $f(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$	④ 	$(0, \infty)$
5. Sigmoid function: $f(z) = \frac{e^x}{1+e^x}$	⑤ 	$(0, 1)$
6. Hyperbolic tan: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	⑥ 	$(-1, 1)$

Activation Function	Type of Problem	Best Used In	Why Use It?	When to Avoid?
ReLU (Rectified Linear Unit) 	General-purpose	Hidden layers in deep networks (CNNs, NLP, Transformers)	Fast, prevents vanishing gradients	Can cause dying neurons (output stuck at zero)
Leaky ReLU 	General-purpose	Deeper networks when ReLU has dying neurons	Fixes ReLU dying problem	Slightly more computation than ReLU
Sigmoid 	Binary classification	Output layer (e.g., spam detection)	Converts outputs into probabilities (0-1)	Causes vanishing gradients , slow learning
Softmax 	Multi-class classification	Output layer for multi-class problems	Converts scores into probabilities summing to 1	Not useful for binary classification
Tanh (Hyperbolic Tangent) 	Regression (bounded)	When input values range from negative to positive	Centered around zero (-1 to 1)	Suffers from vanishing gradients
Linear (No Activation) 	Regression	Output layer when predicting real values	Keeps output as continuous values	Not for classification

So,

ReLU is the **default** for hidden layers.

Sigmoid & Softmax are used for **classification outputs**.

Tanh is used when **inputs contain negative values**.

Linear Activation is used for **regression outputs**.

The **Perceptron** is one of the simplest **artificial neural network** architectures. Perceptron consists of a single layer of input nodes that are fully connected to a layer of output nodes. It is particularly good at learning **linearly separable patterns**. It utilizes a variation of artificial neurons called **Threshold Logic Units (TLU)**, which were first introduced by McCulloch and Walter Pitts in the 1940s.

Types of Perceptron:

Perceptrons are the foundation of neural networks and come in different types based on their structure and capabilities.

1. Single-Layer Perceptron (SLP)

- Has **one input layer and one output layer** (no hidden layers).
 - Uses **linear activation functions** (e.g., step function).
 - Can only solve **linearly separable problems** (e.g., AND, OR logic gates).
 - **Fails with XOR problem** because it needs non-linearity.
-

Used For: Simple binary classification tasks.

Limitations: Cannot learn complex patterns.

2. Multi-Layer Perceptron (MLP)

- Has **one or more hidden layers** between input and output.
 - Uses **non-linear activation functions** (ReLU, Sigmoid, Tanh).
 - Can solve **linearly inseparable problems** (e.g., XOR).
 - Uses **backpropagation and gradient descent** for training.
-

Used For:

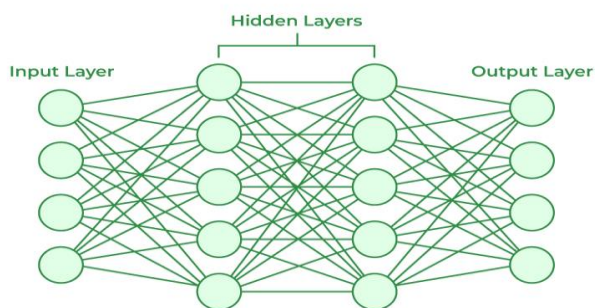
Complex pattern recognition, Image classification, NLP (Natural Language Processing)

Limitations: Computationally intensive, requires large datasets.

Artificial Neural Network :

Artificial Neural Networks contain artificial neurons which are called **units** . These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.



◆ How is MLP Different from ANN?

MLP is a **type of Artificial Neural Network (ANN)** but with certain specific features.

Feature	MLP (Multi-Layer Perceptron)	General ANN (Artificial Neural Network)
Structure	Fully connected layers	Can include CNNs, RNNs, Transformers, etc.
Hidden Layers	Must have at least 1 hidden layer	Can have any number of hidden layers

Feature	MLP (Multi-Layer Perceptron)	General ANN (Artificial Neural Network)
Data Type	Works well with structured data (tables, numbers)	Works with images (CNNs), sequences (RNNs), etc.
Activation Functions	Uses ReLU, Sigmoid, Tanh	Varies (CNN uses ReLU, RNN uses Tanh)
Use Cases	Simple classification & regression	Broader AI applications

Prerequisites for implementing ANN—

1. Pytorch :

PyTorch is an open-source machine learning framework primarily used for training deep neural networks. Developed by Meta in 2016, it has become one of the most popular frameworks in both research and industry². PyTorch provides a way to build neural networks simply and train them efficiently, which has led to its widespread adoption.

Tensors

At the core of PyTorch is the **tensor**, a multi-dimensional array similar to NumPy arrays but with additional capabilities for GPU acceleration¹. Tensors are used to encode the inputs, outputs, and parameters of models. PyTorch supports various tensor operations, including creation, manipulation, and mathematical computations. Uses `torch.nn.Module` for building models.

2. Tensorflow :

TensorFlow is an open-source machine learning library developed by Google. It is designed to facilitate the creation and training of deep learning models by providing a flexible architecture that allows computation to be deployed on various hardware platforms, including CPUs, GPUs, and TPUs.

3. Keras:

Keras is High-level API built on top of TensorFlow. It is designed to enable fast experimentation with deep neural networks and can run on top of TensorFlow, Theano, or CNTK.

Key Functions in Keras

1. Model Building

Keras provides two ways to create models:

Sequential API – Simple, layer-by-layer models

Functional API – Complex architectures like multi-input or custom connections

Example: Building a Neural Network Using Sequential API

```
from tensorflow import keras

from tensorflow.keras.layers import Dense

# Define a simple ANN model
model = keras.Sequential([
    Dense(16, activation='relu', input_shape=(10,)), # Hidden layer
    Dense(1, activation='sigmoid') # Output layer for binary classification
])
```

2. Model Compilation

Before training, a model must be compiled with:

- **Loss Function** (e.g., binary_crossentropy for classification)
- **Optimizer** (e.g., adam, sgd)
- **Metrics** (e.g., accuracy, mse)

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

3. Model Training

To train a model, use `.fit()`, specifying data, epochs, and batch size.

```
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
```

4. Model Evaluation

To check model performance:

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print(f"Test Accuracy: {accuracy:.2f}")
```

5. Making Predictions

Use `.predict()` to get predictions on new data.

```
y_pred = model.predict(X_new)
```

Show ANN working on Churn_prediction.ipynb using Churn Modelling data.

How to Calculate the Number of Parameters in a Neural Network?

Each **Dense (fully connected) layer** in Keras contains **weights and biases**. The number of parameters in a layer is calculated as:

$$\text{Total Parameters} = (\text{Number of Inputs} \times \text{Number of Neurons}) + \text{Number of Neurons}$$

$$\text{Total Parameters} = (\text{Number of Inputs} \times \text{Number of Neurons}) + \text{Number of Neurons}$$

where:

- **Weights** = Number of inputs × Number of neurons
 - **Biases** = 1 per neuron
-

✓ Parameter Calculation for Each Layer

Given the model:

```
model = Sequential()  
model.add(Dense(11, activation='sigmoid', input_dim=11)) # Layer 1  
model.add(Dense(11, activation='sigmoid')) # Layer 2  
model.add(Dense(1, activation='sigmoid')) # Output Layer
```

1 First Layer (Input Layer → Hidden Layer 1)

- **Input Dim = 11, Neurons = 11**
 - **Weights:** $11 \times 11 = 121$
 - **Biases:** 11
 - **Total Parameters:** $121 + 11 = 132$
-

2 Second Layer (Hidden Layer 1 → Hidden Layer 2)

- **Input = 11 neurons, Neurons = 11**
 - **Weights:** $11 \times 11 = 121$
 - **Biases:** 11
 - **Total Parameters:** $121 + 11 = 132$
-

3 Third Layer (Hidden Layer 2 → Output Layer)

- **Input = 11 neurons, Neurons = 1**
- **Weights: 11×1=11**
- **Biases: 1**
- **Total Parameters: 11 + 1 = 12**

Final Total Parameters

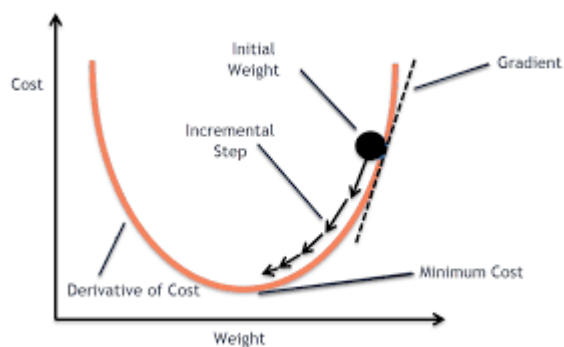
Layer	Input Neurons	Output Neurons	Weights	Biases	Total Parameters
Dense 1	11	11	121	11	132
Dense 2	11	11	121	11	132
Dense 3	11	1	11	1	12
Total	-	-	253	23	276

Final Answer: 276 Parameters

Next is during Compilation we have used optimizer as Adam, loss as binary cross entropy.

Optimizers control **how neural networks update weights** to minimize the loss function. Choosing the right optimizer **improves training speed and accuracy**.

i.e.



The general **weight update rule** is:

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

where:

- W = Weights
- η = Learning rate
- $\partial L / \partial W$ = Gradient of the loss function
-

Popular Optimizers :-

1. SGD (Stochastic Gradient Descent): Updates weights using gradients of a **single** data point at a time.

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

Use Case: Small datasets, convex optimization.

2. Momentum Optimizer

Definition: Uses a velocity or momentum term to accelerate gradient descent.

Formula:

$$v_t = \beta v_{t-1} + (1 - \beta) \frac{\partial L}{\partial W}, \quad W = W - \eta v_t$$

Use Case: Deep networks, faster convergence.

3. Nesterov Accelerated Gradient (NAG)

Definition: Prevents overshooting by applying updates to look-ahead weights.

Formula:

$$v_t = \beta v_{t-1} + \eta \frac{\partial L}{\partial (W - \beta v_{t-1})}, \quad W = W - v_t$$

Use Case: Prevents overshooting, stabilizes updates.

4. Adagrad (Adaptive Gradient Algorithm)

Definition: Adjusts learning rate for each parameter based on past gradients.

Formula:

$$W = W - \frac{\eta}{\sqrt{G_t + \epsilon}} \frac{\partial L}{\partial W}, \quad G_t = \sum \left(\frac{\partial L}{\partial W} \right)^2$$

Use Case: Sparse data, NLP, embeddings.

5. RMSprop (Root Mean Square Propagation)

Definition: Uses a moving average of squared gradients to normalize updates.

Formula:

$$v_t = \beta v_{t-1} + (1 - \beta) \left(\frac{\partial L}{\partial W} \right)^2, \quad W = W - \frac{\eta}{\sqrt{v_t + \epsilon}} \frac{\partial L}{\partial W}$$

Use Case: RNNs, time-series tasks.

6. Adam (Adaptive Moment Estimation)

Definition: Combines Momentum and RMSprop for adaptive learning rates.

Formula:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial W}, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial W} \right)^2$$

$$W = W - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

Use Case: Default optimizer for most deep learning models.

7. AdamW (Adam with Weight Decay)

Definition: Improves **Adam** by adding **weight decay** for better generalization.

Formula:

$$W = W - \eta \left(\frac{\partial L}{\partial W} + \lambda W \right)$$

Use Case: Large-scale deep learning (Transformers, CNNs).

Loss:

For binary classification = `binary_crossentropy`

For multiclass classification = `sparse_categorical_crossentropy`

For regression = `mse`

Bias Variance Tradeoff – On plotting error on y-axis and complexity on x-axis, high bias low variance i.e. high error on training data results in underfitting, whereas low bias and high variance i.e. high error on testing data results in overfitting.

Currently we're working with dense layers as we're working on ANN. As we go ahead, we will learn other different layers.

Number of parameters we have calculated.

So, while creating a model we don't consider input layer differently.

So, nn of 4 layers has 3 hidden layers and 1 output layer.

Metrics give accuracy of model.

Next is Fit—in fit we have batch size, so if I have 500 total rows and I want 50 rows in batch,

Then the total number of batches we will have are 10.

In our case , there are total 8000 rows in training data,

We have batch size of 50, so total number of batches we will have are $8000/50$

=160. So, Total 160 batches we will create.

Next is validation split. In training we have 8000 rows,

So we have kept 20% rows for validation. So around 1600

Rows will go for validation and rest 6400 for training dataset.

i.e. on 6400 rows it is training the data and on rest at the sametime it is testing what it has learned.

Next is epoch. For epoch=10, So now since we have kept 20% of data i.e. 1600

Rows for validation , we are left with 6400 rows, with a batch size of 50,

So $6400/50=128$ batches are created.

Now in 1 epoch the training is done on these complete 128 batches.

With each epoch we can see accuracy of both training and testing data is increasing.

So these are all hyperparameters.

Model.predict—is for prediction.

So over here u can do experiment with,

No. of nodes, Activation function, no. of hidden layers, epochs

Activation on the output layer will be sigmoid only. Don't change that.

The hidden layers extracts features from data.

Here u can discuss bias variance tradeoff.

Next , is deep learning on regression problems.

Data—Admission_Predict.xlsx, graduate_Admission.ipynb

Target here is chance of Admit column.

So putting Dense hidden layer means output from all the nodes are given as the input too all the other nodes.

Activation here is linear, number of features are 7 so $\text{input_dim}=7$.

So number of input nodes=7, so $7*7+7=56$

Next for output layer= $7*1+1=8$

So total parameters = $56+8=64$ parameters.

Now, finding perfect weights and bias values.

So, we need to understand few concept.

1. Epoch, 2. Forward Propagation, 3. Backward Propagation, 4, Gradient Descent.

Forward propagation is moving from input to output.

Now lets suppose in 1 forward pass, model predicted o/p 1.2

But actual o/p is 3.2.

So there are errors. The one with the least error gives highest accuracy.

At the start, some random values are given as weights and biases.

And the model does prediction based on these random weights.

Then the loss is calculated based on MSE.

Now, to minimize error we need to update these weights.

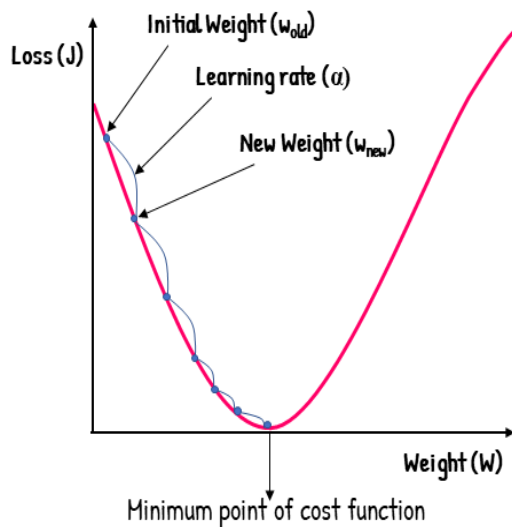
i.e. from o/p now we will go towards i/p, as i/p values can't be changed

but weight and bias values can be changed.

And this 1 forward and backward round is called as an epoch.

Now this coefficients updation is done by Gradient Dscent.

Gradient Descent



$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{\delta J}{\delta w}$$

We want to reach minimum point of cost function which gives the minimum loss.

So the aim of gradient descent is to find the values of weights and biases which gives least error.

$$w_{\text{new}} = w_{\text{old}} - \lambda \left(\frac{d\text{loss}}{dw} \right)$$

Where, $\left(\frac{d\text{loss}}{dw} \right)$ is the partial derivative of loss w.r.t. weight and bias.

$$\text{Now, } \hat{y} = z = w_0 + w_1 x_1$$

Now mse denoted by $J = \frac{\text{summation of } (y - \hat{y})^2}{n}$

$$\text{i.e. } = \frac{\text{summation of } (y - (w_0 + w_1 x_1))^2}{n}$$

$$= \frac{\text{summation of } (y - w_0 - w_1 x_1)^2}{n}$$

Now few rules for applying partial derivation.

$$x^2/dx = 2x, -x/dx = -1$$

So,

$$dJ/dw_0 = 2 \cdot (y - w_0 - w_1 x_1) \cdot (-1) \text{ since it is } -w_0$$

$$\text{Similarly, } dJ/dw_1.$$

So when we differentiate something, we are calculating the tangent on this

U curve w.r.t w_0 . Now if this tangent makes acute angle with the x-axis,

Then the differentiation is +ve, and when tangent makes obtuse angle i.e.

More than 90 degree then its differentiation is -ve.

So dJ/dw_0 will give -ve/+ve value based on the angle it is making with the x-axis.

So, if we assume $w_{old}=2, \lambda=0.1, dJ/dw_0=-5$

Then $w_{new}=2-0.1(-5) = 2.05$

So, we can see loss getting down and weight value moving towards minimum cost function.

Now, when tangent becomes parallel to x axis, its value becomes 0.

Then $w_{new}=w_{old}$

Now, earlier we assumed weight value to be 2

Now assuming weight value to be 10. The perfect weight value is 5.

So 10 is on the right of the weight value.

So, if we assume $w_{old}=5, \lambda=0.1, dJ/dw_0=5$ since the suppose with x-axis is acute angle.

So. $w_{new}=5-0.5=4.5$

And like this ultimately we r trying to get to the lowest cost by adjusting weight values.

Now how gd knows it has to stop updating weights?

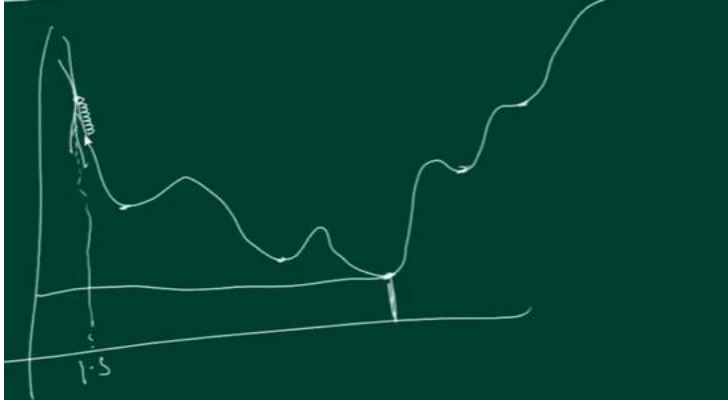
When new weight value is very close to old weight then stop.

Now, if you noticed , the partial derivative is multiplied by learning rate.

So if lr value is very low like 0.01 or 0.001 then it will take too much time to reach correct value of weight.

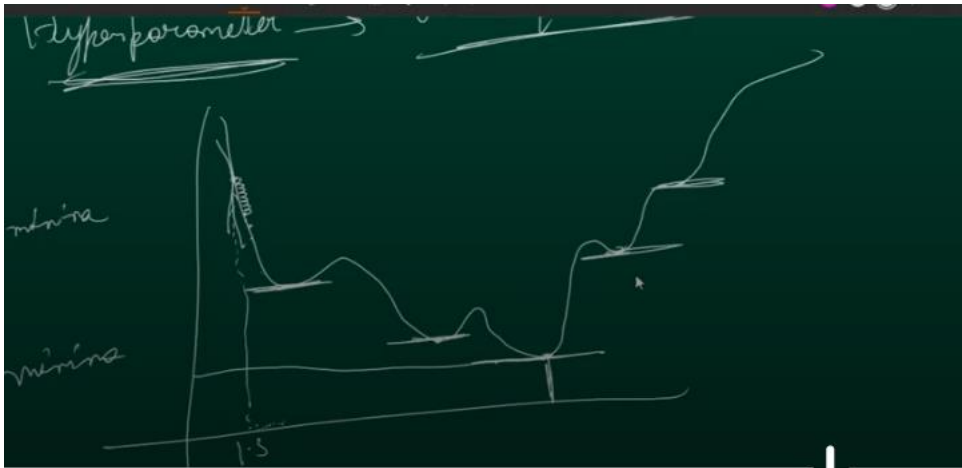
And if this value is too high, then it might miss the minimum.

This is again hyperparameter tuning.



Now model's task is to reach global minima.

Now if the learning rate is too low, then it will get stuck at the local minima thinking it reached the global minima.



In all the above point where we see small horizontal line, the differentiation is 0.

So, η should be the value that it can skip these local minima and also not so big that it miss the global minima.

Adam optimizer optimizes weights by gradient descent.

So, while giving optimizer, u can define learning rate value.

Now, there are many varieties of gradient descent.

1. Batch gd- lets suppose there are 100 rows, epoch 10, 1 batch=100 rows

Then in 1 epoch, for 100 rows , 100 predictions will be made fist in forward pass.

After the prediction for all the 100 rows are made then backward passes will happen

And weight updation will happen. And this will happen total 10 epoch times.

. So weight updations are happening only 10 times.

2. Stochastic GD-100 rows, 10 epochs, it considers all the rows as single batch.

So total 100 batches we have. So 100 rows 100 batches. For every single row,

Fwd pass will happen, prediction will be done and backward pass will happen

i.e. weight updation. But this 1 batch is running for how many epochs- 10 times.

So, 1 batch is running for 10 times and for 1 batch weight updation is happening 10 times.

So the weight updation is happening for 10×100 times i.e. 1000 times.

Next is mini batch—here we decide the batch_size, this we can give

During fit. If 100 rows, 10 batches, so number of rows in each batch=10.

So, 1st batch containing 10 rows is passed, prediction is made for them and backward weight updation for these 10 rows. So 10 times weight will be updated for these 10 rows.

Then the 2nd batch will go and total 10 epochs weights will be updated.

So for 10 batches, 10 epochs, 100 times weights will be updated.

Loss function for regression=mse

For categorical data, binary cross_entropy.= summation- $y \log \hat{y} - (1-y) \log(1-\hat{y})$

Similarly softmax also called as sparse categorical crossentropy for multiclass classification.

The image shows handwritten mathematical formulas on a green background. The first formula is $J = \text{MSE} = \frac{\sum (y - \hat{y})^2}{n}$. The second formula is $J = \sum -y \log \hat{y} - (1-y) \log(1-\hat{y})$, with an arrow pointing to it from the text "Binary cross entropy". The third formula is $J = \sum_{i=1}^{100} \sum_{j=1}^3 -y \log \hat{y}$, which is enclosed in a box, with an arrow pointing to it from the text "Sparse categorical cross entropy".

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

```
or loss="sparse_categorical_crossentropy"
```

```
or loss="mean_squared_error"
```

Batch GD- has 1 batch—epochs=10, so,10 times weight updation will be done

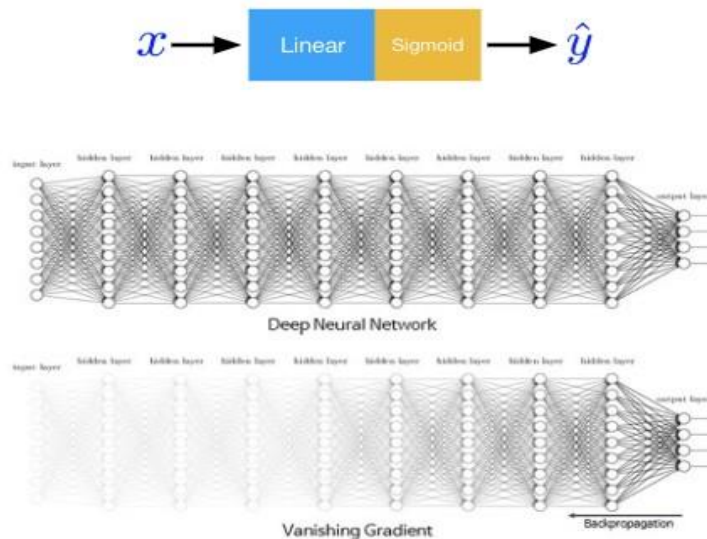
Mini Batch- for 1000 rows ,10 batch, 10 epochs, so rows in 1 batch=1000/10=100. So 100 rows

So fwd propagation for 100 rows, predictions for these 100 rows. Then 1 backpropagation for weight updation. Now this is the 1st epoch for batch 1. So like this total 10 epochs will happen for this mini batch.

So for 1 batch 10 times weights are updated. So for mini batch weight updation=
batch_size*epoch=10*10=100

But as we go back, we see some gradients values becoming 0. This is vanishing gradient problem.

Sigmoid: Vanishing Gradient Problem



i.e. model itself is vanishing and not getting trained.

Steps to Handle Vanishing Gradient:

- Reduce model depth(complexity).

- Use ReLU as activation function instead of sigmoid and Tanh.
- Proper weight initialization.
- Batch Normalization.

Vanishing_gradient.ipynb on make_moons dataset from sklearn.datasets import make_moons

The weights are not getting updated anymore.

Glorot and he are weight initialization techniques.

Overfitting----- deal with overfitting by adding dropout layer with value 0.1 to 0.4.

It switches off or dropout that many % of rows.

See dropout_notebook.ipynb on a random dataset.

Next is perform classification on multiclass classification.

It has digits in picture format and we have to predict what that digit is.

Mnist_classification.ipynb – dataset is imported from keras.

X_test.shape-- (10000, 28, 28) – 28*28 is 28 pixels in x axis and 28 pixels in y-axis of the image in width and height.

model.add(Flatten(input_shape=(28,28))) – converts 2 dim data into 1 dimension i.e. 28*28.

model.add(Dense(10,activation='softmax'))- 10 classes , so 10 nodes in output layer.

y_pred = y_prob.argmax(axis=1) – give the maximum argument value.

Axis=1, because, if u check y_prob.shape, the shape is (10000,10). So, out of 10 probabilities in axis 1,

We want the highest one.

Next, plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

the graph is showing a lot of difference in loss and validation loss is showing overfitting.

Plt.legend(["train","validation"]) give blue color for train, orange-validation.

U have to do assignment on fashion mnist.

(X_train,y_train),(X_test,y_test) = keras.datasets.fashion_mnist.load_data()

U can search keras fashion mnist for knowing the labels. Don't see solution there.

Or from Kaggle mobile price prediction – it is a multi class classification problem. Price_range has 4 different values. File name- train.csv in folder.

os.getcwd() gives the current working directory.

os.chdir() changes directory.

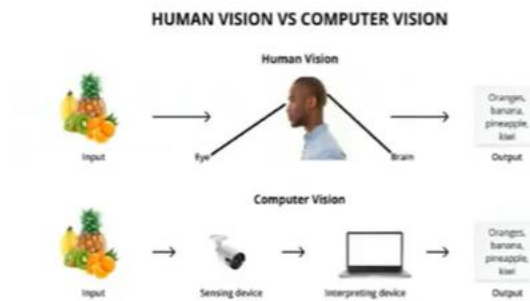
Next is computer vision and open cv

Field of AI that deals with images/videos.

1.0 Introduction to Computer Vision → AI Images / videos

Computer vision is the process of providing the ability to understand the context of images and videos to computers and converting images and videos into mathematical representation which helps computers to work on artificial intelligence.

Computer vision is collection of many processes like :



Applications- image segmentation. Segmenting the image based on object.

1.4 Application for Computer Vision

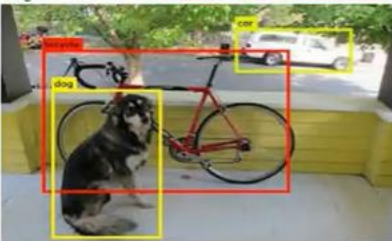
Application

- Image segmentation



Object Detection – trying to find in image with the help of bounding boxes

- Object detection



- Classification of images

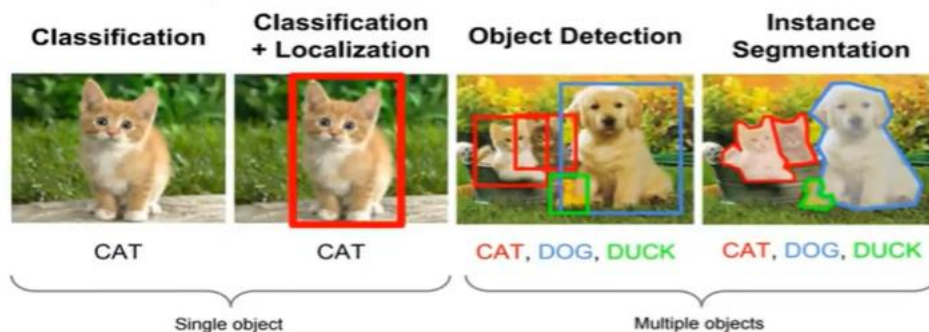


Image classification. Now only classification doesnot give location i.e. no bounding boxes.

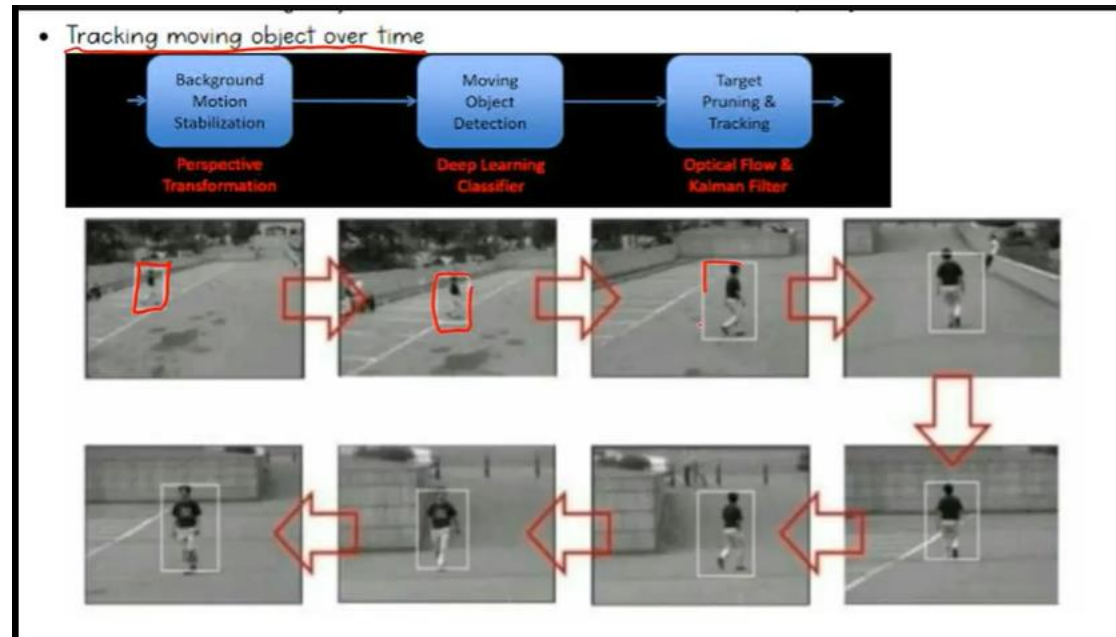
Now we have multiclass classification and multi label classification.

Multi class is when we predict more than 2 classes. For eg. Predict on dog cat horse.

It cannot be both or all.

But in multilabel- in a single image frame if both cat and dog are there , then model will o/p both the labels. Again there are no bounding boxes location o/p here.

Tracking moving object- the bounding box also move.



Face detection and face recognition. Optical character recognition.



Detecting face and putting that in numerical features and store in database.

Face recognition recognizes if the given face is the same face as stored in the database.

Software and application available to us for doing this task is OpenCV.

OpenCV stands for **Open Source Computer Vision** and is a library of functions which is useful in real time computer vision application programming. The term Computer vision is used for a subject of performing the analysis of digital images and videos using a computer program. Computer vision is an important constituent of modern disciplines such as artificial intelligence and machine learning.

Originally developed by Intel, OpenCV is a cross platform library written in C++ but also has a C Interface Wrappers for OpenCV which have been developed for many other programming languages such as Java and Python. In this tutorial, functionality of OpenCV's Python library will be described.

OpenCV-Python

OpenCV-Python is a Python wrapper around C++ implementation of OpenCV library. It makes use of NumPy library for numerical operations and is a rapid prototyping tool for computer vision problems.

OpenCV-Python is a cross-platform library, available for use on all Operating System (OS) platforms including, Windows, Linux, MacOS and Android. OpenCV also supports the Graphics Processing Unit (GPU) acceleration.

At the start OpenCv was developed for working with python.

Later it was developed to work with python.

OpenCV is an open source software I.e its free to use.

OpenCV installation-

- In most of the cases, using pip should be sufficient to install OpenCV-Python on your computer.

The command which is used to install pip is as follows:

```
pip install opencv-python
```

Performing this installation in a new virtual environment is recommended. The current version of OpenCV-Python is 4.5.1.48 and it can be verified by following command:

```
>>> import cv2
>>> cv2.__version__
'4.5.1'
```

Since OpenCV-Python relies on NumPy, it is also installed automatically. Optionally, you may install Matplotlib for rendering certain graphical output.

Open anaconda prompt or cmd

```

C:\Users\hp>pip install opencv-python
Defaulting to user installation because normal site-packages is not writeable
Collecting opencv-python
  Downloading opencv_python-4.11.0.86-cp37-abi3-win_amd64.whl.metadata (20 kB)
Requirement already satisfied: numpy>=1.21.2 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.12_qbz5n2kfra8p0\localcache\local-packages\python312\site-packages (from opencv-python) (1.26.4)
Downloading opencv_python-4.11.0.86-cp37-abi3-win_amd64.whl (39.5 MB)
----- 39.5/39.5 MB 11.4 MB/s eta 0:00:00
Installing collected packages: opencv-python
Successfully installed opencv-python-4.11.0.86

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: C:\Users\hp\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip

C:\Users\hp>

```

In colab also u can install like this only. But preference is to use jupyter.

Jupyter installation on cmd prompt -- pip install notebook

Running jupyter file -- python -m notebook

Keep some images in folder like- cats,dopg,m,oip,rl,UIEBe

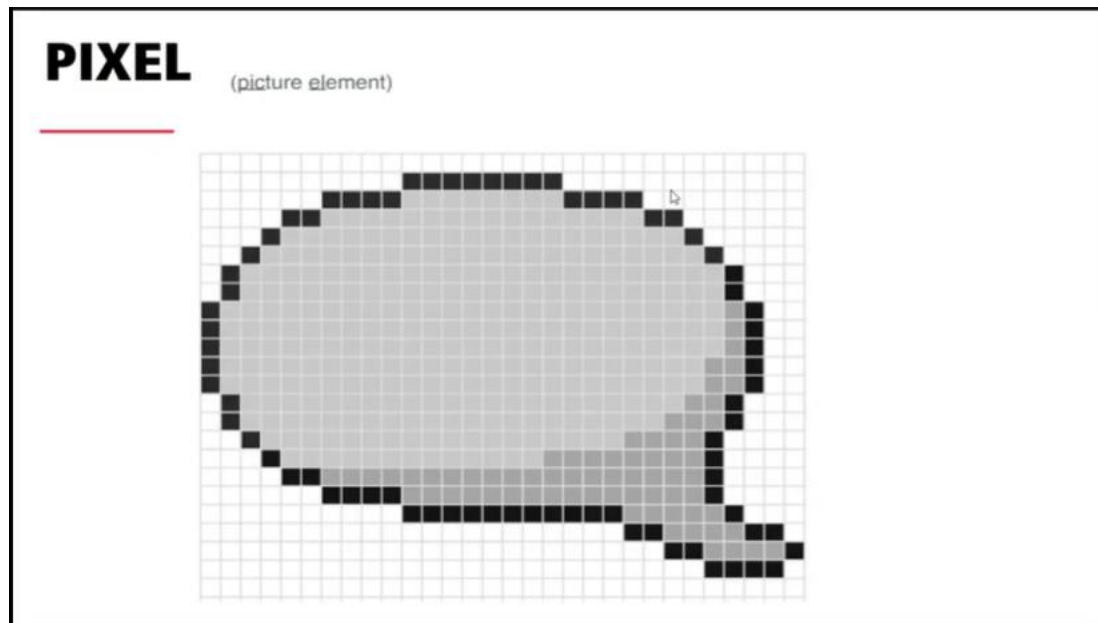
And video- vtest.

So, 1st understanding, what is an image in terms of machine..

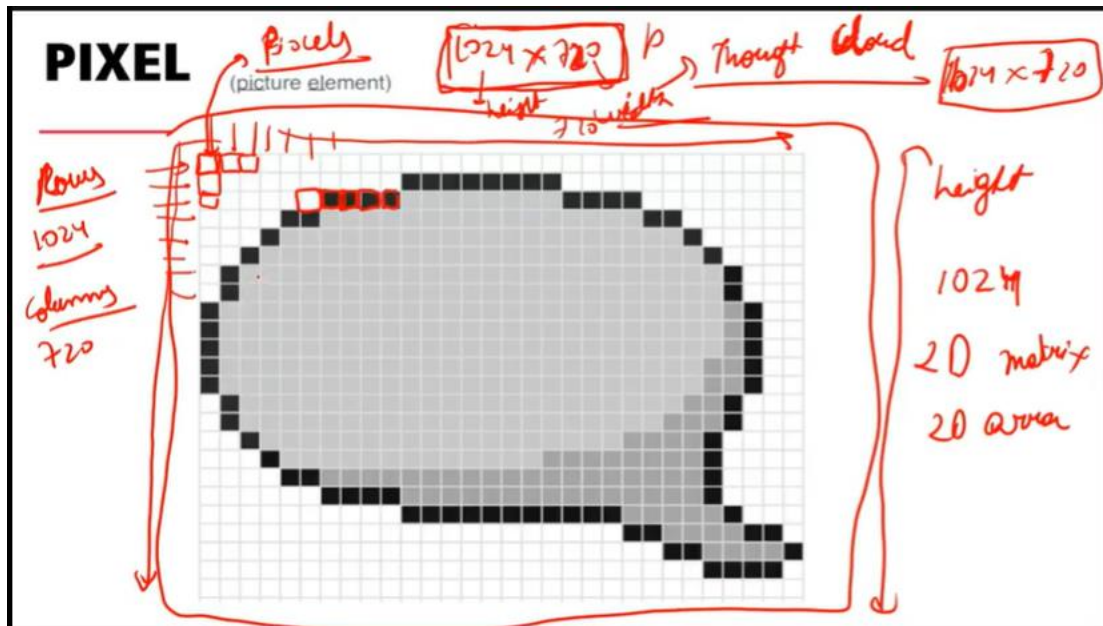
Images are of 2 types—greyscale and colored image.

GreyScale has 1 channel, where as colored image has 3 channels(R,G,B)

GreyScale image for eg, this is a thought cloud—



We can see small squares here. These squares are called as pixels.

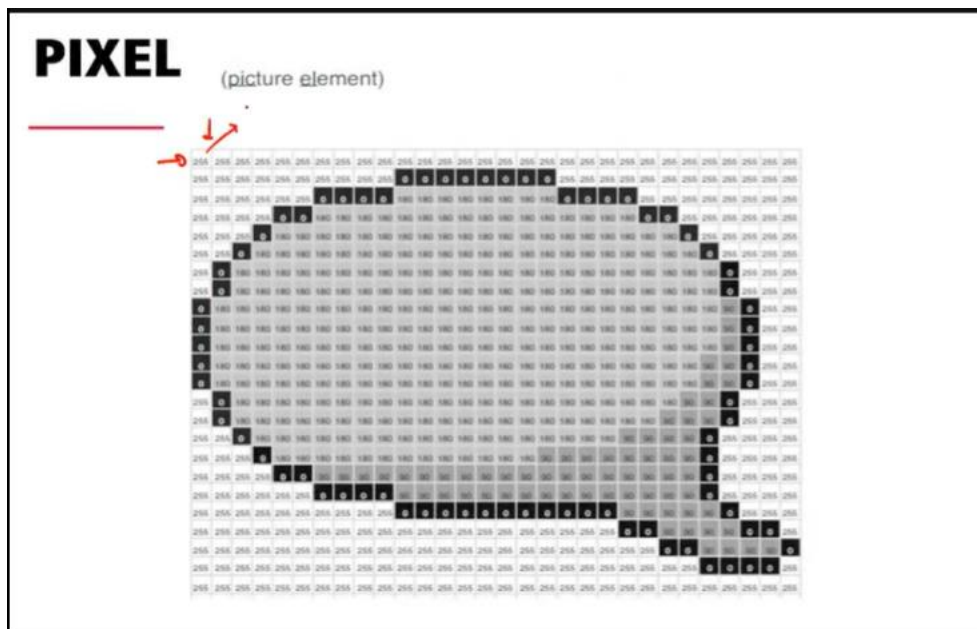


When we say the size of the picture is (1024*720), 1024 is the height and 720 is width.

Number of rows are 1024 and number of cols are 720.

Now every pixel is attached with some number, which is from 0-255.

In the greyscale image A pixel value of 0 is black, and a value of 255 is white. And in between values give grey.

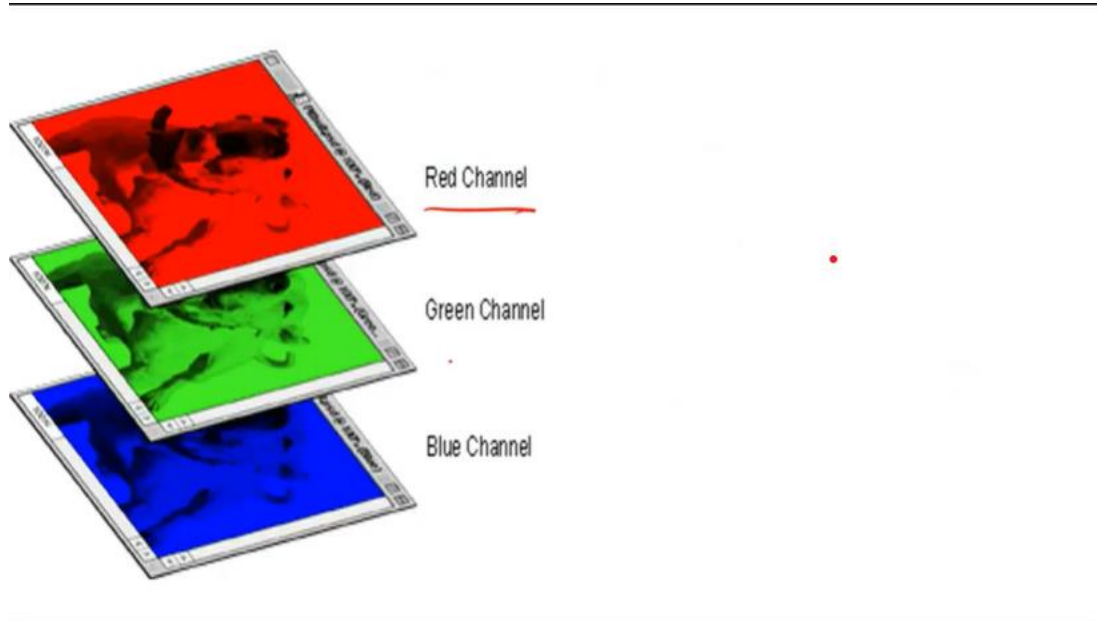


The numbers of these pixels give the color. The number closer to 255 is more whitish and that closer to 0 is blackish.

Now grayscale image has only 1 channel. That means it's a 2D array. Here the number of rows and columns give dimension of the array. So we will need numpy for understanding some basic concept.

You can say image is nothing but a numpy array.

In colored channel, we have red channel, green channel and blue channel.

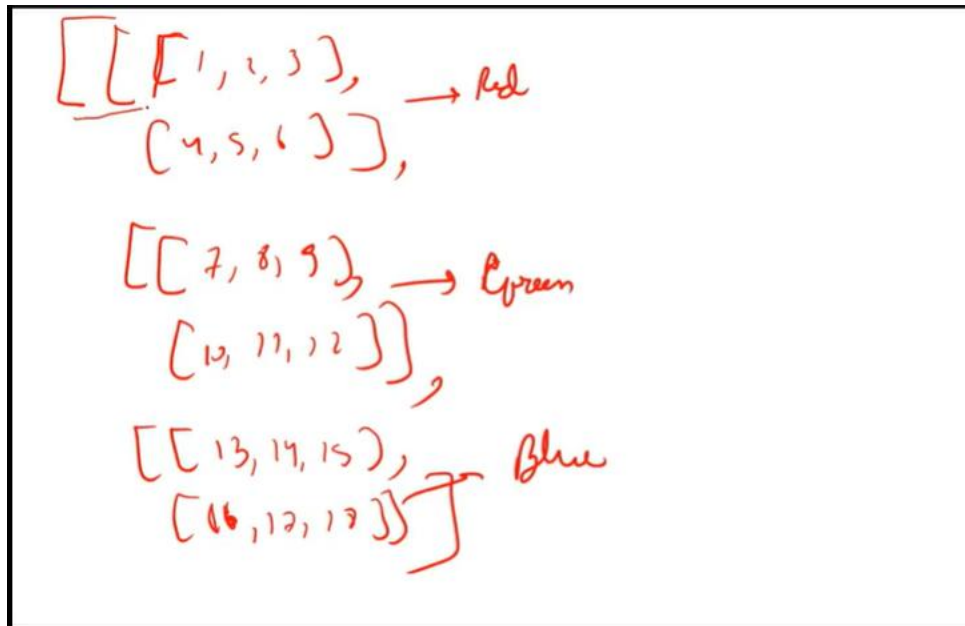


On combining the values from these 3 channels valuing from 0-255, a colorful image is created.

This red channel in itself is a 2D array. Lly, green and blue.

Now on stacking these channels one above another gives 3 D array.

Something like-



RGB channels too have pixel values from 0-255.

In Red channel, 0 – no red color, 255 – pure red color, and in between values will give the intensity of red. Same goes for red and green color.



cv_introduction_updated.ipynb

```
import numpy as np
```

```
import cv2 --- cv is for computer vision and 2 is the version.
```

With the help of open cv2 u can show the image, and also with the help of matplotlib, u can show the image.

In images, we actually have 4 channels, 3 are r,g,b and another one is alpha channel.

It is rarely available and available to only few images. Its kind of a transparent channel.

And works as the backgrounds of image. For eg UIEBE.png image. Then the background apart from of tree can be visible with the help of alpha channel.

Now,

Reading the image – `cv2.imread(path of image, u want to read this image as original/colored image, grey scale image or unchanged, that means if alpha channel is present then it will be read in the same way as present.`

Now, if u will see the values in `color_img` variable after file is read, it looks like a numpy array.

Now important—when the image is read by opencv, then the image is in BGR format.

So if we will write `color_img[0]`, it will give the 1st channel.

Ily, `gray_img` is 2 D, whereas `alpha_img` has 4 channels., which u can check by

`alpha_img_1.shape`. but if u will print something like `alpha_img_1`, it doesn't show the 4th channel.

-No need of writing `cv2.IMREAD_COLOR`, `cv2.IMREAD_GRAYSCALE`, u can even give numbers.

Like 1,0,-1

Now printing the image is done by `imshow`.

In `cv2.imshow('Gray',gray_img)`— 'Gray' is the name of the window. 2nd argument is the numpy array. Here it is `gray_img`.

`Cv2.waitKey(0)`—the code will stop running only when I will click cross.

So this basically gives second for that many seconds. U can give 1000, 5000 etc.

`Cv2.destroyAllWindows()` closes the open window.

In 1 cell, we can open multiple windows. Put `waitkey` only ones.

By pressing keyboard key also, u can close the window.

Eg if `cv2.waitKey(0) & 0xFF == ord('r')`: -- this means if I cross the window then close it or if I press r from my keyboard, close it.

0xFF is for pressing keys from keyboard.

Now u can also show the image bu using `matplotlib`.

`plt.imshow(color_img)` – but this image o/p is little different then the original.

Because, when opencv reads the image, the created numpy array is created in the form of bgr. Original images are in the format RGB. So that's why when this numpy array is printed using plt, it gives a weird image. So if we want to show image using plt,

```
RGB_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2RGB)

plt.imshow(RGB_img)
```

The x and y axis are inverted in o/p i.e. 0 at top and x is changing from left to right.

Next is saving image. `imwrite` is for saving the image.

`cv2.imwrite('edited.jpg', color_img)` – 1st arg is name of the image, 2nd arg is the image u want to save.

Now if u have any RGB image then u can convert it by passing to `cv2.COLOR_RGB2BGR`.

Because open cv needs image in BGR format only.

Next 2 operations are applying brightness to the image and contrast to the image.

In brightness image has more white effect, Now image is a numpy array and if we want to add brightness, then add some value to each pixel then they all will move towards white.

Brightness and contrast

```
new_gray_img=gray_img+70

cv2.imshow("new_color_img", new_gray_img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

while in contrast—this is achieved through multiplication.

Now $0 * \text{any value} = 0$, so black will be black, rest all will change.

Now, the maximum value of our pixel is 255 only, so if after $*$, value is greater than 255, then cap it to 255.

Next is extracting separate channels from image.

Using `cv2.split()`

Next is region of interest on image. So by using cropping, So for this we use the concept of numpy array slicing.

Now lets say we have our Neeraj Chopra image and we want to crop this face.

Now, the shape of this image is (266, 474, 3) as we can see on the axis of plotted image too.

So, if we see the face is from 0th row to some 150th row, and col is from 180 to 300 something .

So with the help of this axis , we can roi.

Next is extracting height and width of image.

So, image.shape gives (266, 474, 3)

So, image.shape[:2] will give (266,473)

Next is resizing the image—

Shrink the image means low height and width, stretch the image means more height and width.

Now 1 very imp concept comes into play i.e. aspect ratio=h/w

Eg if h=120, w=60, now lets suppose I want to increase h to 200 but I want aspect ratio to be same.

So aspect ratio here = $120/60=2/1=2$

Now I know, ht=200, then w=? $w_{new}=h/ratio = 200/2 = 100$,

Lly , if $w_{new}=80$, $h_{new}=ratio*w \rightarrow 2*80 = 160$

So, resizing can be done in many ways. For eg—

1. Preserve Aspect Ratio (height to width ratio of image is preserved)
 - Downscale (Decrease the size of the image)
 - Upscale (Increase the size of the image)
2. Do not preserve Aspect Ratio
 - Resize only the width (Increase or decrease the width of the image keeping height unchanged)
 - Resize only the height (Increase or decrease the height of the image keeping width unchanged)
3. Resize to specific width and height

So, 1st is downscale. Means short it. Now when we give scale_percent,

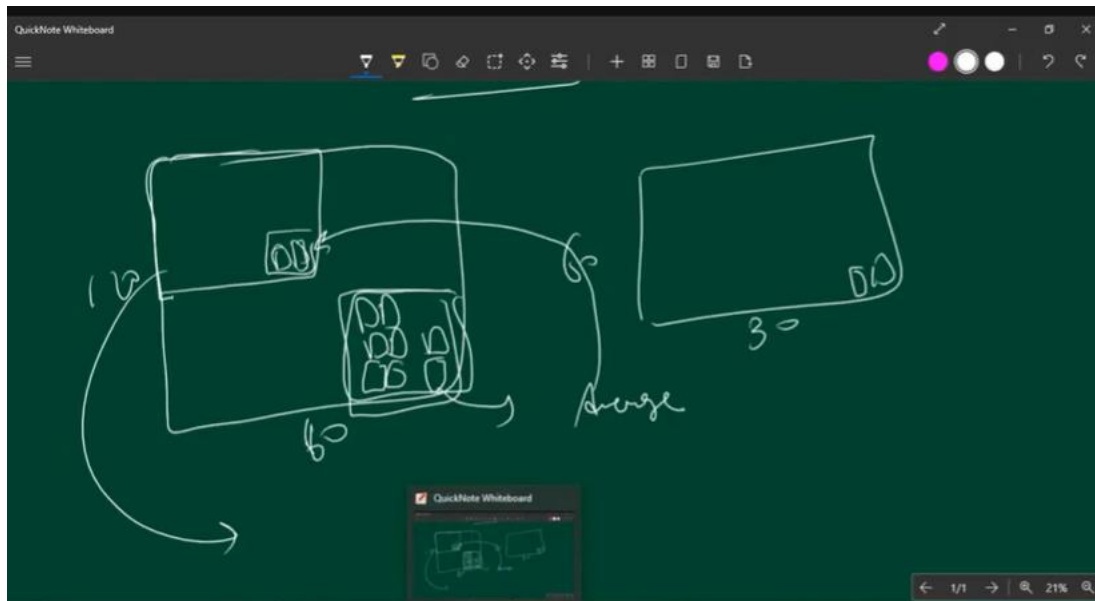
If scale_percent value=100, then that means original image. That means size (266,474) only.

But if scale_percent=60, then $266*60/100$ and $474*60/100$.

So, using cv2.resize() i.e.

`resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)` – 1st arg is input image, 2nd arg is target shape, 3rd arg is interpolation, which we will discuss after a while.

Interpolation – here u r adjusting pixels.



So this everything is happening in backend we only need to mention which interpolation we r using.

3 best interpolation techniques are—

cv2.INTER_AREA: This is used when we need to shrink an image. – for downscaling

cv2.INTER_CUBIC: This is slow but more efficient. – for either upscaling/downscaling.

cv2.INTER_LINEAR: This is primarily used when zooming is required. This is the default interpolation technique in OpenCV. – for upscaling

Next is rotation of the image. There are 2 methods that we need to run at the same time to rotate the image. They are `getRotationMatrix2D()` and `warpAffine()`

Here we first need to find the center of the image.

So, center of image is half of width and half of height. So we r doing floor division.

In `getRotationMatrix2D()` 1st pass center of the image, then the angle, next is scaling argument.

Now, this method gives a 2*3 matrix. And when u will multiply this numpy array with original numpy array, it will give the rotation. This multiplication is performed by `warpAffine()`

In which we pass original numpy array, rotated matrix and after rotation new image width and height.

So currently we have kept original [w,h] only. Changing width and height changes window size.

So to fill the black space of window by image, scale component plays the role.

Next is creating a rectangle or a bounding box.

`rectangle = cv2.rectangle(img, (320, 60), (430, 155), (50, 150, 250), 3)`—1st pass original image,

Next is to write something on rectangle `putText()` is used.

Next is Video Capture—So 1st we will see how to turn on webcam. In `cv2.VideoCapture()` is there.

Here u can pass either of the 2 things. Number of device basically indexing of the device.

Laptop's webcam indexing is 0. If some cctv camera is attached then that camera's indexing will be something else like -1 or -2 or some other number depending on the configuration which number will be assigned.

Next if u pass the path of the video file, then it will read the video from this path.

So 1st understanding what is video in terms of opencv. So, it's a sequence or collection of images.

In terms of videos we say something like 60 frames per second, 30 frames per second.

This frame is nothing but an image.

So 60 frames per second means passing 60 images in 1 second from the front of our eyes.

At such speed, our mind feels like something is moving instead of being still.

So 20-22 fps can be considered as a video. For lesser than that, it becomes more like a slide show.

So, in video processing, we will learn, 1. How to capture video from front/webcam. 2. Capture video from local video file. 3. Generate frames from videos i.e. extraction, 4. Create a video from collection of images, 5. Save a video.

For Video Capture – `cv2.VideoCapture(0)` – 0 is to mention whether reading is from webcam or from some other device. So we r doing by webcam. If instead of 0, path of the file is sent, then it will run the file on that path.

```
cap = cv2.VideoCapture(0)
```

So, video feed from webcam is stored in variable `cap`.

Next,

```
while(cap.isOpened()) – means till the time webcam is opened, continue this loop.
```

Inside the loop; `ret, frame = cap.read()` – video feed inside `cap` is read. On reading, it gives 2 variables `ret, frame`. `ret` is Boolean and contains true/false. True tells yes I can read something from the webcam. So when the webcam doesn't start, this value is false. Next is `frame` – so `cap.read()` reads the frame one by one and store it in `frame` variable.

So this frame is a single channel web frame if web cam is giving gray input, which is generally not the case, So the webcam which capture colourful images has BGR format and is a 3 channel numpy array.

if `ret:` – here im saying by the time my camera is reading something and `ret` value is true, then

convert this frame from bgr to gray. i.e. converting from 3 channels to single channel.

By `cv2.imshow("Text",gray)` , u can show this in a window.

Then if `cv2.waitKey(1) & 0xFF == ord('q')`: we r saying if user enters q from the keyboard, then it will break and will come out of this while loop. And if ret value is false, then anyways, it will break the loop and will come out of while loop.

Next is `cap.release()`, if we will not release the cap then opencv will go on using my webcam.

And if we want to use our camera somewhere else, then we wont be able to use it. So release it to stop the camera getting used by opencv. `cv2.destroyAllWindows()`, So by `imshow()` we open windows, so by `cv2.destroyAllWindows()` all the windows are destroyed.

Next is showing rotated video.

Next is capturing video from a path.

`while(cap.isOpened())`: -- while video is opened and it is there to play, to stop the video in between press q and close the window.

Saving frames or images from videos—

So when we say 60 fps, these frames are actually in number format. So I want that when the frame is in multiple of 5 like 5,10,15.... At that time save the frame. Means when 1st frame is read it is frame 0, for 2nd frame frame number will be 1,.. this is how opencv numbers these images. So when I say

`i = 0` # means 1st frame index is 0 .

after processing each frame, I am increasing my variable i by 1.

`i += 1`.

Now if ith value is divisible by 5, then save the frame by using `cv2.imwrite()` . i.e.

if `i%5 == 0`:

```
cv2.imwrite(f"saved_frames/frame_{i}.jpg", frame)
```

inside folder saved_frames and the name of the frame inside folder will be frame_{i}.jpg

So for a frame of 30 frames/sec and video length of 20 seconds, $30 \times 20 = 600$ frames.

So if we will save all 600 of them it will be huge. So u can decide in how many frames u want to save the frames.

We have put f in `cv2.imwrite` to treat {i} as place holder and not string.

Next is saving videos.

`fourcc = cv2.VideoWriter_fourcc(*'XVID')` – Here fourcc is called as a video codec. So we have created a `VideoWriter_fourcc` object. And then I said the video that I want will be in format `*'XVID'`.

And here, `out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))` we saved our video.

Rest everything is same as before.

Now do remember this is a video without any sound because u r only accessing the camera.

Open cv has no option to access microphone.

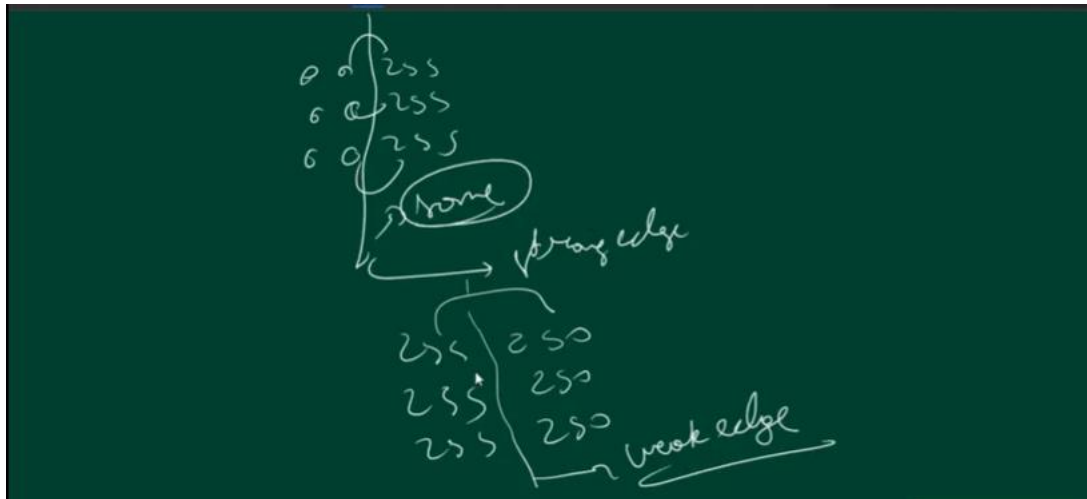
Next is creating videos from collection of images.

Next is edge detection. With the help of edge detection machine can understand some object is there and label it. So when we will see CNN, we will know how this labelling can be done or classification can be done.

So, what is edge—edge is the change in intensity of color. CNN has 1 architecture called as facenet, it has taken all the models . By understanding the pattern of these edges i.e. eyes, nose, etc, the machine tries to categorize it to eyes, nose, etc.

So 0 is black and 255 is white. So we have strong edge and weak edge.

Strong edge is like



Weak edge is like gradual decay from 255 to 250 to 245 like that.

Now if u want to find the edges , opencv has 2 methods. Canny and Sobel.

Remember whenever u want to detect edges, convert image into gray scale, and then do it.

This makes problem much easier as the image will have black and white colors.

So when this tries to catch change in the gradient of edge, there are some weak edges and there are some strong edges. SO on detecting the change in intensity, If that value is greater than that threshold, then that edge is strong edge, so capture that. Between the threshold1 and threshold 2 value is considered as weak edge. And lesser than threshold1 values are no edge.

Now these threshold values are not pixel values which should be in the range 0-255. It is a threshold for values generated by canny algorithm.

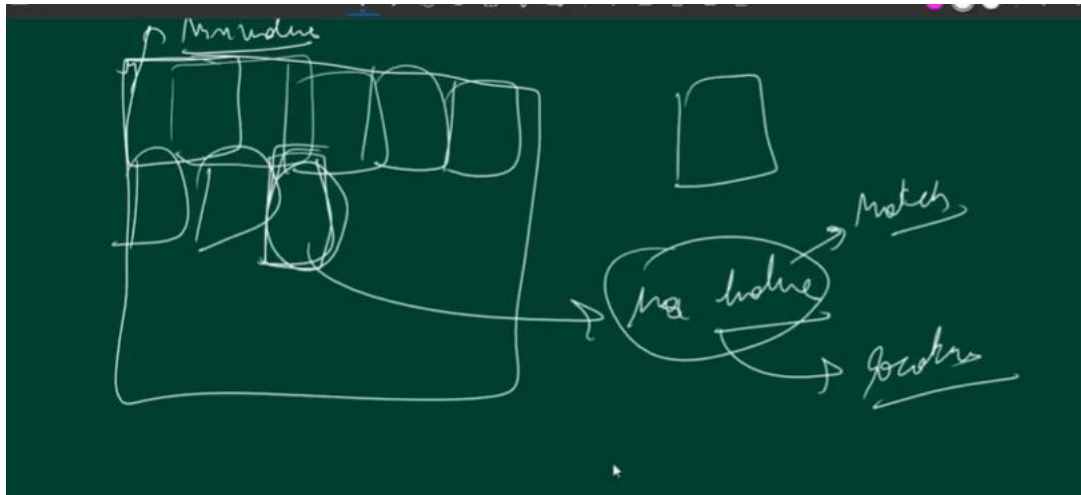
lly, for sobel, vertical horizontal edges can be found.

Canny finds all the edges, there's no concept of hori and vertical.

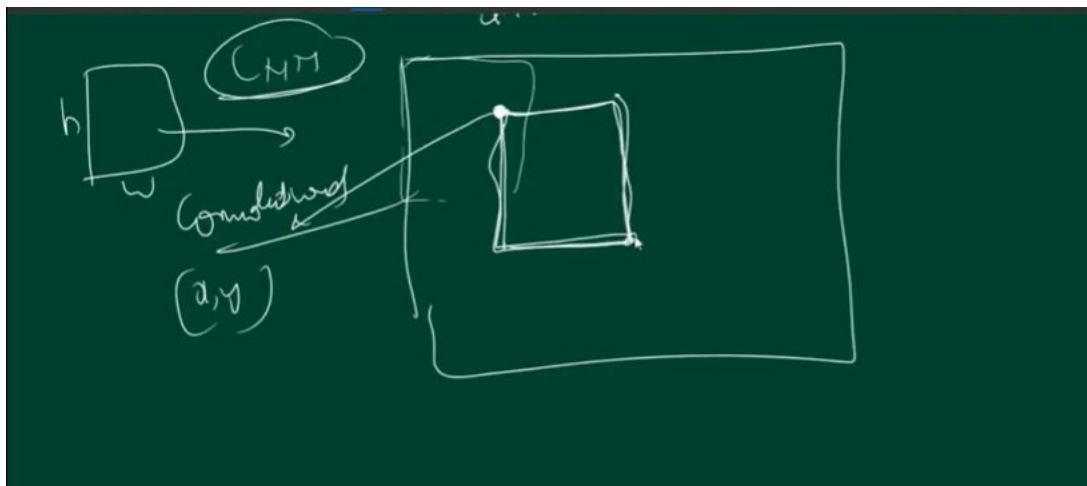
Next is template matching. U have a template and u want to find that template in an image.

Lets's suppose our template is m.jpg and we want to find it in rl.png.

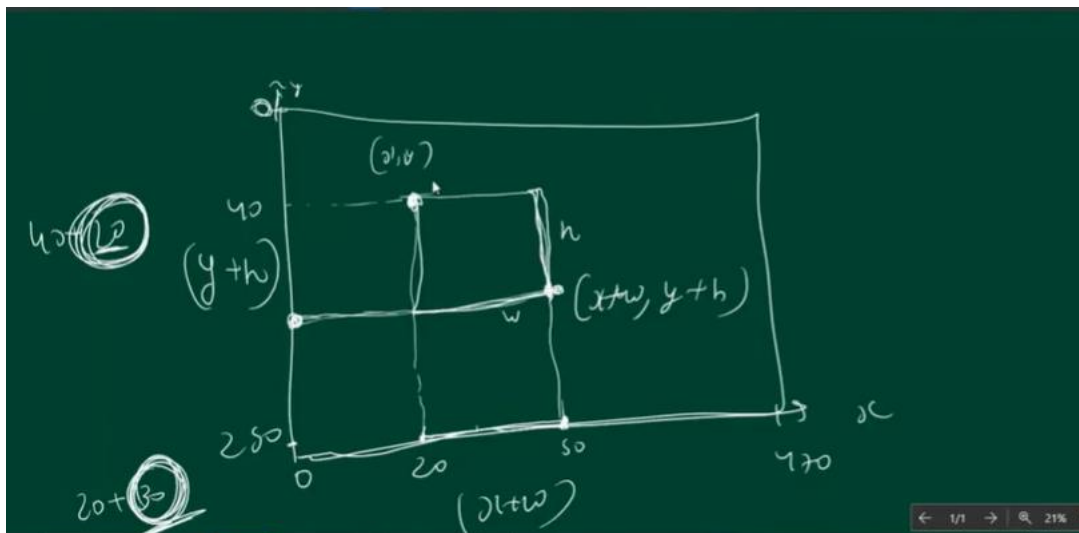
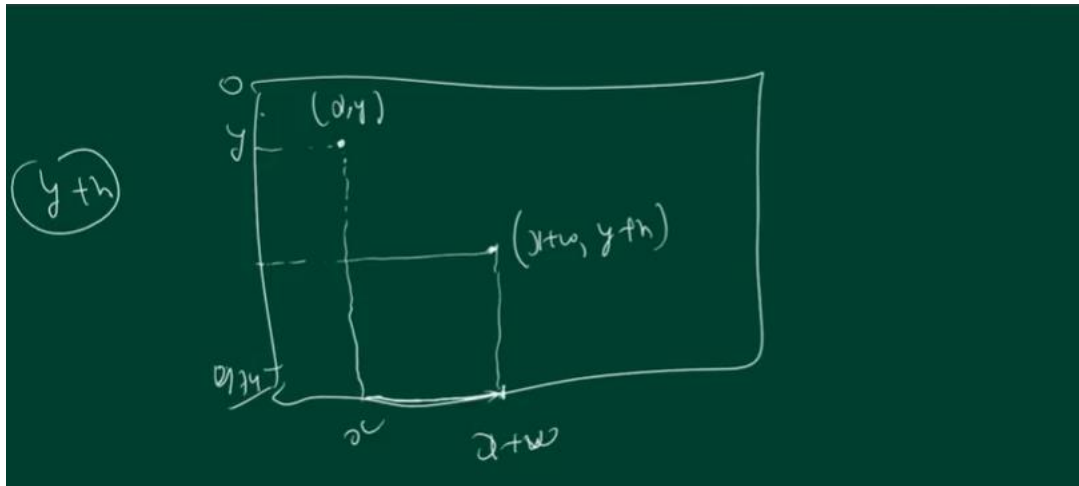
`cv2.matchTemplate()`.



`Cv2.match()` gives location



But I want other corner point too, so that I will get with the help of width and height.



Next is Face Detection— i.e detecting human face in an image or video using haarcascade.

This is a model that detect a frontal face and has been trained on millions and billions of frontal face. haarcascade_frontalface_default.xml file contains that trained code. So to read that code,

haarcascade_frontalface_default.xml enables reading face from the haarcascade library using haarcascade model. And that object we have stored in face_cascade. Next we read the image `img = cv2.imread(r'C:\Users\hp\Documents\AI\DL_MyTry\Resources\rl.png')` and store it in a numpy array.

And convert it into gray scale. Then in face_cascade we have `face_cascade.detectMultiScale()`

This is for detecting multiple faces in an image. It needs 3 things, 1st is gray scale image, Next is scale factor. So basically , haarcascade generates the pyramid of the image, and checks in which pyramid layer face is clearly visible? So basically here we r decreasing image at the rate of 1.3%, as

the haarcascade is trained on some face size, so as the pyramid layer decrease to that size, and the face is detected, those coordinates are given. i.e.

[[471 97 112 112]

[158 103 115 115]]

Where 1st image coordinates are x=471,y=97,w=112,h=112 . So as explained earlier with the help of these values, bounding box can be found.

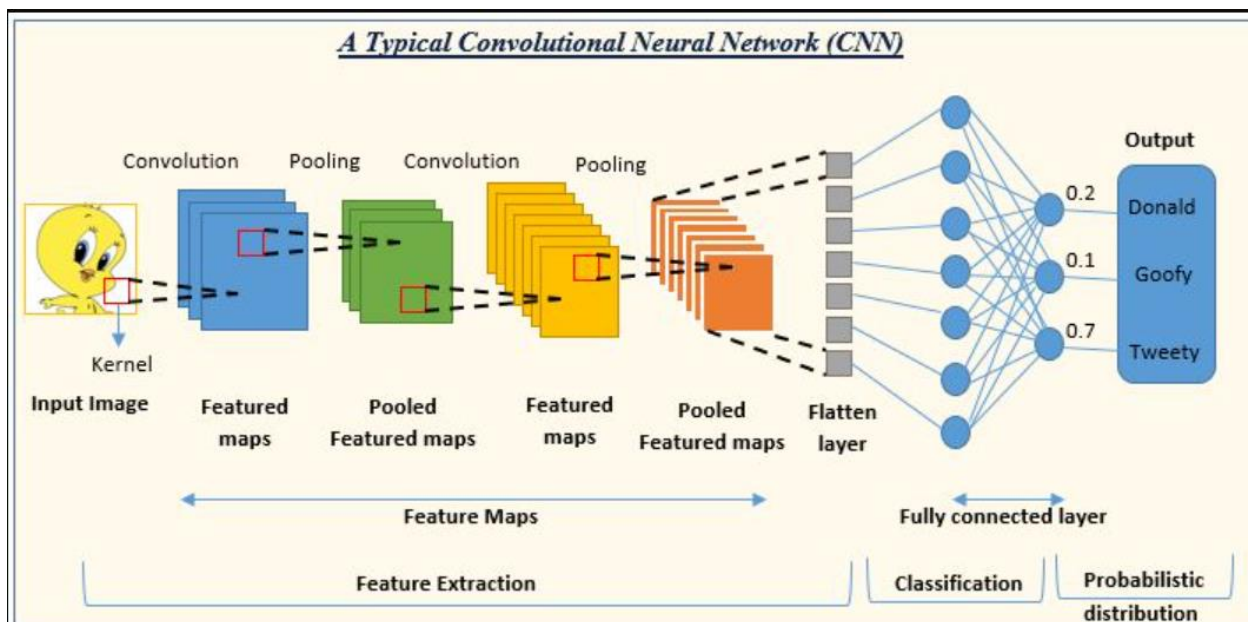
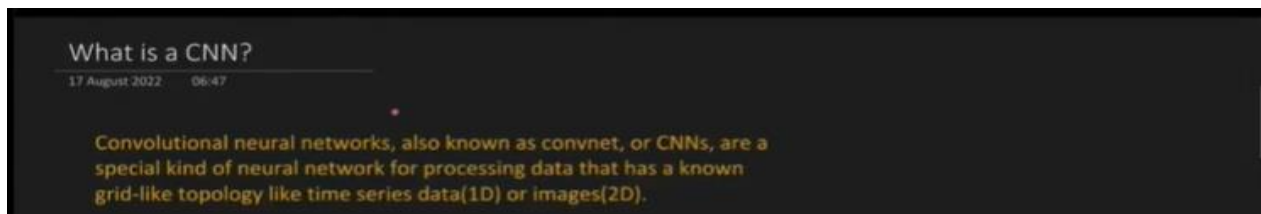
So, cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2) – img,coordinates,color, rect thickness.

Also, haarcascade is only for face detection. Its not for face verification.

Next is detecting my face from the webcam.

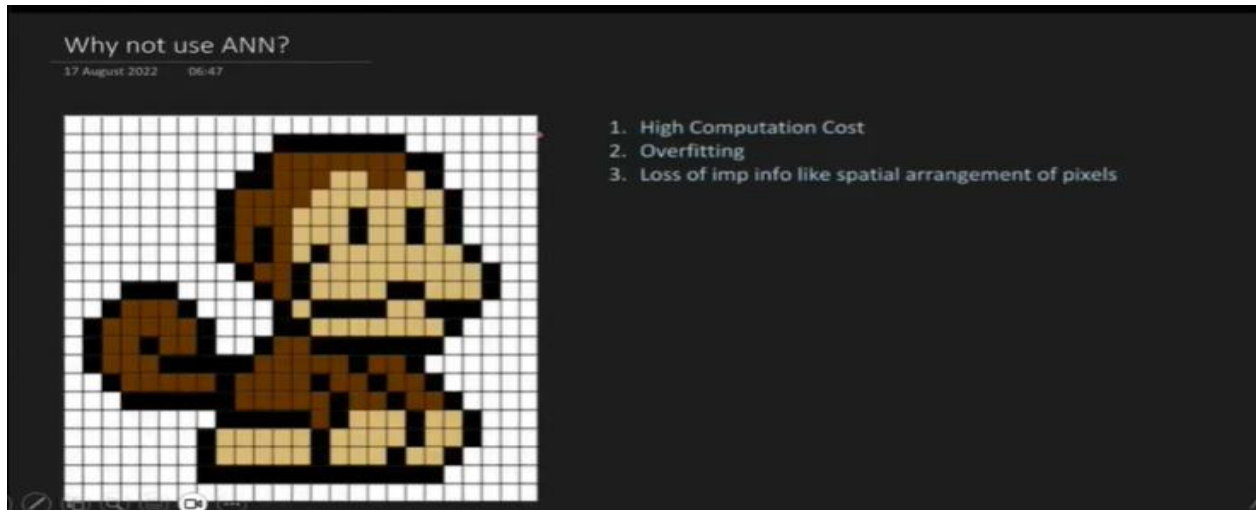
So, 1st taking the code to open the webcam.

#-----



So CNN is specifically developed for image and video data. The convolutional layers and pooling layer helps in extracting features. This detect edges. Flatten array means making 2d into 1 d

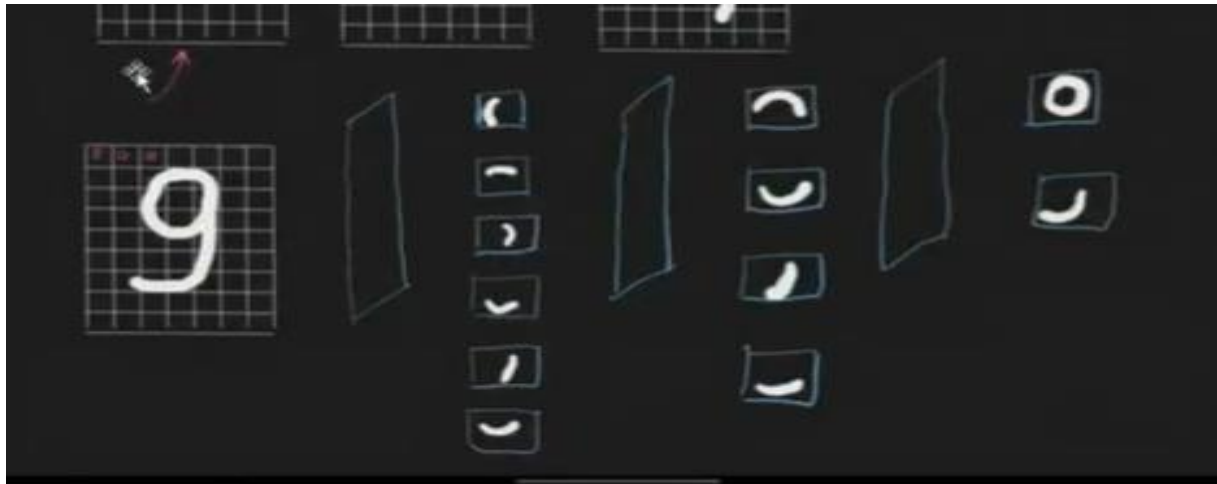
array. Small ANN makes for Fully connected layer. So, before ANN we have applied CNN. So not applying ANN only on images and videos because-- ANN is based on finding weights and biases.



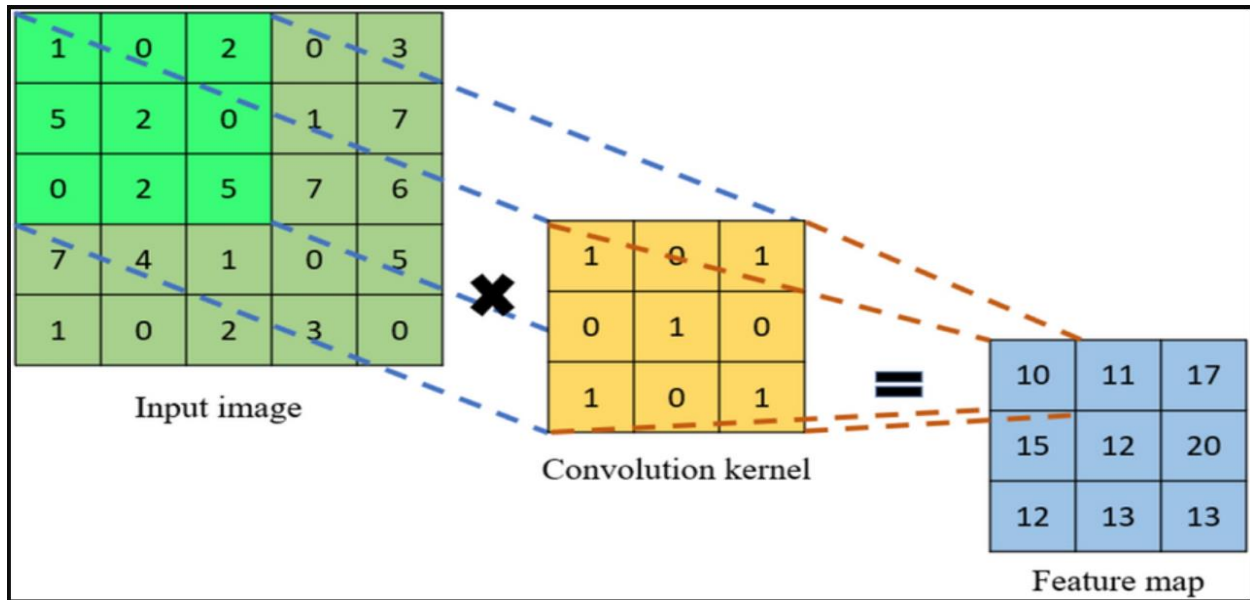
Now MNIST data contains hand written digits.

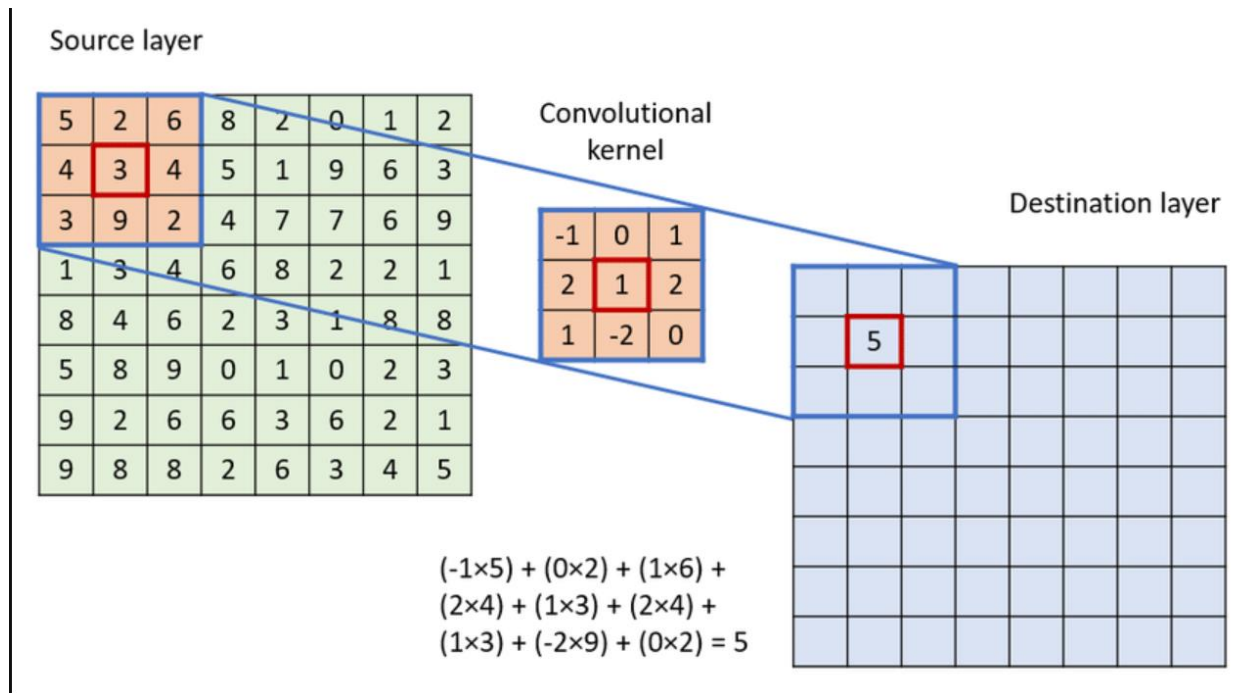
We have solved this problem using ANN. So for applying ANN, on 2D data of 28×28 pixels. So we flattened this image first i.e. complete 1st row vertically, then 2nd row vertically. So all the rows are vertically stacked. So our input data has 28×28 features. We applied ANN on it and got an accuracy of 94-98%. Which is very good. But there are certain problems with ANN as showed above. $28 \times 28 = 784$. So total 784 features are there. So in case when we have bigger image like 1024×1024 , it results in many more features, resulting in high computational cost. Model will take a lot of time and consume lots of resources like a lot of memory in ram. ANN is open to overfitting. 3rd is loss of imp info like spatial arrangement of pixels i.e. for eg in above image we have monkey, so distance between pixels of eyes and nose. So in ANN when we vertically stack all the rows, we r going to lose such kind of information. Also, if we will notice, monkey is little bit on the right side, if it will shift to left, then again ANN might not work effectively. So, ANN is not good for extracting features from images.

So, CNN basically extracts features by combining detected edges. And try to make sense that if such kind of edge is there, then it can belong to this class. For eg.



So cnn basically tries to find the pattern of edges with the help of convolution filters. This filter is also called as kernel.





These convolutional filters or kernels are responsible for finding the edges. These filters are in the form of numpy arrays.

FREQUENTLY USED 3x3 CONVOLUTION KERNELS

1	1	1
1	1	1
1	1	1

Averaging

-1	0	1
-1	0	1
-1	0	1

Vertical Edges

-1	-1	-1
0	0	0
1	1	1

Horizontal Edges

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

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

Sobel Templates

0	-1	0
-1	4	-1
0	-1	0

Laplacian

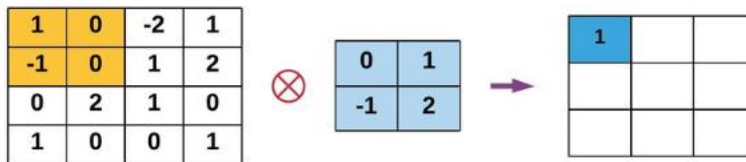
September 15, 1998

1

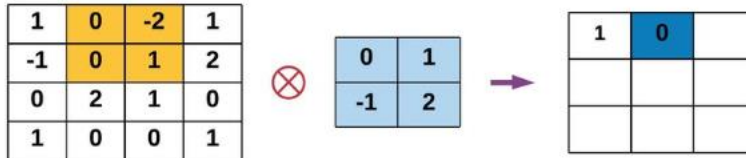
Kernel size is usually the odd number like 3*3,5*5 and 3*3 is the most common one.

SO the kernel is applied on image like—

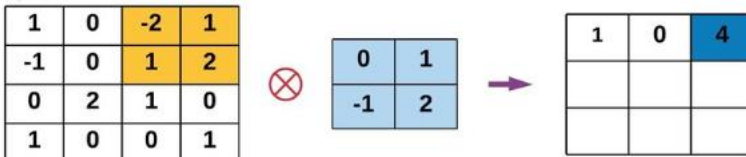
Step-1



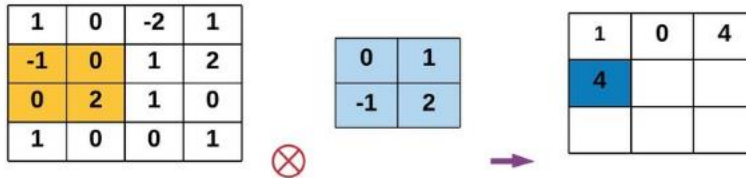
Step-2



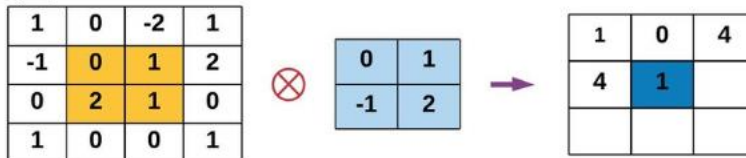
Step-3



Step-4



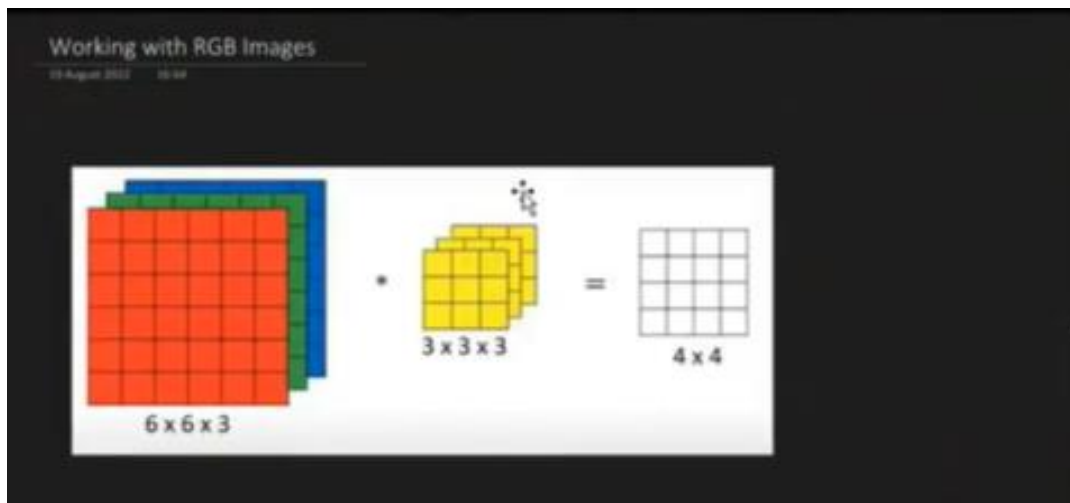
Step-5



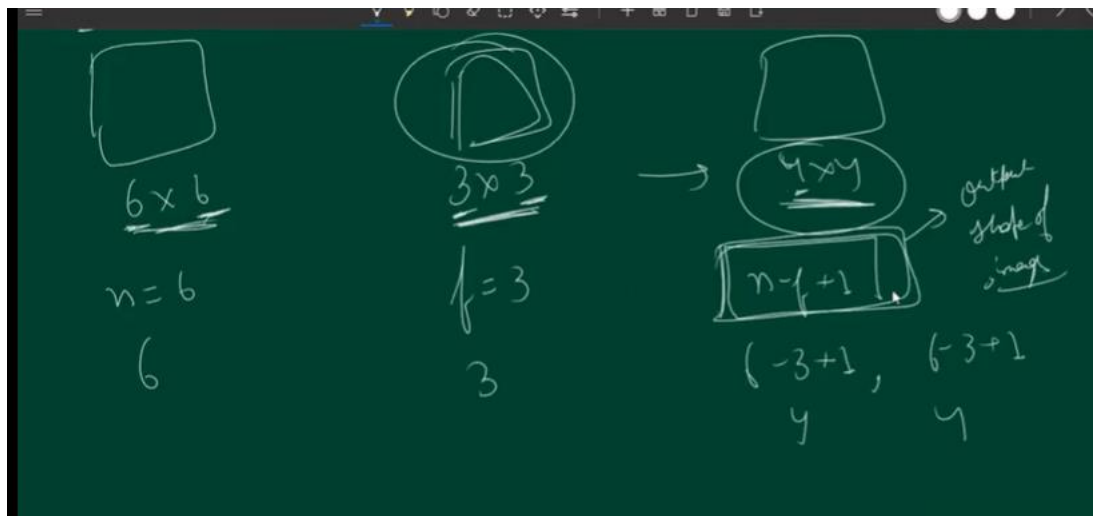
This process is called as convolution. Now u don't have to decide which kernel to apply. Model decides itself. Just like as in ANN, model itself decides bias and weights. In the same way.

So, the model decides the filter based on backpropagation.

U can apply cnn on both gray and colored image. On the gray image single channel is there so single kernel is applied. Whereas On the colored image, 3 channels are there so, 3 kernels each of same size.



It's a 3 channel filter. Now after applying convolutional filter, the image size decreases, but by how much?

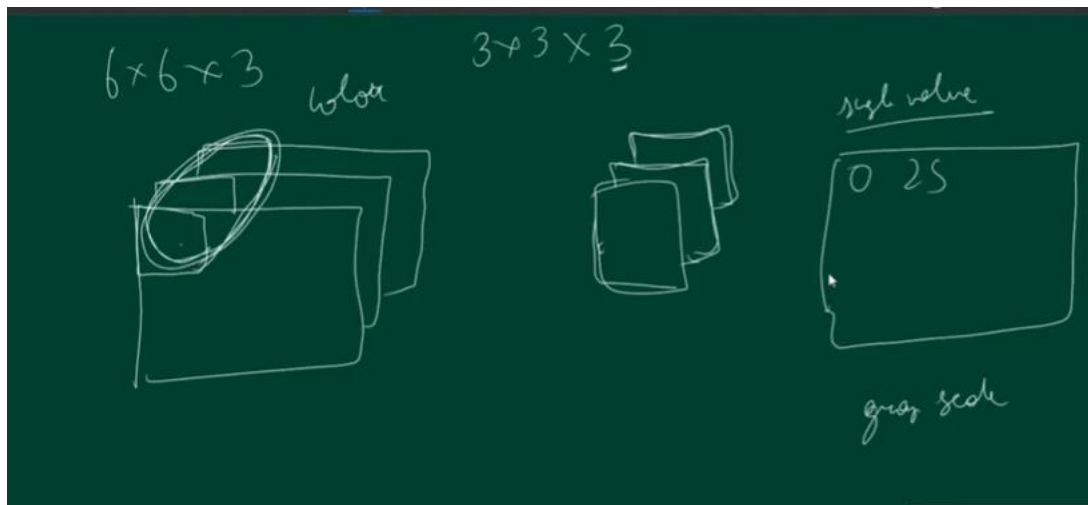


N is the shape of original image, f is filter $\rightarrow n-f+1$

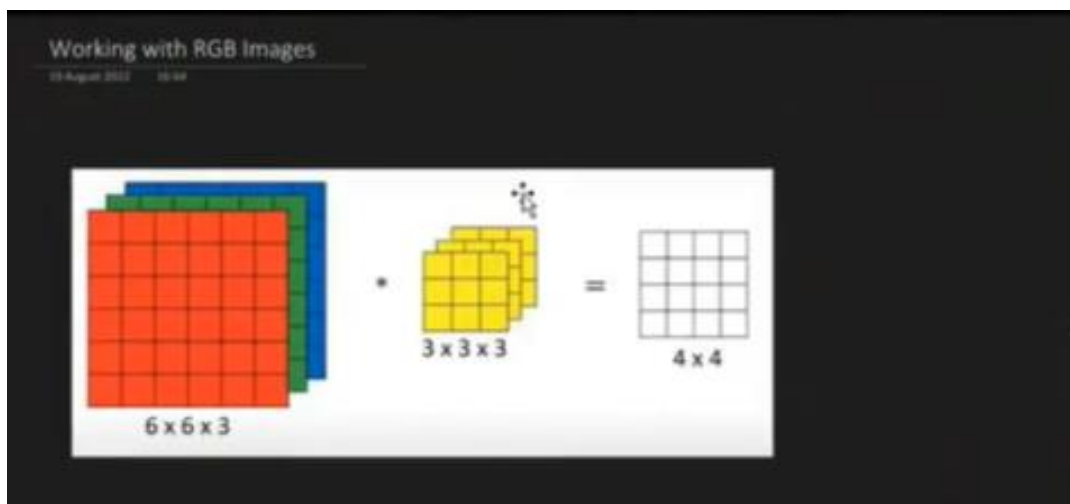
So we can see, after applying convolutional filter, image size is decreasing.

So, in RGB image as we have 3 channels, 1 each for RGB, we will have 3 filters too.

So if original image size = $6 \times 6 \times 3$ and filter size = $3 \times 3 \times 3$, apply filter on each channel and then sum all the values, this will give a single value i.e. ultimately we r getting a gray scale image.

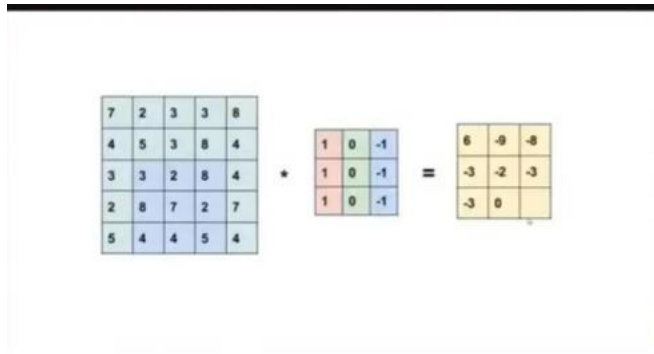


Like here –

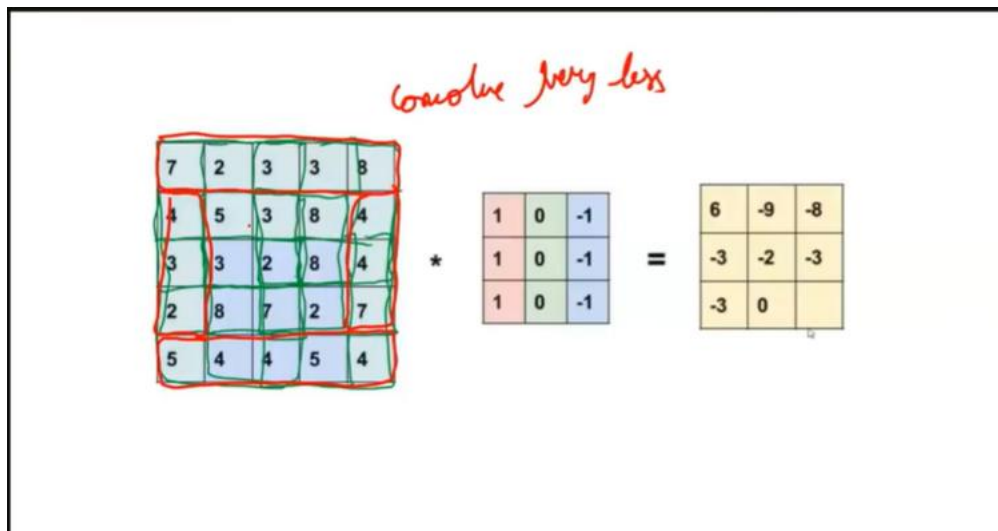


I.e. the extracted feature is in a gray scale image.

There is another concept called as padding. As we r applying convolutional filter, the feature map is of lesser than original image size and we r also losing the information. So how we r losing the information?



On applying conv filter, we notice that on the border pixels, filter is applied less times than on the other features.



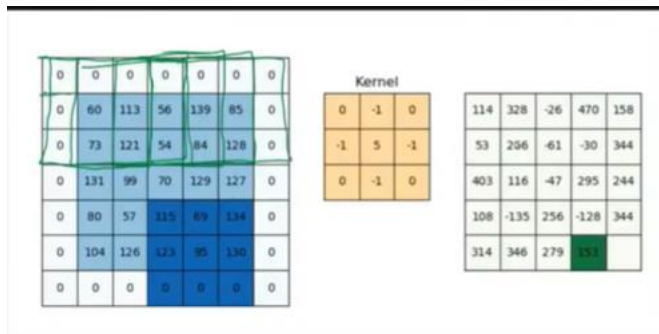
i.e. middle values feature extraction is done more times than border.

Now on 6×6 conv $3 \times 3 = 4 \times 4$ and if on 4×4 conv $3 \times 3 = 4 - 3 + 1 = 2 \times 2$. So now image size is 2×2 .

So we apply padding to make sure that input image shape is equivalent to output image shape.

And also we will end up losing lesser information.

Padding is adding 0's on the border.



In case if the padding is applied on the image, then, the formula for calculation is $n+2p-f+1$.

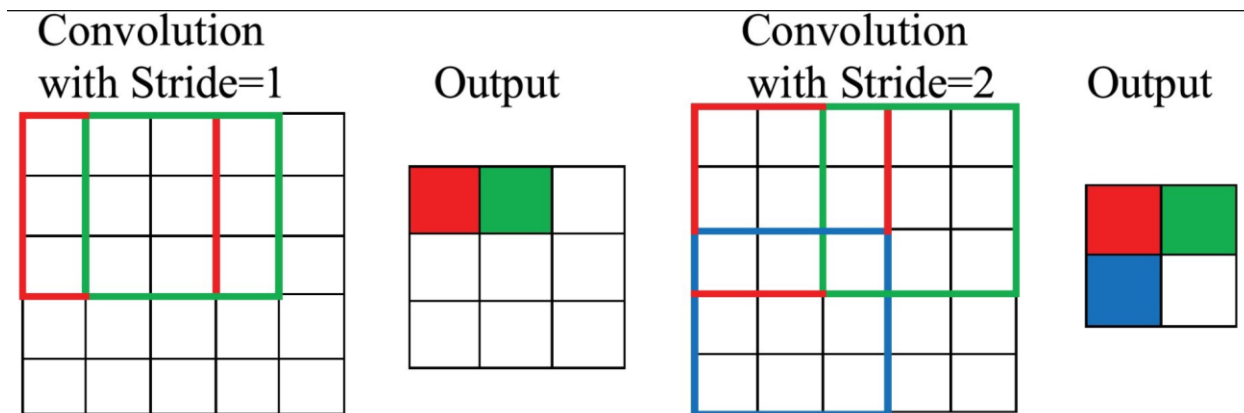
So if padding $p=1$ on $6*6$ image, $3*3$ filter, then, $6+2*1-3+1=6$

Its not necessary that padding will always be of layer 1

For eg if image size= $8*8$ and filter size= $5*5$, then if $p=1$, then $8+2*1-5+1=6$,

But this is not same as original image size. So, the padding size depends on the filter size. The padding size is automatically found by model.

There is another concept called as stride. Stride is the **number of pixels** by which the filter moves across the input **image** in CNNs. It affects the output size, computational efficiency, field of view, and downsampling of the model.



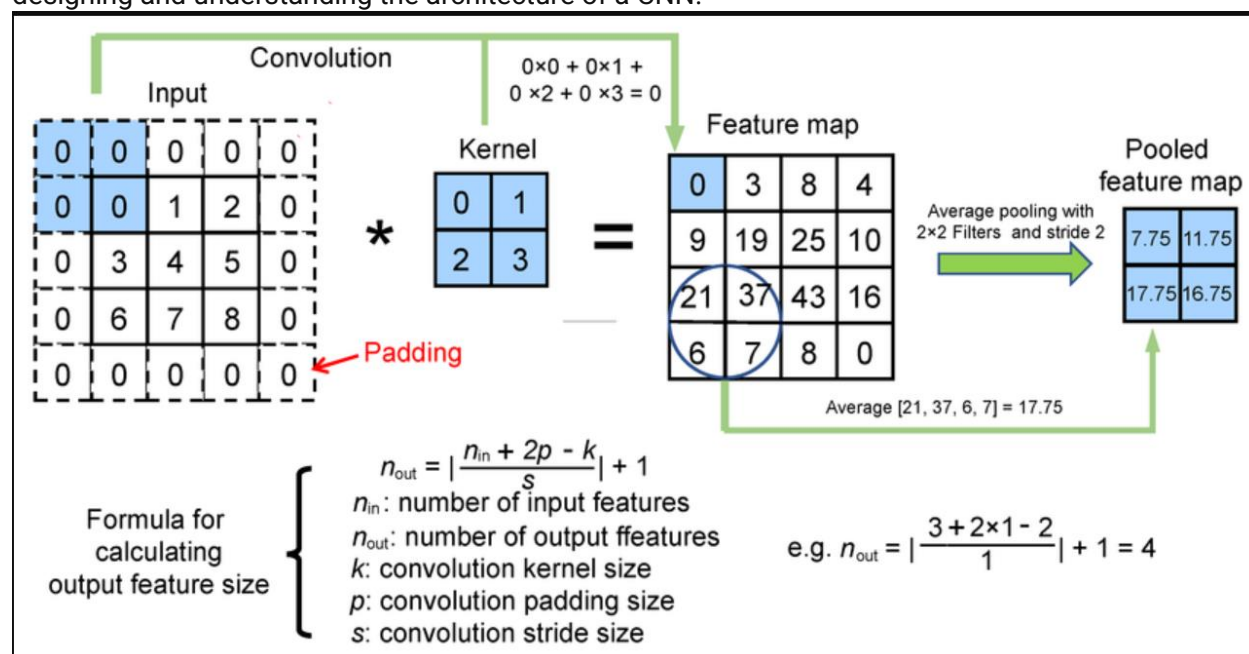
Calculating Output Size with Stride

The output size of a convolutional operation can be calculated using the following formula:

$$n_{out} = ((W - K + 2P) / S) + 1$$

Where: n_{out} is the output size, n_{in} is the input size (width or height), K is the kernel size, P is the padding and S is the stride

This formula helps to determine the dimensions of the output feature map, which is essential for designing and understanding the architecture of a CNN.



This feature map is a gray scale image with some extracted features. Where the extracted fetures are something like—

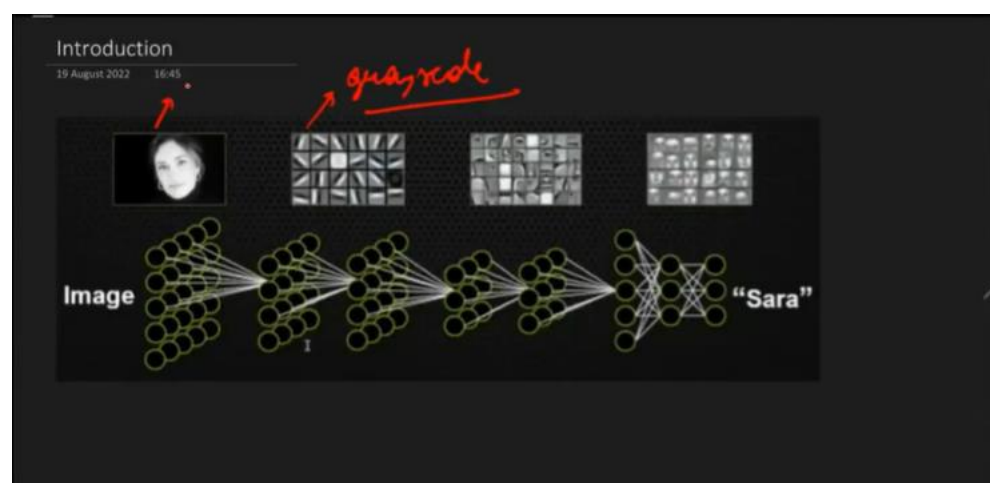
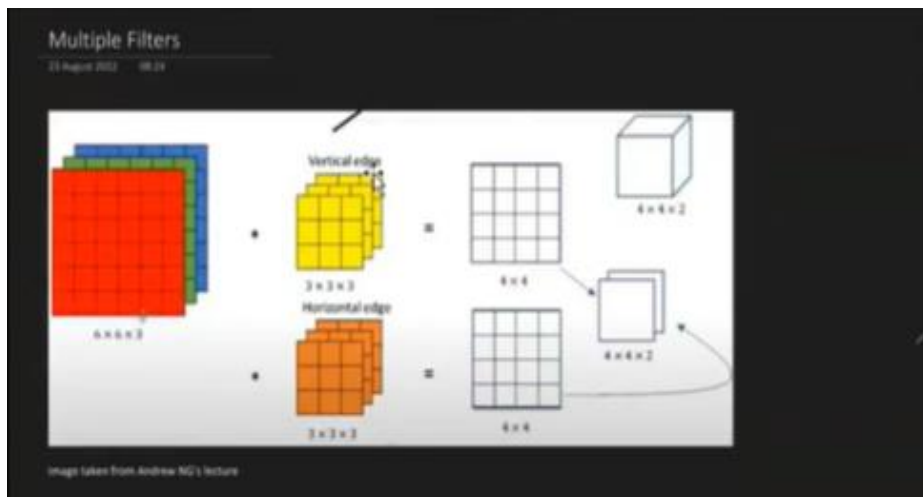


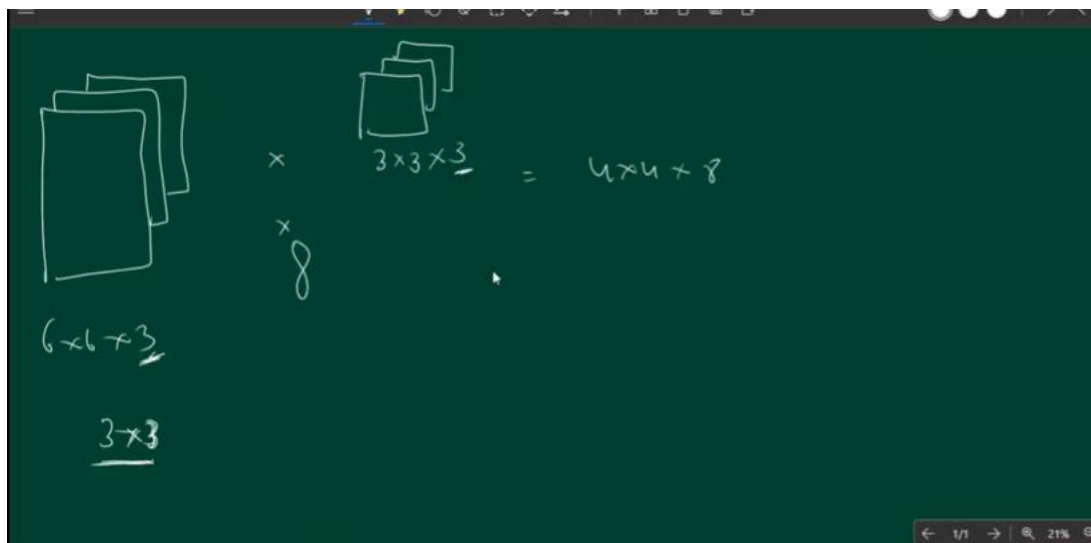
Image Classification using CNN.ipynb on mnist data. Do on colab, change runtime to gpu.

In cnn architecture, we apply Dense,Conv2D,Flatten, MaxPooling2D.

Flatten is applied at the end after applying all the convolutional layers. Conv2D is the conv filter where we will give number of filters. This is because on a single image multiple filters are applied and not just single filter as shown above, For eg.



So, in $4 \times 4 \times 2$, 2 is the number of filters applied on the image. Now next conv filter we will apply on this $4 \times 4 \times 2$. So here the filter size can be any number but number of channels depends on the input channels. For eg if original image size is 6×6 and number of channels = 3 then $6 \times 6 \times 3$, filter to be applied size if 3×3 , the number of channels will be equal to image channel i.e. 3 so $3 \times 3 \times 3$. And if 2 such filters will be applied, then final o/p = $4 \times 4 \times 2$, so applying 2nd conv layer on this $4 \times 4 \times 2$, the conv size = $3 \times 3 \times 2$. For eg,



i.e. 8 stacks of 4×4 filter.

The size of the filter on each layer can be different and given while defining conv layer.

Image Classification.ipynb – Do in colab ---We do perform eda on the data. 1st we see the shape of different images. It is possible that imaged in the dataset are of different sizes, like $28 \times 28, 30 \times 28, 30 \times 30$,

Then the image with the lowest shape is taken, and all the other images are made of the same shape. This is because while creating the model, u tell about how many number of input features are there. All the image given to the cnn model should have same shape.

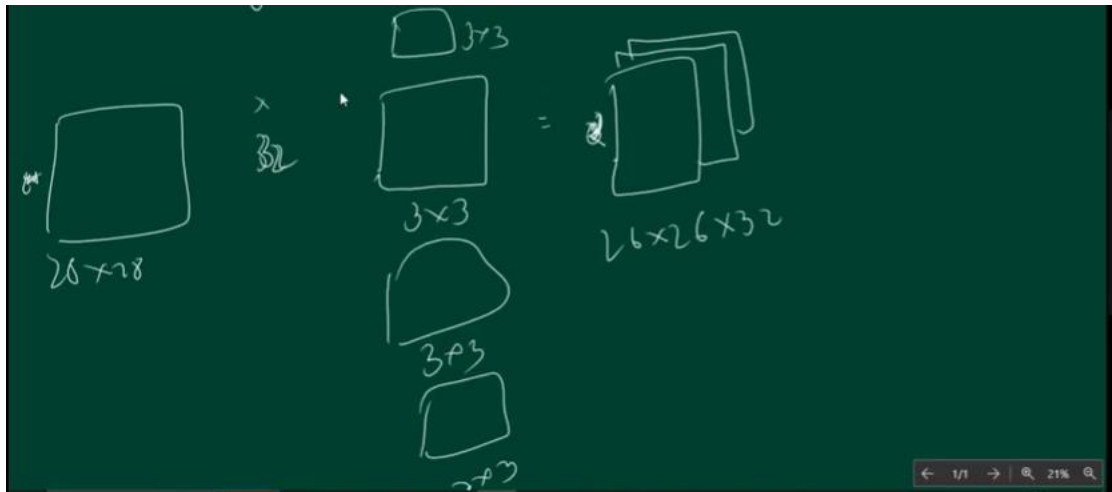
Apart from this no other eda is required . In keras this is automatically done. So this we will do later.

```
model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)))
```

 # 32 filters of size 3*3. The shape of the input image is input_shape=(28,28,1) i.e. it's a gray scale image of size 28*28.

Each image is sent 1 by 1. i.e. xtrain[0] 1st then xtrain[1]... So after applying 1st layer on 1st image,



So 26*26*32 ,

```
Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1))
```

Parameters=(3*3*1)*32+32=320

Where, **Weights:** (3*3*1)*32=288

And **Biases:** 32

Next we are applying 16 filter of size 3*3*32

```
model.add(Conv2D(16, kernel_size=(3,3), activation='relu'))
```

Here, The number of channels are 32 same as input features channels, 24*24*32 . On this we r applying 16 filters of size 3*3*32. i.e. then the ouput shape of the feature map will be 24*24*16.

```
Conv2D(16, (3,3), activation='relu')
```

(3*3*32)*16+16=4624

Next, `model.add(Conv2D(8, kernel_size=(3,3), activation='relu'))`

Here, we are applying 8 conv filter of size $3 \times 3 \times 16$ on image of size $24 \times 24 \times 16$. So output feature will be of size $22 \times 22 \times 8$ and number of parameters will be—

Conv2D(8, (3,3), activation='relu')

$(3 \times 3 \times 16) \times 8 + 8 = 1160$

Then the flatten layer is applied. Here each row is taken and stacked vertically.

`model.add(Flatten())` # Converts **(22,22,8)** → **(22×22×8 = 3872 neurons)**

`model.add(Dense(128, activation='relu'))` #— $(3872 \times 128) + 128 = 495744$

`model.add(Dense(10, activation='softmax'))` #— $(128 \times 10) + 10 = 1290$

Hence, Total Parameters: 502,138.

For training of model `model.compile`.

So this we have done without padding. In keras u can apply padding with 2 options. Valid and same.

By default padding is valid and final shape is decided by $n-f+1$.

And for same, output shape will be same as input shape. $N+2p-f+1$.

So valid doesnot add padding where as same add padding.

Input Shape: (28,28,1)

- A **grayscale image** (e.g., from the MNIST dataset)
 - **1 channel** (since it's grayscale)
-

2. Applying Convolutional Layers (Feature Extraction)

Each **Conv2D** layer has:

- **Padding = 'same'** → Output size remains the same as input
- **Kernel size = (3×3)**
- **Activation = ReLU** (Removes negative values, keeps only useful features)

So, **Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(28,28,1))**

- **32 filters** of size **(3×3)** are applied

- **Padding = 'same'** → Output remains (28,28,32)
- **No reduction in size** because padding compensates for boundary pixels

Then, `Conv2D(16, (3,3), padding='same', activation='relu')`

- **16 filters of (3×3)** applied
- **Padding = 'same'** → Output remains (28,28,16)

Next, `Conv2D(8, (3,3), padding='same', activation='relu')`

- **8 filters of (3×3)** applied
- **Padding = 'same'** → Output remains (28,28,8)

Then comes Flatten Layer

`model.add(Flatten())`

- Converts (28,28,8) → (28×28×8 = 6272 neurons)
- Makes it compatible for **fully connected layers**

4. Fully Connected Layers (Classification)

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

- **128 neurons with ReLU activation**
- Extracts **high-level patterns**

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

- **10 output neurons** (for 10 classes, e.g., digits 0-9)
- **Softmax activation** converts outputs to **probabilities**

2. Parameter Calculation

Each layer has **trainable parameters** (weights & biases).

`Conv2D(32, (3,3), padding='same', activation='relu')`

Parameters=(3×3×1)×32+32=320

- **Weights:** (3×3×1)×32=288

- **Biases:** 32

28*28*32 is input shape to next layer so conv layer applied will be **3*3*32**

Conv2D(16, (3,3), padding='same', activation='relu')

$$(3 \times 3 \times 32) \times 16 + 16 = 4624$$

28*28*16 is input shape to next layer so conv layer applied will be **3*3*16**

Conv2D(8, (3,3), padding='same', activation='relu')

$$(3 \times 3 \times 16) \times 8 + 8 = 1160$$

Output shape will be **28*28*8**

Fully Connected (Dense) Layers

- **Flatten layer** = No parameters => $28 \times 28 \times 8 = 6272$
- **Dense(128, activation='relu')** $(6272 \times 128) + 128 = 802944$
- **Dense(10, activation='softmax')** $(128 \times 10) + 10 = 1290$

Total Parameters: 810,338

Now, the data on which we r working, most of the data is in middle and in corners there is no data.

So, in such type of image we can work without padding.

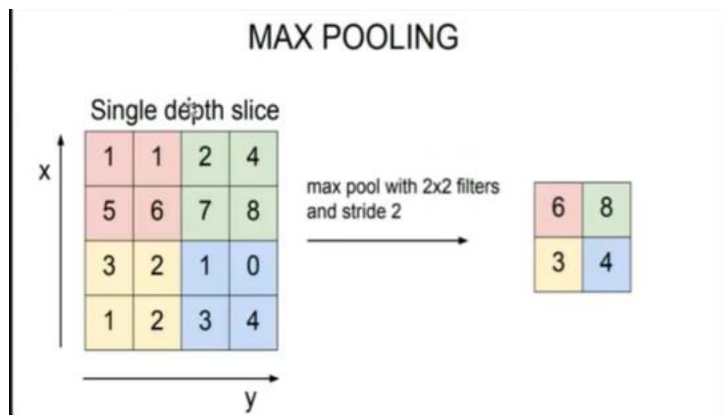
So accuracy increasing or decreasing after applying padding depends on where the important features are present.

Strides were used earlier because we have many computational power.

Its disadvantage is it is skipping the features and so it is loosing the information.

Because of which model's performance can decrease.

Next we will discuss Pooling. We have different types of pooling and most used one is MaxPooling. Then we have average pooling. Then Min Pooling. Pooling is used to enhance the features, i.e. those features which are not much visible, they are enhanced more.



Lets suppose we have an image of 28*28, padding=valid i.e. no padding applied, 1 filter size = 3*3,

Then, output shape=26*26. Now over this output, pooling is applied. So based on formula—

$(n-f)/s+1 \rightarrow (26-2)/2+1 \rightarrow 13$. Now the image size got reduced but because of the max pooling, we have maximum information and not lost. So now conv filter of size 3*3 will be applied on feature map size 13*13. Giving o/p of 11*11. Again maxpooling, so, $(11-2)/2+1 =$

$9/2+1 = 4.5+1=4+1=5$.

So now applying pooling on model – on colan – Images Classification.ipynb

Input Shape: (28,28,1)

- A grayscale image (e.g., from the MNIST dataset)
- 1 channel (since it's grayscale)

Next,Convolutional + Max Pooling Layers (Feature Extraction)

Each Conv2D layer extracts features, and MaxPooling2D reduces the spatial size.

Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1))

- 32 filters of size (3×3) are applied
- ReLU activation removes negative values
- Output Shape: (26,26,32)
 - Formula: Output Size=(Input Size–Filter Size)/Stride+1= (28–3)/1+1=26
 - (28,28,1) → (26,26,32)

Next,MaxPooling2D(pool_size=(2,2), strides=2)

- Reduces spatial dimensions by half

- `pool_size=(2,2), strides=2` → (26,26,32) → (13,13,32)

Next `Conv2D(16, (3,3), activation='relu')`

- 16 filters of (3×3) are applied
- Output Shape: (11,11,16)
 - $(13-3)/1+1=11$ $(13-3)/1 + 1 = 11$ $(13-3)/1+1=11$
 - (13,13,32) → (11,11,16)

`MaxPooling2D(pool_size=(2,2), strides=2)`

- Reduces spatial dimensions by half
 - `pool_size=(2,2), strides=2` → (11,11,16) → (5,5,16)
-

3.Flatten Layer

`model.add(Flatten())`

- Converts (5,5,16) → (5×5×16 = 400 neurons)
 - Makes it compatible for fully connected layers
-

Next, Fully Connected Layers (Classification)

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

- 128 neurons with ReLU activation
- Extracts high-level patterns

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

- 10 output neurons (for 10 classes, e.g., digits 0-9)
 - Softmax activation converts outputs to probabilities
-

So, Parameter Calculation

Each layer has trainable parameters (weights & biases).

1.`Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1))`

Parameters=(3×3×1)×32+32=320

- Weights: $(3 \times 3 \times 1) \times 32 = 288$
- Biases: 32

2. Conv2D(16, (3,3), activation='relu')

$$(3 \times 3 \times 32) \times 16 + 16 = 4624$$

Fully Connected (Dense) Layers

- Flatten layer = No parameters
- Dense(128, activation='relu') $\rightarrow (400 \times 128) + 128 = 51328$
- Dense(10, activation='softmax') $(128 \times 10) + 10 = 1290$

Total Parameters: 57,562

It helps in dealing with Translation Variance. i.e in the image it doesn't matter where object is preset on the left or on the right. It can detect that. i.e. it nullifies the object location dependency.

Disadvantage of maxpooling—problems in which object is location dependent for eg bounding box creation, then don't use pooling.

Next is saving the model for reusing values of its parameters.

Model is saved in h5 format only.

And for reading the saved model use load_model.

Pickle is used for ml models and not dl models.

In pytorch dl model gets saved with -pth extension.

<https://drive.google.com/drive/folders/1iHXXqRdFTN8YuVMJxwsFzkOcaKFh9DGX>

cat vs dog – It's a project. We will use tensorflow keras. 1st download the dataset. Do it in colab use gpu for training dataset. 1st create account in Kaggle. Go to ur profile. Here u have settings. Here u have API token. Just create new token. Once that will be done Kaggle.json will be downloaded. Upload that Kaggle.json file in google colab like normal files. After that it will automatically download that dataset.

Theory is written in file. Next u can work on horse vs human data.

In the backpropagation of cnn, filters values are adjusted for better feature extraction.

Next is OCR. Optical Character Recognition. Used to extract text from image or pdf.

For eg. u click pictures of invoice and from these images, u want to extract information. So we will use pytesseract. This is a library from ocr. It is open source.

U can use colab. We will do in colab - cpu

<https://drive.google.com/drive/folders/1iHXXqRdFTN8YuVMJxwsFzkOcaKFh9DGX>

1:17

Media Pipe is developed by google. It's a framework just like pyteserract or opencv. It is open source . It is everything related to computer vision. i.e it does OCR,object detection, classification, face detection, human pose detection, face mesh for eg applying filter on Instagram ,hands ,etc. , u can change color of hair or hair style itself, box tracking for motion tracking,etc.

We will apply few of them. So 1st MediaPipe Face Mesh.

MediaPipe Face Mesh

MediaPipe Face Mesh is a solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor. Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance critical for live experiences.

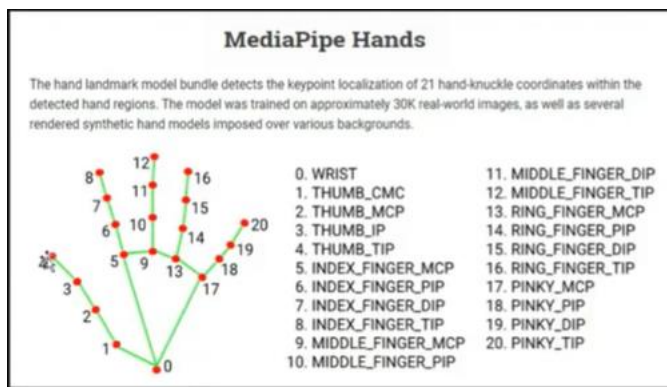


i.e. mediapipe of face marks 468 points. This is like a 3D mask. These are called as 468 landmarks.

For eg- in face mesh we r getting 468 landmarks.



Media Pipe hands has 21 hand-knuckle coordinates .



In media Pipe pose we get 33 landmarks. Then we have holistic image which finds all. Next is object Detetction. Here u need to classify the images and also the bounding box. i.e. find location also. So Finding image , classifying it and localizing it. Here we will use RCNN (Regional Proposal CNN),YOLO (You Only Look Once),SSD(Single shot multi box detector). So we will get the overview of them. RCNN has CNN for feature extraction of regions. So this 1st propose the regions containing different objects like cat car dog in an image. These regions form Region Proposal Network. These region can be found out using edge detection, Then cnn will work only on these regions to classify images inside them. Now we can do object detection by haar cascade, but this is related to human. And finds objects only related to humans.

RCNN—

RCNN (Region-based Convolutional Neural Networks) is a type of deep learning model used for **object detection**. It combines the power of **Convolutional Neural Networks (CNNs)** with **region proposals** to detect objects in images. It is trained on COCO dataset.

Let's break down how **RCNN** works and how you can implement it.

How RCNN Works

1. Region Proposals

- **Selective Search** (or other region proposal techniques) is used to **generate regions of interest (RoIs)** in an image. These are areas where objects are likely to be located.

2. CNN Feature Extraction

- Each proposed region is passed through a **CNN** to extract **features** (e.g., edge, texture, etc.).

3. Classification

- A classifier (like **SVM**) is used to classify the regions as containing an object (e.g., cat, car) or not.

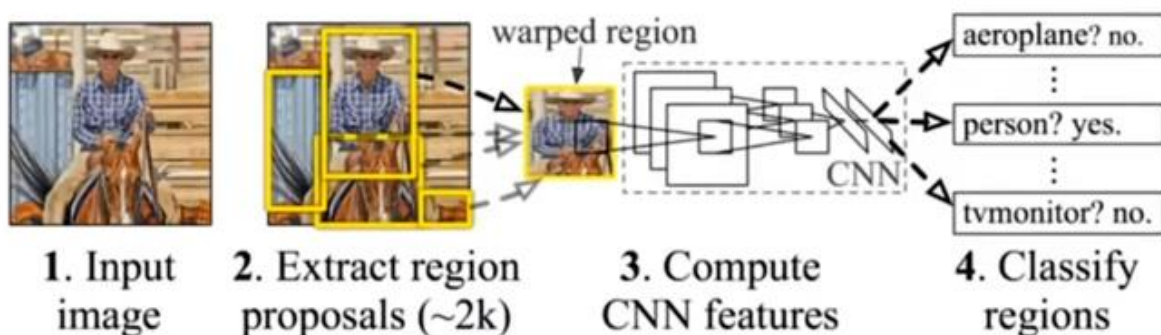
4. Bounding Box Refinement

- The **bounding boxes** around the detected objects are refined using a method like **regression** to improve the accuracy of the object localization.

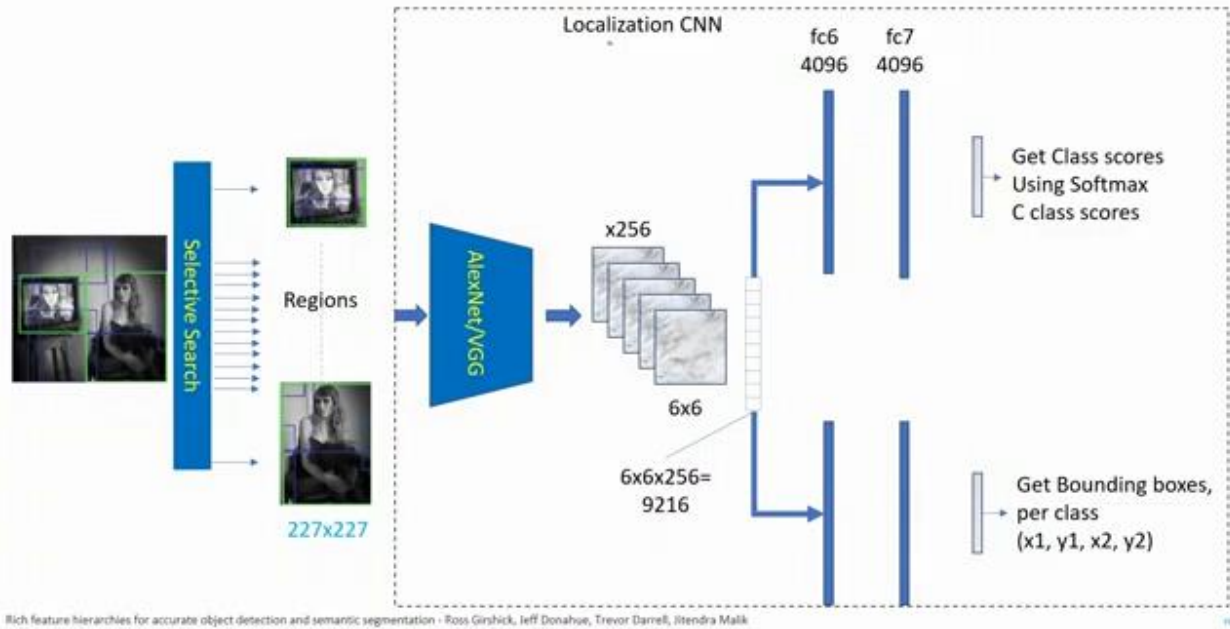
RCNN Variants

- **Fast RCNN**: Faster than original RCNN because it processes the entire image once and then applies RoI pooling for each proposed region.
- **Faster RCNN**: Further improvement with the introduction of a **Region Proposal Network (RPN)** that generates region proposals **end-to-end**, making it faster and more efficient.
- **Mask RCNN**: An extension of Faster RCNN that also outputs **object masks** (segmentation), not just bounding boxes.

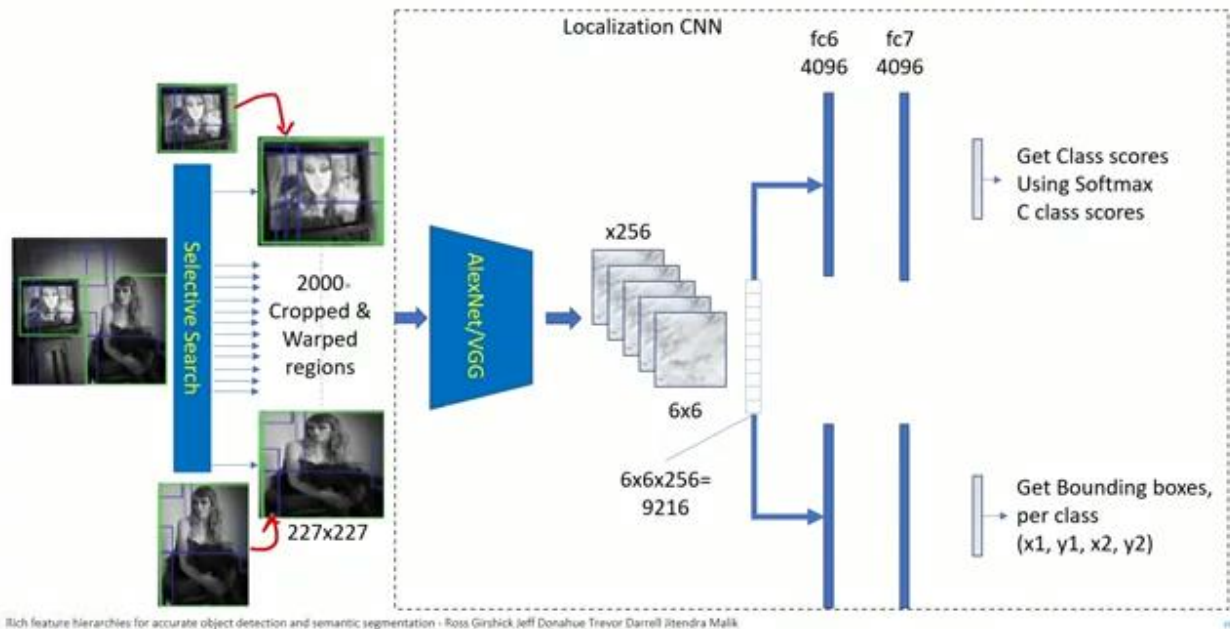
#-----



RCNN - Region proposals with CNNs



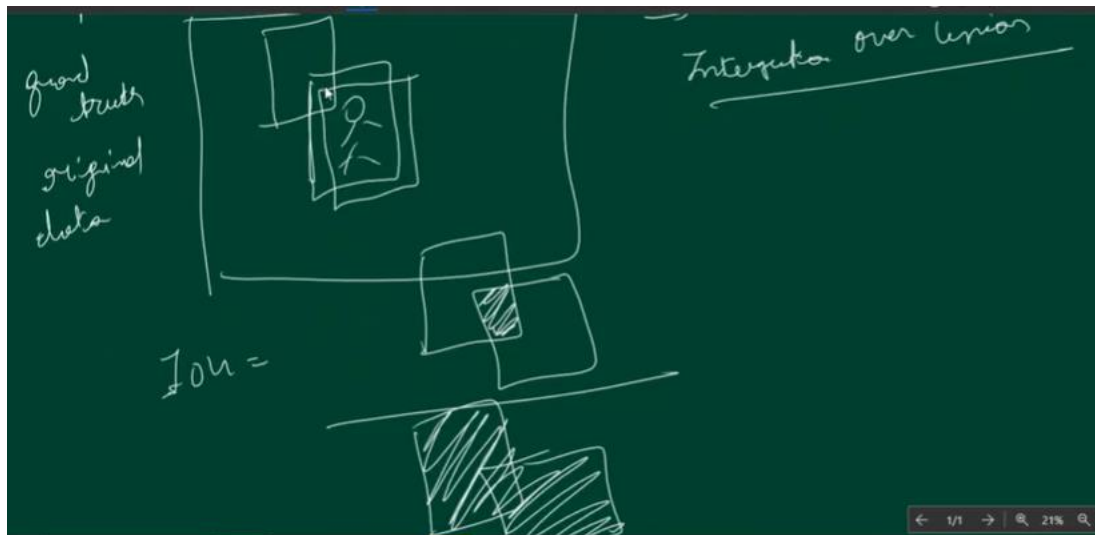
RCNN - Region proposals with CNNs



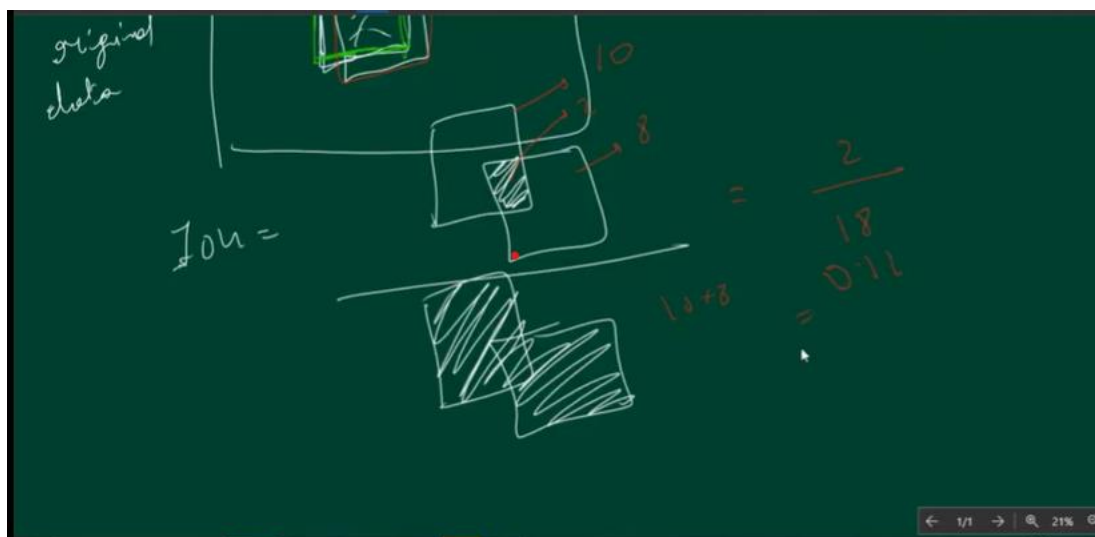
Its o/p is like [class label, bounding boxes coordinates, confidence score]

MobileNetSSD – they are mainly developed for working on mobiles. SSD is Single Shot Multibox Detector. Here 1st we apply feature extraction using CNN and then apply 6 layer convolutional layer.

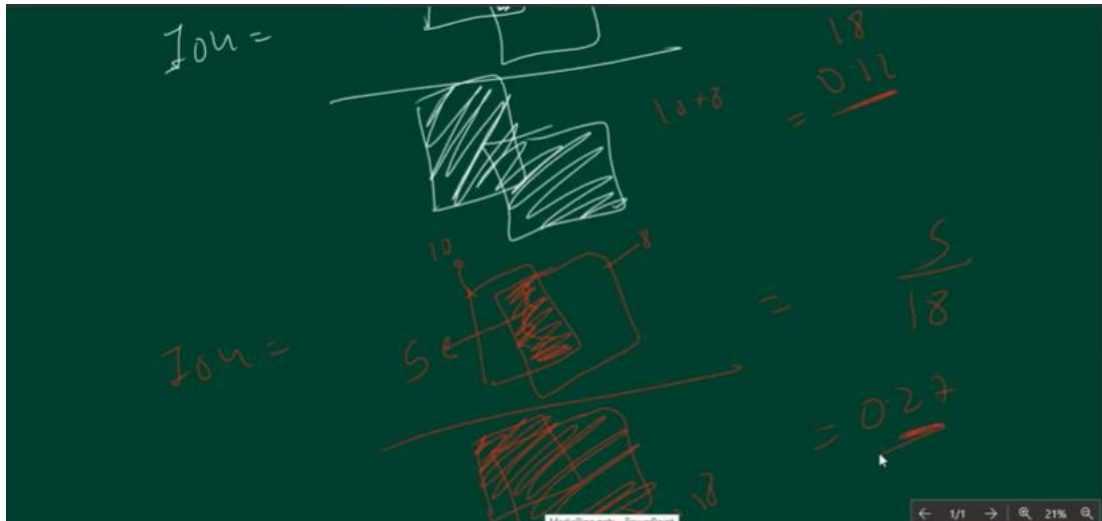
It predicts boxes around 8000 on image. And then select the best box. This is called as Non-max Suppression. So how this works?– Ground truth is basically an original image. In NMS we apply IOU i.e. Intersection Over Union.



Now u have to select that box, which has maximum overlap.



IOU in upper boxes=0.11



IOU in above image=0.27, The best box is ground truth overlapped with prediction.

So highest value of iou is 1 and lowest is 0.

So Object Detection.ipynb --- this is for pretrained YOLO Object Detection on jupyter.

ObjectDetetctionYOLOPreTrainedCustomAllInColab.ipynb – for on colab

So for custome detection on ObjectDetetctionYOLOPreTrainedCustomAllInColab.ipynb 1st perform following--

In order to label the images, we will need a link labellmg .

<https://github.com/HumanSignal/labellmg/releases>

Just scroll down and click on link windows_v1.8.0.zip . Once clicked download will begin. Extract it. Copy data and labeling from inside folder and create a new folder somewhere with lets suppose name ObjectDetection. And inside it paste our content.. So path is like--

C:\Users\hp\Documents\AI\DL_MyTry\Resources\ObjectDetection

Next within data folder we have predefined_classes text file. Here we don't have these many classes. We r going to train our model only on 2 things.

cat

dog

0-cat, 1-dog

In this folder only where we have predefined_classes.txt , paste folder containing train, valid and test data. Inside each of these folders there are 2 folders . Images and Labels. Images folder contain the images. After this go back to ObjectDetection folder and click on labellmg.exe file. Window which u r

getting will help in getting bounding boxes and also label the name. On window, its Pascal/VOC ,click that and it will change to YOLO.

Next click on the button open dir, select images folder inside train folder i.e.

C:\Users\hp\Documents\AI\DL_MyTry\Resources\ObjectDetection\data\train --select images folder. Don't open it further. So this will bring all the training images.

Next we need to upload labels. So click on change save dir . And select labels folder inside train.

C:\Users\hp\Documents\AI\DL_MyTry\Resources\ObjectDetection\data\train

So now we want to label our images. So for this , click on create rect box . Using this u can create rectangular boxes. And the label it as dog by clicking on dog on the open window. Save

Next same for cat. Save.

Now we will go to next image. So like this create rectangle boxes on the images.

So once we have labelled all the images, now if u will see labels folder inside then some labels have come inside it.

Next validation data—so open dir – select images inside valid folder. Now directory of images have changed so change labels folder too. So change save dir – and select labels folder inside valid.

And label each image.

Lly now for test image . Label all the test folder images.

Next go to folder data inside Object Detection.

C:\Users\hp\Documents\AI\DL_MyTry\Resources\ObjectDetection\data

U need to create yaml file here. So right click → new text document. Inside it write—

path: ../data

train: ../data/train/images

test: ../data/test/images

val: ../data/valid/images

nc: 2

names: ["dog","cat"]

save as—file name : data.yaml , save as type : all files

Next we will use google colab for getting the servers. ObjectDetectionYOLO.ipynb

In the colab click on google drive folder. And connect to google drive.

While this is happening, click on new tab and open google drive there and upload data folder present inside ObjectDetection folder. Now on google colab refresh drive and data folder will appear.

in the google colab u can add this link. Currently working on cpu.

