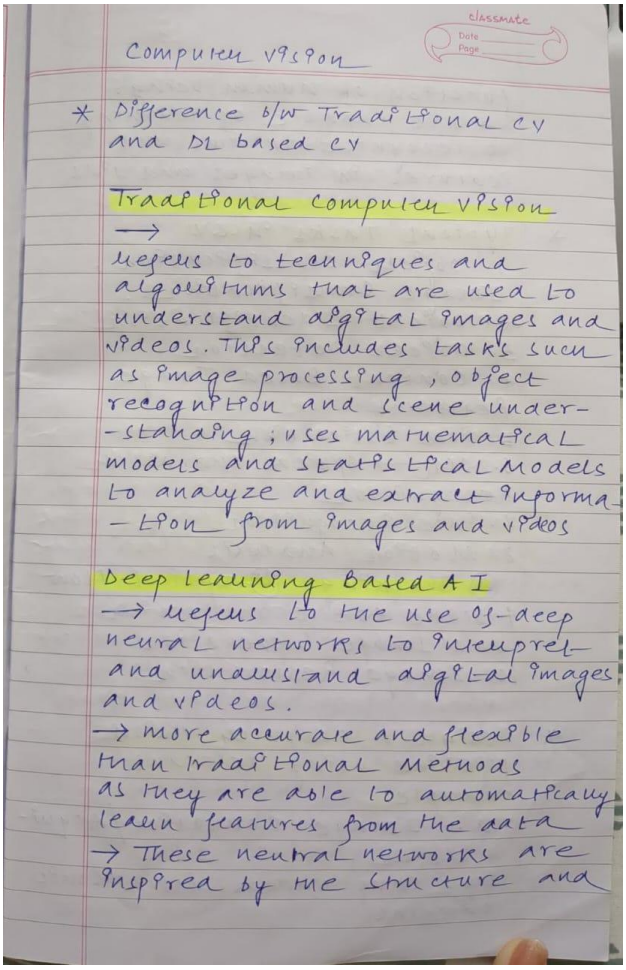
PPT 1



Computer vision

* Difference b/w Traditional CV and DL based CV

Traditional computer vision
→
Refers to techniques and algorithms that are used to understand digital images and videos. This includes tasks such as image processing, object recognition and scene understanding; uses mathematical models and statistical models to analyze and extract information from images and videos

Deep learning based AI
→ Refers to the use of deep neural networks to interpret and understand digital images and videos.
→ more accurate and flexible than traditional methods as they are able to automatically learn features from the data
→ These neural networks are inspired by the structure and

function of human being.
→ are capable of learning to recognize patterns and features in images and vids

* Typical Tasks in CV
1. Object classification, Image classification
2. Object Identification
3. Object Detection
4. Content Based Image Retrieval
5. Pose Estimation
6. OCR → optical character Recognition
7. Face Recognition
8. Motion Analysis
   (a) Tracking (b) Optical Flow
9. Scene ~~con~~ reconstruction and Image Generation
10. Image Restoration

* Design a computer vision pipeline using relevant example
a. Image Collection/Acquisition
Eg: Cams, radars, ultra sonic cameras

b. Preprocessing
→ Resampling to assure the image coordinate system; noise reduction

c. Feature Extraction
→ Lines, edges, ridges, corners

d. Detection/Segmentation/
Different Operation

e. Decision Making
→ generating report of a medical diagnosis

* Digital Image as 2D function and image representation
→ An image may be defined as a 2D function where x, y are spatial coordinates
Amplitude of f at any pair of coordinates (x, y) is called intensity or gray level of the image at that pt

In an 8 bit image each pixel occupies exactly one byte.

Each pixel has 256 possible numerical vals from 0 to 255. The color palette for an 8-bit image normally contains 256 entries, defining color 0 through color 255.

Applications of Computer Vision

1. Transportation
   - Self Driving Cars, Pedestrian Detection, Traffic Flow Analysis
2. Healthcare
   - Xray Analysis, Cancer Detection, Movement Analysis, CT and MRI
3. Manufacturing
   - Reading Text and Barcodes, Defect Inspection
4. Agriculture
   - Aerial survey and imaging, livestock health monitoring, plant disease detection

For more applications visit this link: https://www.v7labs.com/blog/computer-vision-applications

Color Space

Commonly used color spaces

1. Grayscale – 1 channel, Dimensions: (height x width)
2. RGB – 3 channels, Dimensions: (height x width x channels)
3. HSV – 3 channels, Dimensions: (height x width x channels)

HSV color space: It stores color information in a cylindrical representation of RGB color points. It attempts to depict the colors as perceived by the human eye. Hue value varies from 0-179, Saturation value varies from 0-255 and Value (brightness value) varies from 0-255. It is mostly used for color segmentation purpose

Image Operations

1. Pixel based operations
    1. Contrast Stretching
    2. Thresholding
    3. Inverting/Negative Images
    4. Erosion and Dilation
    5. Bitwise operations, etc.
2. Regions based operations
    1. Contour Detection
    2. Edge Detection
    3. Line Detection, etc.

What is Contour Detection

Contour detection is a computer vision technique that involves identifying and extracting the contours or boundaries of objects in an image. The contour of an object refers to the outline that separates it from its background or other objects in the scene.

Contour detection algorithms typically work by analyzing the changes in intensity or color in an image and identifying the points where there is a significant change. These points are then connected to form a curve that represents the contour of the object.

Contour detection has a wide range of applications in computer vision, including object recognition, image segmentation, and motion detection. It is often used as a preprocessing step in many computer vision tasks to help extract relevant information from images.

How is contour detection different from edge detection

Contour detection and edge detection are both techniques used in computer vision to identify the boundaries of objects in an image, but they differ in their approach and output.

Edge detection algorithms focus on identifying points in the image where the intensity or color changes rapidly, resulting in a sharp discontinuity in the image. The output of an edge detection algorithm is a set of pixels or points that represent the location of the edges in the image.

On the other hand, contour detection algorithms analyze the shape and curvature of the edges to extract a curve or boundary that represents the contour of the object in the image. The output of a contour detection algorithm is a curve or set of curves that represent the boundaries of objects in the image.

In summary, edge detection algorithms focus on identifying individual points that represent the sharp boundaries between objects, while contour detection algorithms extract curves that represent the boundaries of the objects themselves. Edge detection is often used as a preprocessing step for contour detection, which in turn is used in many computer vision applications, such as object recognition and tracking.

PPT 2

What is line detection

Line detection is a computer vision technique that involves identifying straight lines or linear features in an image. Linear features can include edges, ridges, and other patterns that can be represented by a straight line.

Line detection algorithms typically work by analyzing the changes in intensity or color along a series of adjacent pixels or points in the image. The algorithm searches for sets of pixels that form a straight line, and then returns the parameters of the line, such as its slope and intercept.

through color ...

* What is Edge Detection? Explain Canny Edge Detection Algorithm

Edge Detection

→ Technique used in computer vision to identify and locate sharp changes in the intensity of an image. used to find boundaries of objects in an image

→ output of the algorithm is a binary image where the edges are represented by white pixels and non-edges are represented by black pixels

→ Points where the image brightness varies sharply are called the Edges

→ Method of segmenting an image into regions of discontinuity

→ allows users to observe the features of an image for a significant change in gray level. This texture indicates the end of 1 region in the image and beginning of another.

Canny Edge Detection

Technique for detecting edges in images ; Edge detection operator.

5 steps

1. **Noise Reduction**

→

reduce the Noise in the image using Gaussian filter ; helps to smooth out any small variations in pixel intensity ; making it easier to detect edges

2. **Gradient Calculation**

→ calcs the gradient of the image; gradient is a measure of the change in intensity in the image

→ used to identify areas where intensity changes rapidly

3. Non-Maximum ~~Sup~~ Suppression
Looks at the gradient value of each pixel and set any pixels that are not a local maximum to zero ; helps to thin out the edges and reduce the number of false positives

4. Double Thresholding :
Comparing the gradient vals of each pixel to 2 threshold values ; a high threshold and a low threshold.
Any pixel with a gradient value
above high threshold → strong edges
below low threshold → ~~weak~~ no edges
b/w low and high threshold → weak edges

5. Edge Tracking
Any pixel that are connected to a strong edge and have

a gradient value above the low threshold are considered to be as edges.

This step helps to connect the edges and create a more complex picture of the edges in the images.

* **Explain Hough Transform Algorithm for Line detection**

Hough Transform algorithm is a popular method for detecting lines in an image.
The basic idea behind the algorithm is to transform the image into a different representation called the Hough space; where lines in the image correspond to points in the Hough space.

1. **Edge Detection**

→ apply an edge detection algorithm to the image to find edges. done by using a Sobel, Canny or other edge detection filter.

2. Image Space to Hough Space →
Transform the image space into Hough space
Each pt in the image corresponds to a sinusoidal curve; where curve's parameters are represented by the slope and y-intercept of the line.

~~3. Voting →~~

Steps:
① Edge Detection: detect the edges in the image using edge detection algorithm like canny Edge or Sobel Edge Detection

② Conversion to polar coordinates:
convert the image from cartesian (x, y) to polar (r, θ) coordinate system; replace each pixel in the image with its corresponding polar coordinates.

(3) Initialization of the accumulator array →
Create a 2D accumulator array (Hough space or parameter space)

(4) Voting Process →
Iterate over all the edge points in the image and for each pt; vote for the parameters of the line that passes through pt. done by incrementing the accumulator array at the coordinates (r, θ) that correspond to the line passing through current pt

(5) Find the line with most votes.

(6) Final step is to draw the line on the original image using the parameters of the detected line.

* Hough Line Transform Theory

popular technique to detect any shape ; if you can represent that shape in a mathematical form . It can detect the shape even if it is broken or distorted a little bit

(edge pixels)

We find the edges, using any Edge Detection Method

↓

Now we want the geometrical representation of the Edges

We can use Hough Transform to represent the edges or pixels in mathematical / geometrical form

→ After we find out the edges using Edge Detector; we have the seq of pixels so we can loop through all the pixels and we can figure out the slopes or whatever ;

BUT DIFF TASK

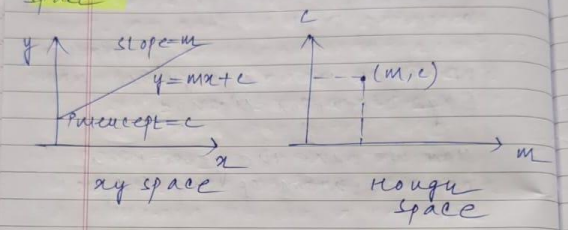
NEED A MECHANISM

→ A line in the image space can be expressed with 2 variables.

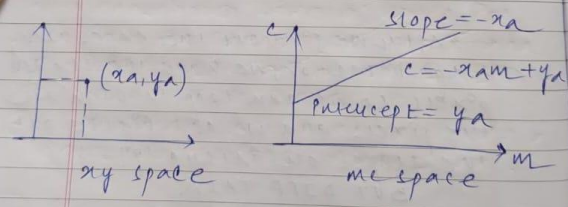
→ Cartesian coordinate system $y_i = mx_i + c$

→ polar coordinate system $x\cos\theta + y\sin\theta = r$

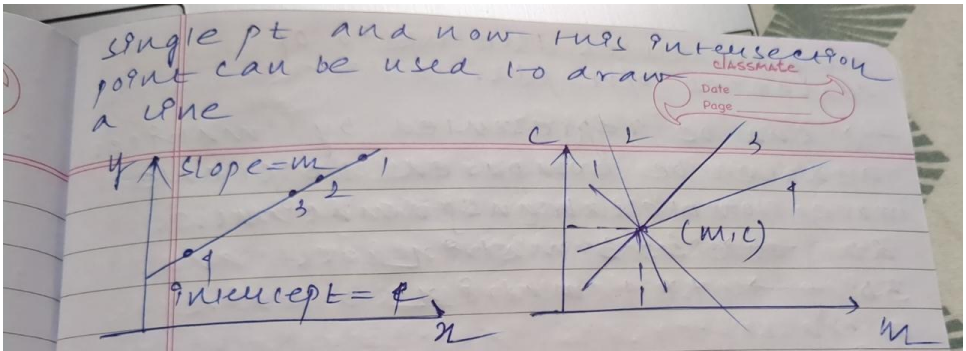
Representation of lines in Hough space



→ Representing a line in the point of a point in mc space and vice versa



In mc space; you can represent each and every point as a line and they will intersect on a single pt and now this intersection point can be used to draw a line



$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{r}{\sin\theta}$$

PPT 3

Gradient Filters

Sobel

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

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

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Prewitt

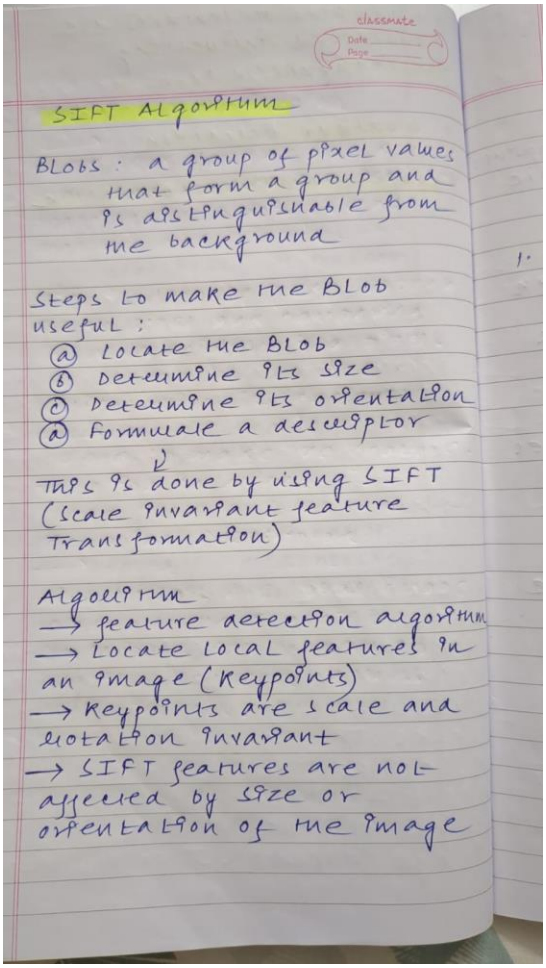
1	0	-1
1	0	-1
1	0	-1

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

Roberts

0	1
-1	0

1	0
0	-1



SIFT Algorithm

Blobs : a group of pixel values that form a group and is distinguishable from the background

Steps to make the Blob useful :
(a) Locate the Blob
(b) Determine its size
(c) Determine its orientation
(d) Formulate a descriptor

This is done by using SIFT (Scale invariant feature Transformation)

Algorithm
→ feature detection algorithm
→ Locate local features in an image (Keypoints)
→ Keypoints are scale and rotation invariant
→ SIFT features are not affected by size or orientation of the image

Steps:

1. Construct a scale space
2. Keypoint Localisation
3. Orientation Assignment
4. Keypoint Descriptor

Constructing a ~~G~~ Scale Space

→ Identify the most distinct features and that are not scale dependent ; ignore any noise present.

→ convert image to greyscale

→ apply gaussian filter to remove the noise

→ As we need to find the features that are scale independent ; the features are located on various scale creating a scale space.

→ for new sets of image ; we take the original image and reduce its scale by half and create diff blur versions of it.

→ we take DOG (Diff of Gaussian) substracting an image from its prev image to enhance its features

2. Keypoint Localisation
→
We look for the imp keypoints that can be used for feature mapping by locating local maxima and minima.

This is done by going through every pixel and comparing it with neighbouring pixels

3. Orientation Assignment
→
done to make the features rotation invariant

4. Keypoint Description
→
Descriptors are local image around the keypoint.
Descriptors can be used to match keypoints across diff images.

Region of Interest (ROI)

→ portion of the image that is considered relevant for a specific task

→ speeds up processing time and computational capacity

→ can be defined using various methods, such as drawing a rectangle around the area of interest or using image segmentation techniques

* Affine Transformation

→ Combine all linear transformations into a single transformation matrix known as affine Transformation Matrix.

→ It has 6 degree of freedom

→ Type of Geometric Transformation that preserves collinearity and ratios of distances b/w pts

→ // space in original space will remain // in transformed space

→ Affine Transformations do not preserve lengths and angles; it

only preserves the ratio of distances b/w pts.

→ can be represented by matrices and can be compared to create more complex transformations.

2D → 3×3 matrix

3D → 4×4 matrix

① Translation

moves an obj by a certain amt in the x and y direction without changing its shape or size

2×3 or 3×4 matrix

② Rotation

Rotates an object around a certain pt by a certain angle

2×2 or 3×3

③ Scaling

changes the size of an object by a certain factor in the x and y direction 2×2 or 3×3

④ Shearing

skews an object along 1 axis

⑤ Reflection

Projective Transformation

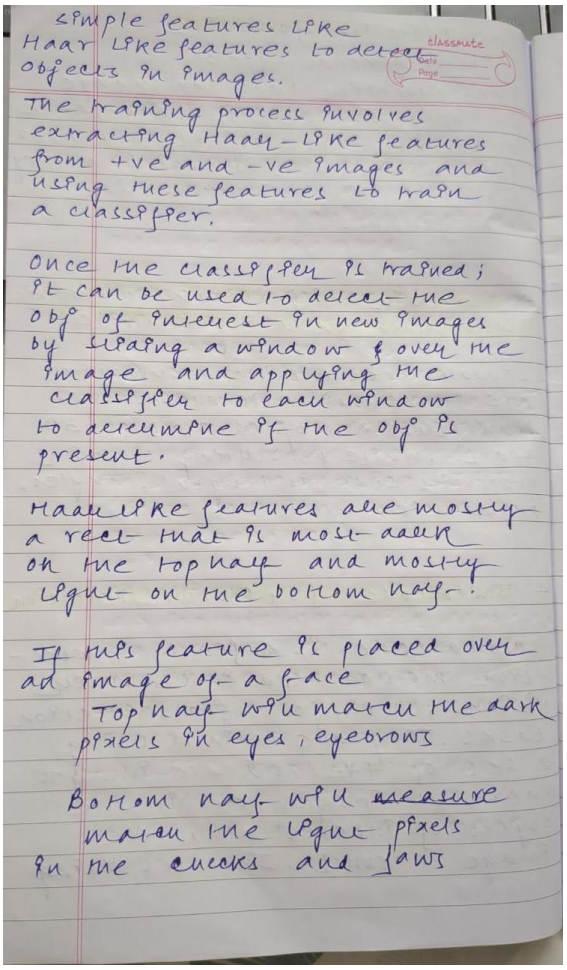
→ more general type of transformation that can change not only the position, orientation and scale of an object but also its shape; this is done by mapping lines to lines, rather than pts to pts.

* Explain diff Linear Geometric transformation using examples

Linear Geometric transformations are mathematical operations that transform an obj in a 2D or 3D space, while preserving certain properties such as length, angles, parallelism.

* Haarcascade for feature detection

Haarcascade is a feature detection method used in computer vision; used for object detection tasks. They are trained using a set of +ve and -ve images. where +ve images contain the object of interest and -ve image doesn't. Involves using



simple features like Haar like features to detect objects in images.

The training process involves extracting Haar-like features from +ve and -ve images and using these features to train a classifier.

Once the classifier is trained; it can be used to detect the obj of interest in new images by sliding a window over the image and applying the classifier to each window to determine if the obj is present.

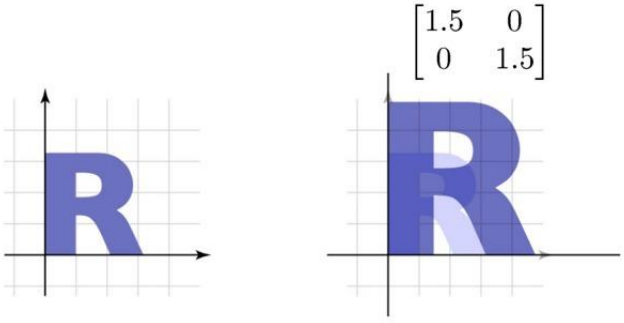
Haar like features are mostly a rect that is most dark on the top half and mostly light on the bottom half.

If this feature is placed over ad image of a face
Top half will match the dark pixels in eyes, eyebrows

Bottom half will measure match the light pixels in the cheeks and jaws

Scaling

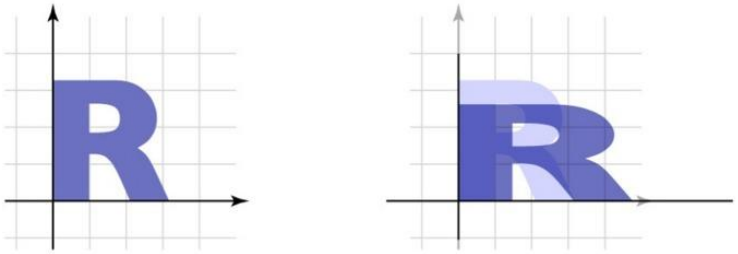
- **Uniform scale** $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$



Scaling

- **Nonuniform scale** $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$

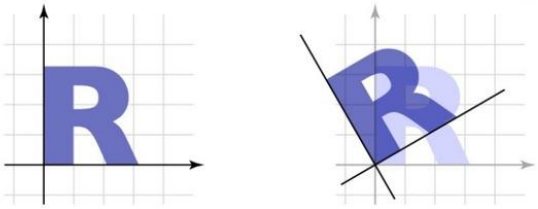
$$\begin{bmatrix} 1.5 & 0 \\ 0 & 0.8 \end{bmatrix}$$



Rotation

- **Rotation** $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$

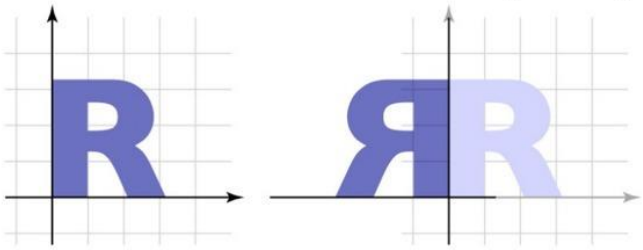
$$\begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix}$$



Reflection

- **Reflection**
  - can consider it a special case of nonuniform scale

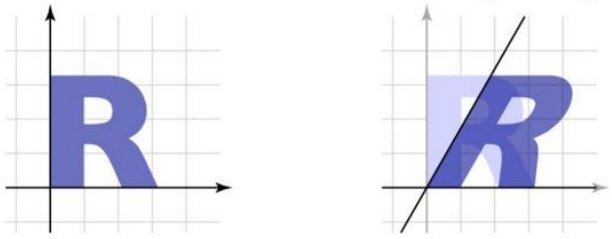
$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



Shearing

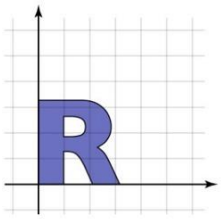
- **Shear** $\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + ay \\ y \end{bmatrix}$

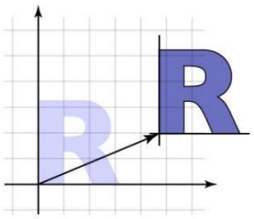
$$\begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$$



Translation

Can we represent the transformation matrix of translation using a 2x2 matrix ? **Ans: NO**





$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{matrix} X \\ Y \\ 1 \end{matrix}$$

X*1 + Y*0 + tx = translated X
X*0 + Y*1 + ty = translated Y
X*0 + Y*0 + 1 = 1

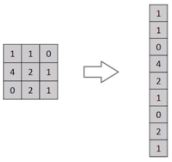
PPT 4-5

How to perform image classification using ANNs ?

1. Flatten the image
2. Pass the flattened image to the neural network

Problems with this for image related tasks:-

1. Images have a 2D spatial structure, therefore considering only the flattened pixel values are not very useful features for tasks like classification
   - Pixels that spatially separated are treated the same way as pixels that are adjacent
2. No way for network to learn features at different places in an input image
3. Imagine flattening an image of resolution 224x224x3, every image will be represented using a vector of size 150,528.
4. Now imagine using a neural network which has 2 hidden layers of 512 and 128 neurons each and one output layer of 1 neuron. Can you calculate the number of parameters required ?



Params
1. 512 x 150,528 + 512 x 1 = 77,070,848
2. 128 x 512 + 128 x 1 = 65,664
3. 1 x 128 + 1 x 1 = 129

Total = 77,136,641

Flattening the image results in loss of features of the image

1. Batch Gradient Descent: In batch gradient descent, the algorithm computes the gradient of the loss function with respect to the weights using the entire training set. The weights are then updated once for each epoch, which is a pass through the entire training set. Batch gradient descent can be computationally expensive for large datasets, but it generally provides a more accurate estimate of the gradient and converges to a better minimum.
2. Stochastic Gradient Descent: In stochastic gradient descent, the algorithm computes the gradient of the loss function with respect to the weights using a single training example at a time. The weights are then updated after each example, resulting in frequent updates to the weights. Stochastic gradient descent can be faster than batch gradient descent and can escape from local minima, but it can also be noisy and less accurate than batch gradient descent.
3. Mini-Batch Gradient Descent: Mini-batch gradient descent is a compromise between batch and stochastic gradient descent. It computes the gradient of the loss function with respect to the weights using a small batch of training examples (usually 32, 64, or 128) at a time. The weights are then updated once per batch. Mini-batch gradient descent is computationally efficient, provides a balance between accuracy and speed, and is the most commonly used optimization algorithm for training neural networks.

CNN

Kernels/Filters were used to detect features in an image by convolving them on the image.

How to decide weights of kernels?

Make the kernels learnable

Treat the kernels as trainable weights

What about nonlinear patterns in images?

Add activation functions to the kernels

What is CNN

CNN stands for Convolutional Neural Network, which is a type of deep learning neural network commonly used in computer vision tasks such as image recognition, object detection, and video analysis.

The key feature of a CNN is the use of convolutional layers that apply filters to the input data to extract meaningful features. These filters can detect edges, textures, and patterns in the image, which are then combined to identify higher-level features like objects or scenes.

CNNs also use pooling layers to downsample the feature maps and reduce the size of the data, which helps to reduce overfitting and improve computational efficiency. Finally, fully connected layers are used to make predictions based on the extracted features.

CNNs have achieved state-of-the-art performance on a wide range of computer vision tasks and have been widely adopted in various fields, including healthcare, self-driving cars, and robotics.

Filter/Kernel depth should be the same as input image depth

Pooling Layers:

Downsample the input data: Pooling layers reduce the spatial dimensions (height and width) of the input data, making it more manageable for subsequent layers of the network. This can help to reduce the number of parameters in the model and prevent overfitting.

Max pooling: The max pooling operation takes the maximum value of a small window of the input data, discarding the rest of the values. This has the effect of preserving the most salient features of the input data while discarding irrelevant information.
Average pooling: The average pooling operation takes the average value of a small window of the input data, which can help to reduce the impact of noisy or irrelevant features in the input data.

Input Dimensions: Width (W1) x Height (H1) x Depth (D1) [e.g. 224x224x3]

Spatial Extent of Filter/Kernel: F (depth of the filter is same as of input)

Output Dimensions: W2 x H2 x D2

Strides: S

Kernels: K

Padding: P

Let's just consider the W1 and H1 for now. Suppose we have an image of size 5x5 (W1xH1) and kernel of size 3x3.

The result of convolution is 3x3

W2: W1 – F + 1 -> 5 – 3 + 1 = 3

H2: H1 – F + 1 -> 5 – 3 + 1 = 3

What if we want the output of convolution to be of same size as input ?

W2: W1 – F + 2P +1, H2: H1 – F + 2P +1

5 – 3 + 2x1 + 1 = 5

Padding:

Padding is a technique used in Convolutional Neural Networks (CNNs) to add additional border pixels around the input data before applying convolutional filters. The border pixels are usually filled with zeros or a constant value, hence the name "padding."

Strides:

Strides are a parameter used in Convolutional Neural Networks (CNNs) to control the movement of the convolutional filter across the input data.

Defines the intervals at which the kernel is applied to the image

Results in an output of smaller dimensions

W2: [(W1 – F + 2P)/S] + 1

H2: [(H1 – F + 2P)/S] + 1

6x6 image, 2x2 kernel and stride 2

[(6 – 2 + 2x0)/2] + 1 = 3

What were the challenges faced by ANNs aka FNNs in image tasks?

Problems with this for image related tasks:

- Images have a 2D spatial structure, therefore considering only the flattened pixel values are not very useful features for tasks like classification.
- Pixels that are spatially separated are treated the same way as pixels that are adjacent.
- No obvious way for network to learn features at different places in an input image (similar edges, corners or blobs).
- Can get computationally expensive for large images.

How do CNNs solve these challenges?

Local Receptive Fields:

- Captures local spatial relationships in pixels
- Reduces the number of parameters significantly

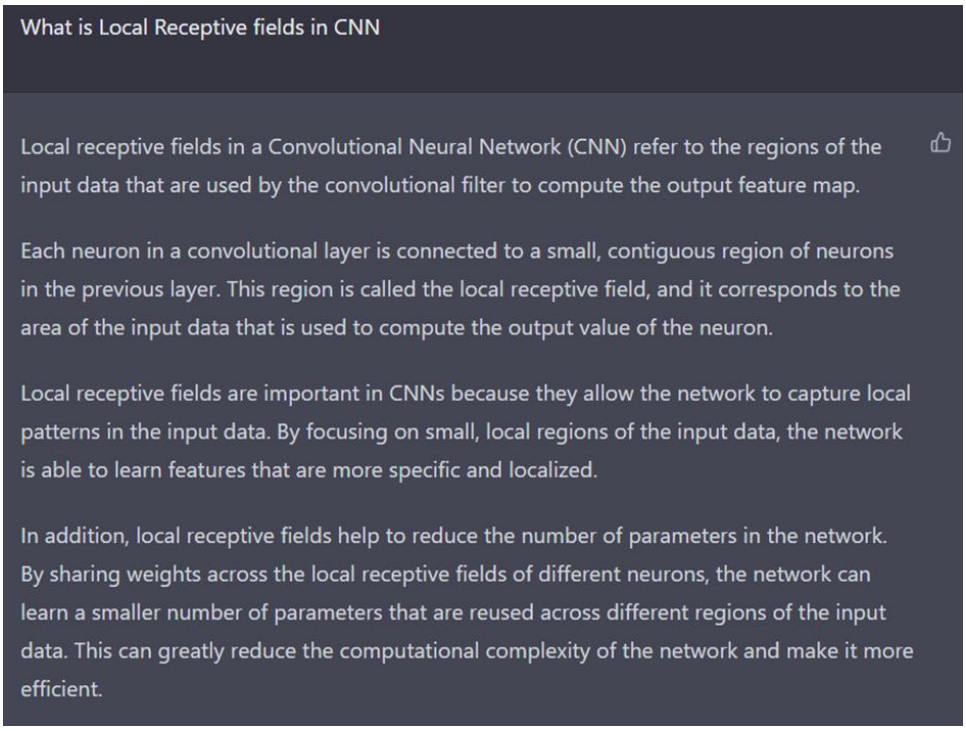
Weight Sharing:

- Enables rotation, scale and translational invariance to objects in images
- Reduces number of parameters in the model

Pooling: condenses info from previous layer

- Aggregates info, especially the minor variations
- Reduces size of output from previous layer, which reduces the number of computations in the previous layer

Local Receptive Fields



Weight Sharing

What is weight sharing in CNN

Weight sharing is a technique used in convolutional neural networks (CNNs) to reduce the number of parameters needed for a given task. In a CNN, the convolutional layers consist of a set of filters, each of which slides over the input image to produce a feature map. Each filter has a set of learnable parameters (weights), which are updated during the training process.

In weight sharing, the same set of filter weights is used to process different parts of the input image. For example, in a typical image classification task, the same set of filters is used to extract features from different regions of the image. This reduces the number of learnable parameters in the network, since each filter is not optimized separately for every position in the input.

CNN Architecture

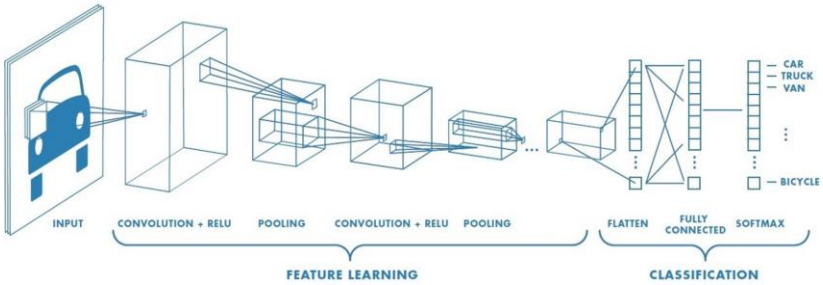
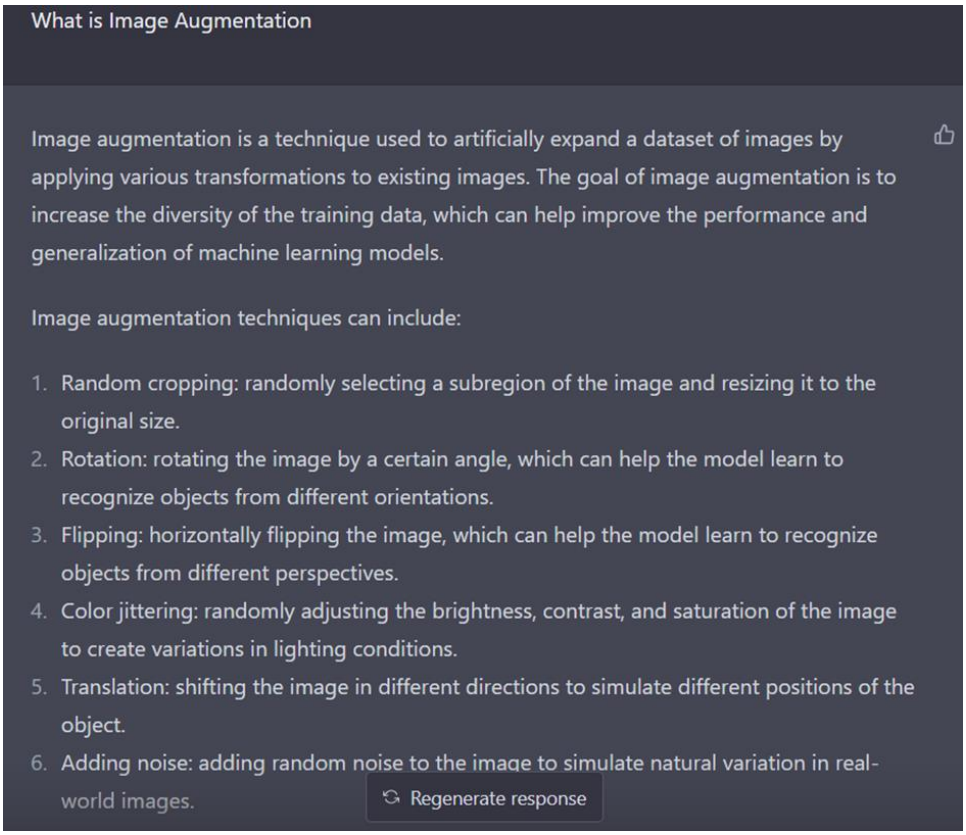
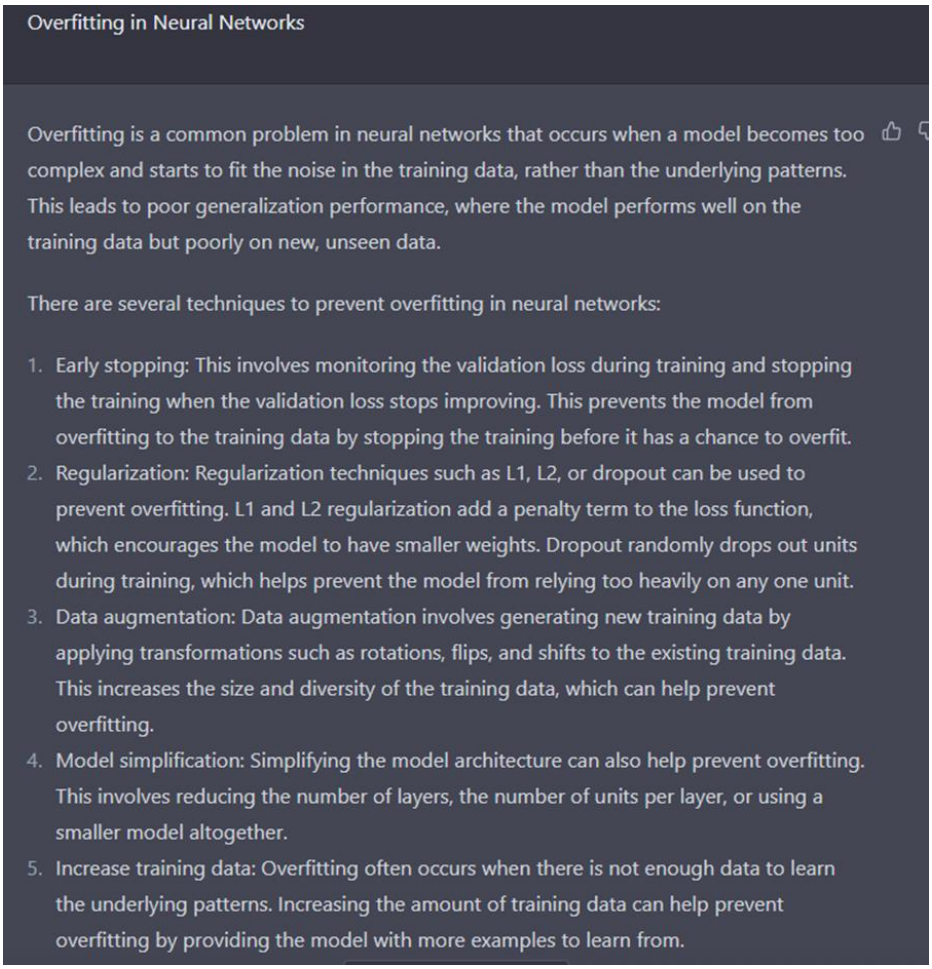


Image Augmentation



Overfitting in NN



Dropout

The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p. E.g., if a hidden layer has 8 neurons and the dropout rate p is 0.5, then 4 neurons will be randomly dropped off from that hidden layer.

Another way to think of it is to realize that a unique neural net is generated at each training step. The final neural network during the inference time will be nothing but the average ensemble of these smaller reduced neural nets. It has the same intuition as Random Forest.

Dropout is a regularization technique used in neural networks to prevent overfitting by randomly dropping out some neurons during training. It was introduced by Hinton et al. in 2012.

During each training iteration, a fraction of the neurons in the network are randomly selected and their outputs are set to zero. This means that their connections to other neurons in the next layer are also temporarily removed. The fraction of neurons to be dropped out is a hyperparameter that is typically set between 20% and 50%.

The idea behind dropout is that it forces the network to learn more robust and distributed features, as different subsets of neurons are activated during each training iteration. It also prevents the network from relying too heavily on any one feature, which can lead to overfitting.

At test time, all neurons are active, but their outputs are scaled down by the dropout probability. This ensures that the expected output of each neuron remains the same as during training and helps the network generalize well to new examples.

Dropout solves the problem of overfitting which is prominently caused by the problem of co-adaptation.

In neural network, co-adaptation means that some neurons are highly dependent on others. If those independent neurons receive “bad” inputs, then the dependent neurons can be affected as well, and ultimately it can significantly alter the model performance, which is what might happen with overfitting.

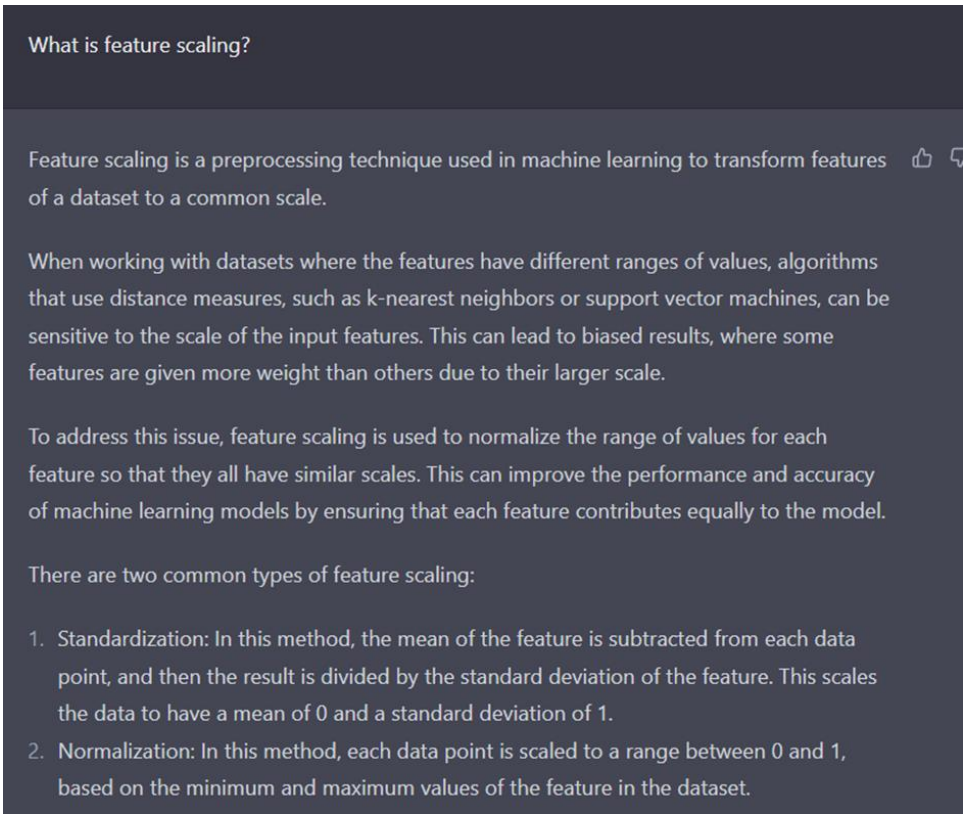
Co-adaptation, also known as co-adaptive overfitting, is a problem that can occur in neural networks when multiple neurons learn to work together to overfit the training data. In other words, co-adaptation occurs when different subsets of neurons become too specialized in memorizing specific aspects of the training data, rather than learning generalizable features that can be applied to new data.

Co-adaptation can lead to poor generalization performance of the neural network, as the network becomes too focused on the specific training data and is unable to generalize well to new data. This problem can be particularly severe in deep neural networks, where there are many layers and many neurons, making it difficult to identify and correct for co-adaptation.

To address the problem of co-adaptation, several regularization techniques can be used. For example, dropout can help prevent co-adaptation by randomly dropping out subsets of neurons during training, forcing the network to learn more robust and generalizable features. Another technique is weight decay, which encourages the network to have smaller weights and can help prevent co-adaptation by reducing the complexity of the network.

During inference (testing/validation) dropout layer is not used. This means that we consider all the neurons of the original/parent neural network. But due to this the final weights of the neural network will be larger than expected. Therefore, the weights are scaled by multiplying p the dropout rate with

the weights for each unique neurons dropped out neural network and then summed together to get the final weights of the neural network at inference time.



Internal Covariate Shift

• “We define Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.” – Authors of Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

• In neural networks, the output of the first layer feeds into the second layer, the output of the second layer feeds into the third, and so on. When the parameters of a layer change, so does the distribution of inputs to subsequent layers.

• These shifts in input distributions known as Internal Covariate Shifts can be problematic for neural networks, especially deep neural networks that could have a large number of layers.

Batch Normalization

What is Batch Normalisation

Batch Normalization is a technique used in deep neural networks to improve the training and performance of the model. It works by normalizing the inputs to each layer, making the optimization process more stable and efficient.

During training, the mean and variance of the activations in each layer of the neural network are calculated over a mini-batch of data. The mean and variance are then used to normalize the activations by subtracting the mean and dividing by the standard deviation. This ensures that the input to each layer has zero mean and unit variance, which can help to stabilize the optimization process and prevent the vanishing and exploding gradient problems that can occur in deep neural networks.

Batch normalization can be applied before or after the activation function in each layer of the neural network. By normalizing the inputs to the activation function, batch normalization can help to prevent the saturation of the activation function and improve the representational power of the network.

Batch normalization has been shown to be an effective technique for improving the performance of deep neural networks, and it has become a popular technique for training deep learning models. It can help to speed up the training process, improve the accuracy of the model, and make the optimization process more stable and robust.

Global Average Pooling

The GAP operation is a form of spatial pooling that aggregates the information from the entire feature map into a compact and interpretable representation. It is computationally efficient and reduces the number of parameters in the network, which helps to prevent overfitting. Additionally, the GAP layer produces features that are more robust to spatial translations and can generalize better to new images that the network has not seen before.

To summarize, Global Average Pooling is a powerful technique that can help to improve the efficiency and performance of CNNs for image classification tasks. By reducing the spatial dimensions of the feature map and producing interpretable features, the GAP layer can help to prevent overfitting, reduce computational complexity, and improve the generalization of the network.

Reduces the increase in number of params caused by flattening layer

Takes an average of all the feature maps of the layer

Transfer Learning

Transfer Learning is a technique in deep learning where a pre-trained model is used as the starting point for a new task instead of training a new model from scratch. The pre-trained model is typically trained on a large dataset and has already learned useful features that can be applied to the new task, which helps to reduce the amount of data and time needed for training a new model. Here is a more detailed explanation of Transfer Learning:

1. Pre-trained model: The pre-trained model is a neural network that has been trained on a large dataset, such as ImageNet. The network has learned to recognize and extract useful features from the images in the dataset, which makes it a good starting point for a new task.
2. Feature Extraction: In Transfer Learning, the pre-trained model is used as a feature extractor by freezing its weights and using the output of one of its layers as input to a new model. The output of the pre-trained model's layer is a set of features that have been learned from the original dataset.
3. New Model: A new model is built on top of the pre-trained model's output by adding one or more layers that are specific to the new task. These layers are randomly initialized and trained on a smaller dataset that is specific to the new task.
4. Fine-tuning: Fine-tuning is an optional step in Transfer Learning where the weights of some or all of the layers in the pre-trained model are allowed to be updated during training on the new task. This can help to adapt the pre-trained features to the new task and improve performance.

Transfer Learning Approaches

1. Using a pretrained network as a classifier
2. Using a pretrained network as a feature extractor
3. Finetuning

AlexNet

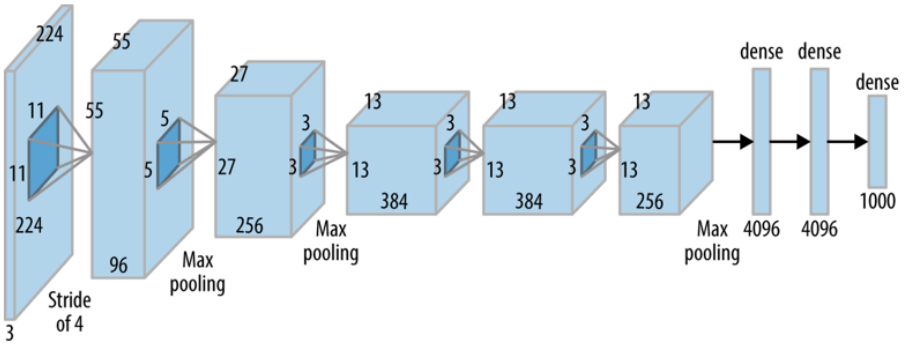
Explain Alex Net architecture in details

AlexNet is a convolutional neural network architecture that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. The architecture consists of 5 convolutional layers, 3 max-pooling layers, and 3 fully connected layers.

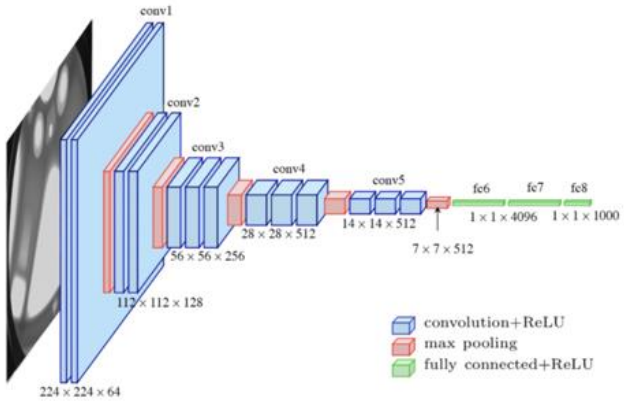
Here is a detailed description of the AlexNet architecture:

1. Input Layer: The input to the network is an RGB image of size 224x224x3.
2. Convolutional Layer 1: The first convolutional layer consists of 96 filters of size 11x11x3 with a stride of 4. The output of this layer is 55x55x96.
3. Max-Pooling Layer 1: The first max-pooling layer follows the first convolutional layer and reduces the spatial dimensions of the output by a factor of 2. It uses a 3x3 window with a stride of 2. The output of this layer is 27x27x96.
4. Convolutional Layer 2: The second convolutional layer consists of 256 filters of size 5x5x48. The output of this layer is 27x27x256.
5. Max-Pooling Layer 2: The second max-pooling layer follows the second convolutional layer and reduces the spatial dimensions of the output by a factor of 2. It uses a 3x3 window with a stride of 2. The output of this layer is 13x13x256.
6. Convolutional Layer 3: The third convolutional layer consists of 384 filters of size 3x3x256. The output of this layer is 13x13x384.
7. Convolutional Layer 4: The fourth convolutional layer consists of 384 filters of size 3x3x192. The output of this layer is 13x13x384.
8. Convolutional Layer 5: The fifth convolutional layer consists of 256 filters of size 3x3x192. The output of this layer is 13x13x256.
9. Max-Pooling Layer 3: The third max-pooling layer follows the fifth convolutional layer and reduces the spatial dimensions of the output by a factor of 2. It uses a 3x3 window with a stride of 2. The output of this layer is 6x6x256.
10. Flatten Layer: The output of the last max-pooling layer is flattened into a 4096-dimensional vector.
11. Fully Connected Layer 1: The first fully connected layer consists of 4096 neurons with ReLU activation.
12. Fully Connected Layer 2: The second fully connected layer consists of 4096 neurons with ReLU activation.
13. Fully Connected Layer 3: The output layer consists of 1000 neurons corresponding to the 1000 classes in the ImageNet dataset. It uses softmax activation to produce the probability distribution over the classes.

AlexNet



VGG16



The VGG16 architecture consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are grouped into 5 blocks, with each block consisting of multiple convolutional layers followed by a max pooling layer. The fully connected layers at the end of the network are used to produce the final output predictions.

Input:
The input to the network is a 224×224 RGB image. The RGB channels are mean centered and normalized.

Convolutional Layers:
The first two blocks of the network have two convolutional layers each, with 64 filters in each layer. The filters are 3×3 with a stride of 1 and padding of 1, which preserves the spatial dimensions of the input. The activation function used is Rectified Linear Units (ReLU). The third convolutional layer in each block has 128 filters, and the fourth convolutional layer has 256 filters. The fifth block has 3 convolutional layers with 512 filters. All of these layers have a filter size of 3×3 with a stride of 1 and padding of 1.

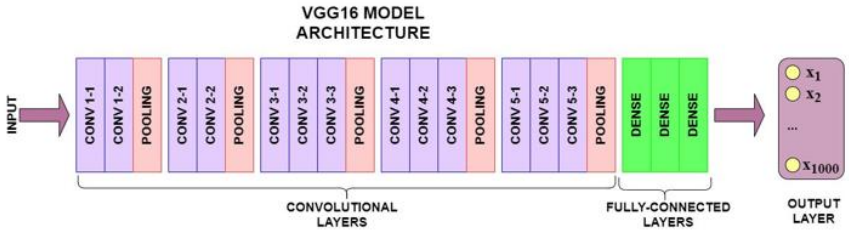
Max Pooling Layers:
After each block of convolutional layers, a max pooling layer is used to downsample the spatial dimensions of the output. The max pooling layers have a filter size of 2×2 with a stride of 2, which reduces the output spatial dimensions by half.

Fully Connected Layers:
The final three layers of the network are fully connected layers. The first two fully connected layers have 4096 units each, and the last fully connected layer has 1000 units, corresponding to the number of classes in the ImageNet dataset. The ReLU activation function is used for the first two fully connected layers, and a Softmax activation function is used for the last layer to produce the class probabilities.

Pre-Trained Network

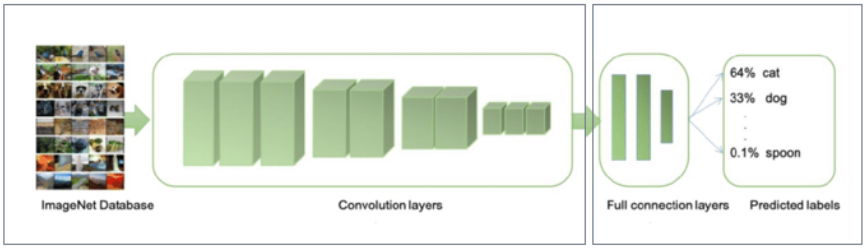
Pretrained Network as a Classifier

Produces the class output from the data labels it has been trained on



Pretrained Network as a Feature Extractor

1. Use the convolutional layers of the trained model to extract features
2. Remove the classifier of the pretrained network and add your own classifier



Freeze the layers and use pretrained weights

Add a classifier on top, fully connected layers + decision neuron

Pretrained Network as a Feature Extractor

1. Use the image embeddings extracted from the pretrained network and build a tabular dataset
2. Use the target labels and these extracted features to train the classifier model like you usually do

```
In [76]: feature_extractor = Model(model.inputs,model.layers[-2].output)

In [77]: features = feature_extractor.predict(validation_generator)

In [90]: features.shape

Out[90]: (2000, 512)
```

Fine Tuning

Fine-tuning involves the following steps:

1. Select a Pre-trained Model: The first step in fine-tuning is to select a pre-trained model that is relevant to the new task. The pre-trained model must be trained on a large, general dataset and should have similar features to the new task. For example, a pre-trained model that is trained on the ImageNet dataset can be fine-tuned for a specific image classification task.
2. Modify the Model Architecture: The next step is to modify the architecture of the pre-trained model to fit the requirements of the new task. This involves removing the output layer of the pre-trained model and replacing it with a new layer that is specific to the new task. The new layer can be initialized randomly or with a pre-defined weight distribution.
3. Freeze some Layers: Once the new layer is added, some of the layers in the pre-trained model can be frozen to prevent their weights from being updated during training. The number of layers that are frozen can be determined by the size of the new dataset and the similarity of the new task to the original task that the model was trained on. Generally, the earlier layers in the model are frozen, as they are more general and have learned more generic features.
4. Train the Model: After the pre-trained model has been modified and some of the layers have been frozen, the model is trained on the new dataset. During training, the weights of the new layer and some of the unfrozen layers are updated using backpropagation, while the weights of the frozen layers remain constant.
5. Evaluate and Fine-tune Again: Once the model is trained, it is evaluated on a validation set, and the performance is analyzed. Depending on the performance, the model can be fine-tuned again by adjusting th Regenerate response ber of frozen layers, or the number of epochs.

So how many layers should we freeze in fine tuning?

The number of layers to freeze during fine-tuning depends on the size and complexity of the new dataset, as well as the pre-trained model used as the starting point. In general, it is recommended to freeze the early layers of the pre-trained model (i.e., the convolutional layers) and fine-tune the later layers (i.e., the fully connected layers) that are more specific to the task at hand.

For example, in the VGG16 architecture, it is common to freeze the first 10-12 convolutional layers and fine-tune the remaining layers. This is because the earlier layers learn more general and low-level features such as edges and textures, while the later layers learn more specific and high-level features such as object shapes and parts. However, it is important to experiment with different numbers of frozen layers and fine-tuning strategies to find the optimal balance between generalization and overfitting on the new dataset.

Choosing the appropriate level of Transfer Learning

Scenario	Size of the target data	Similarity of the original and new datasets	Approach
1	Small	Similar	Use pretrained network as a feature extractor
2	Large	Similar	Freeze 60-80% of the network and fine tune the rest of the network
3	Small	Very Different	Since target dataset is different it might not be the best to freeze the higher level features of the pretrained network because they contain source dataset specific features. It will be better to retrain layers from somewhere early in the network. Freeze 30-50% of the network.
4	Large	Very Different	Fine tune the entire network/Train the entire network by using the existing weights of the pretrained network

Transfer Learning and Learning Rates

Note: Learning rates should be kept small when transfer learning is applied, reason being you do not want the trainable layers' weights to change drastically during your training process.

Model Callbacks

Early Stopping:

Explain Early Stopping in Model Callbacks

Early stopping is a technique used in machine learning to prevent a model from overfitting to the training data. It is implemented as a callback function that monitors the validation loss during training and stops the training process early if the loss does not improve after a certain number of epochs.

The basic idea is to monitor the validation loss during training and stop the training process once the validation loss starts to increase, as this is a sign that the model is starting to overfit to the training data. The number of epochs to wait before stopping the training process is controlled by a hyperparameter called "patience."

The early stopping callback function is typically implemented as part of the training loop in a deep learning framework such as TensorFlow or PyTorch. It is activated at the end of each epoch, and it checks whether the validation loss has improved compared to the previous epoch. If the validation loss has not improved for a certain number of epochs (i.e., the patience value), the training process is stopped early.

Checkpointing

Explain Checkpointing in Model Callbacks

Checkpointing is a technique used in machine learning to save the state of a model during training. It is implemented as a callback function that saves the weights of the model at regular intervals during training, so that the model can be resumed from where it left off in case the training process is interrupted for some reason.

The basic idea behind checkpointing is to save the weights of the model to disk after a certain number of epochs, so that we can restore the model later if needed. The frequency at which the weights are saved is controlled by a hyperparameter called "save frequency."

The checkpointing callback function is typically implemented as part of the training loop in a deep learning framework such as TensorFlow or PyTorch. It is activated at the end of each epoch, and it checks whether it's time to save the weights of the model. If it is, the weights are saved to disk in a format that can be easily restored later.

ReduceLROnPlateau

Explain ReduceLROnPlateau in Model Callbacks

ReduceLROnPlateau is a callback function used in machine learning to reduce the learning rate of the optimizer when the validation loss stops improving. It is implemented as a callback function that monitors the validation loss during training and adjusts the learning rate of the optimizer when a certain condition is met.

The basic idea behind ReduceLROnPlateau is to reduce the learning rate of the optimizer when the validation loss stops improving. The reduction in learning rate is controlled by a hyperparameter called "factor," which determines by how much the learning rate should be reduced. The callback function also includes other parameters such as "patience," which determines how many epochs to wait before reducing the learning rate, and "min_delta," which specifies the minimum amount of improvement in the validation loss required to consider it as an improvement.

The ReduceLROnPlateau callback function is typically implemented as part of the training loop in a deep learning framework such as TensorFlow or PyTorch. It is activated at the end of each epoch, and it checks whether the validation loss has improved compared to the previous epoch. If the validation loss has not improved for a certain number of epochs (i.e., the patience value), the learning rate of the optimizer is reduced by the factor specified in the hyperparameters.

ROC and AUC Curve

ROC and AUC CURVE

ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model. It plots the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. The TPR is the proportion of positive samples that are correctly identified as positive, while the FPR is the proportion of negative samples that are incorrectly identified as positive.

AUC (Area Under the Curve) is the area under the ROC curve. It is a measure of the model's ability to distinguish between positive and negative samples. A perfect model has an AUC of 1, while a model that performs no better than random guessing has an AUC of 0.5.

The ROC curve and AUC are commonly used to evaluate the performance of binary classification models, such as those used in medical diagnosis, credit risk assessment, and fraud detection. They can help you choose the best threshold for your model, assess the trade-off between sensitivity and specificity, and compare the performance of different models.

In summary, ROC and AUC are important metrics for evaluating the performance of binary classification models, and they can help you choose the best model for your specific task.

How to evaluate a model by looking at Roc Auc Curve

The ROC (Receiver Operating Characteristic) curve and AUC (Area Under the Curve) are commonly used evaluation metrics for binary classification models. Here are the steps to evaluate a model by looking at the ROC AUC curve:

1. Plot the ROC curve: The first step is to plot the ROC curve for the model. The ROC curve is a graph that shows the performance of the model at different classification thresholds. The x-axis represents the false positive rate (FPR) and the y-axis represents the true positive rate (TPR). The ROC curve is created by plotting the TPR against the FPR at different thresholds.
2. Calculate the AUC: The next step is to calculate the AUC for the model. The AUC is a summary statistic that measures the overall performance of the classifier. It ranges from 0 to 1, with 0.5 indicating that the classifier performs no better than random, and 1 indicating perfect performance.
3. Interpret the ROC curve and AUC: The final step is to interpret the ROC curve and AUC. A good model will have an ROC curve that is close to the top left corner of the plot, which indicates high TPR and low FPR. A model that performs no better than random will have an ROC curve that is close to the diagonal line. The AUC score provides a single number that summarizes the model's performance across all possible classification thresholds. Generally, an AUC score of 0.5 indicates a random classifier, while an AUC score of 1 indicates a perfect classifier.

What is overfitting and underfitting

Overfitting and underfitting are two common problems that can occur when training a machine learning model.

Overfitting occurs when a model is too complex and starts to fit the training data too well. As a result, it performs poorly on new, unseen data. In other words, the model has "memorized" the training data and is not able to generalize well to new data.

Underfitting, on the other hand, occurs when a model is too simple and is not able to capture the underlying patterns in the data. In this case, the model performs poorly on both the training data and new, unseen data.

In both cases, the goal is to find the right balance between complexity and simplicity so that the model can generalize well to new data while also fitting the training data reasonably well. This is often referred to as the "bias-variance trade-off" in machine learning.

How can a Model be improved wrt to cross entropy

Cross-entropy is a commonly used loss function in machine learning, especially for classification problems. The goal of training a model is to minimize the cross-entropy loss function. Here are some ways to improve a model with respect to cross-entropy:

1. Adjust the learning rate: The learning rate determines the step size taken in each iteration during training. If the learning rate is too high, the model may overshoot the optimal solution and fail to converge. If the learning rate is too low, the model may take too long to converge or get stuck in a local minimum. Tuning the learning rate can help improve the model's performance.
2. Increase the number of training epochs: Training a model for more epochs can help it converge to a better solution. However, training for too many epochs can lead to overfitting.
3. Regularization: Regularization techniques such as L1, L2, and dropout can help prevent overfitting and improve the generalization performance of the model.
4. Feature engineering: Feature engineering is the process of selecting or creating input features that are relevant and informative for the model. A well-designed set of features can help the model learn better and improve its performance.
5. Model architecture: Choosing the right model architecture, such as adding more layers, increasing the number of neurons, or using a different activation function, can also improve the model's performance.
6. Batch normalization: Batch normalization is a technique that normalizes the inputs to each layer to help the model learn better and faster. It can help improve the performance of the model by reducing the internal covariate shift.

LEARNING CURVES AND RESULT

How to understand whether a Model is overfitting by looking at Learning curves (Model Loss)

Learning curves can be a useful tool to help identify whether a model is overfitting. Here are some ways to understand whether a model is overfitting by looking at the learning curves (model loss):

1. Training loss vs. validation loss: One way to identify overfitting is to compare the training loss and validation loss. If the training loss continues to decrease while the validation loss starts to increase or levels off, it's a sign that the model is starting to overfit. This is because the model is becoming too specialized to the training data and is not able to generalize well to new, unseen data.
2. Gap between training loss and validation loss: Another way to identify overfitting is to look at the gap between the training loss and validation loss. If the gap is large, it's a sign that the model is overfitting. This is because the model is fitting the training data too closely and is not able to generalize well to new, unseen data.
3. Learning curve shape: The shape of the learning curve can also provide insight into whether the model is overfitting. If the learning curve for the training loss is decreasing rapidly and the curve for the validation loss is relatively flat, it's a sign that the model is overfitting. This is because the model is memorizing the training data and is not able to generalize well to new, unseen data.

How to understand whether a Model is underfitting by looking at Learning curves (Model Loss)

Learning curves can also help identify whether a model is underfitting. Here are some ways to understand whether a model is underfitting by looking at the learning curves (model loss):

1. High training loss: One way to identify underfitting is to look at the training loss. If the training loss is high and does not decrease significantly over time, it's a sign that the model is underfitting. This is because the model is too simple and is not able to capture the underlying patterns in the data.
2. Low validation loss: Another way to identify underfitting is to look at the validation loss. If the validation loss is low and does not improve significantly over time, it's a sign that the model is underfitting. This is because the model is too simple and is not able to capture the underlying patterns in the data.
3. Learning curve shape: The shape of the learning curve can also provide insight into whether the model is underfitting. If the learning curve for the training loss is relatively flat and the curve for the validation loss is decreasing slowly, it's a sign that the model is underfitting. This is because the model is not able to capture the underlying patterns in the data.

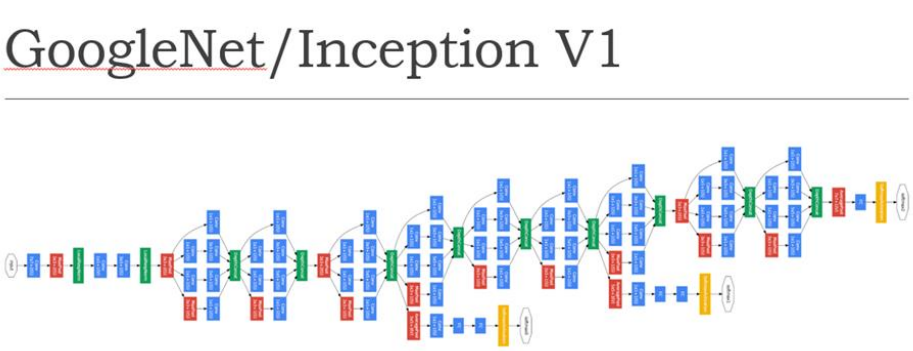
How to understand whether a Model is perfect by looking at Learning curves (Model Loss)

It is highly unlikely that a model is perfect and can perfectly fit the training data while also generalizing well to new, unseen data. However, a model can be close to perfect if it has a low training loss and a low validation loss. Here are some ways to understand whether a model is close to perfect by looking at the learning curves (model loss):

1. Low training loss: A low training loss indicates that the model is fitting the training data well.
2. Low validation loss: A low validation loss indicates that the model is able to generalize well to new, unseen data.
3. Similar training and validation loss: If the training loss and validation loss are both low and are relatively close to each other, it's a sign that the model is able to fit the training data well while also generalizing well to new, unseen data.
4. Learning curve shape: If the learning curves for both the training loss and validation loss are decreasing and are relatively flat, it's a sign that the model is close to perfect. This indicates that the model is able to learn the underlying patterns in the data without overfitting or underfitting.

PPT 6

GoogleNet/InceptionV1



Explain Inception V1 in details

Inception V1, also known as GoogleNet, is a convolutional neural network (CNN) architecture designed for image classification tasks. It was introduced in 2014 by researchers at Google, and it achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset.

The Inception V1 architecture is composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. However, what makes Inception V1 different from other CNN architectures is its use of a technique called "inception module."

An inception module is a building block that consists of multiple parallel convolutional layers with different kernel sizes and pooling layers. These parallel branches are then concatenated together to form the output of the module. The idea behind this design is to capture features at multiple scales without increasing the number of parameters.

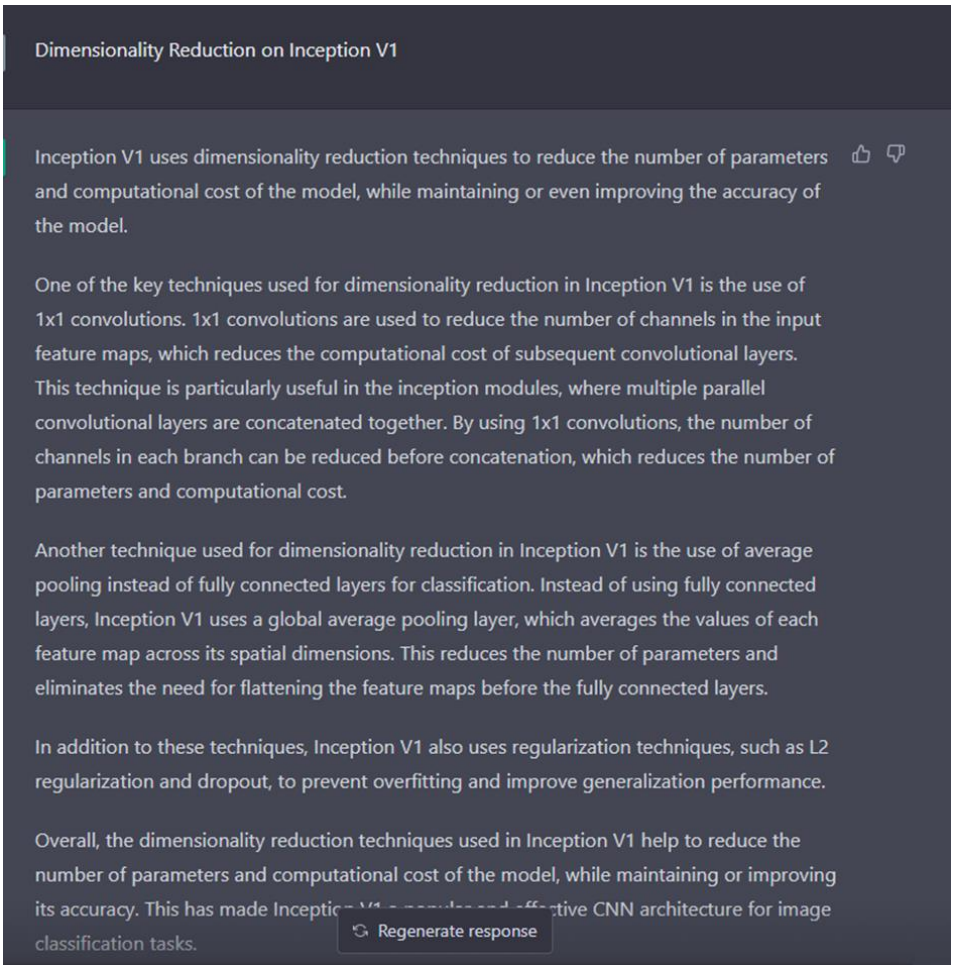
More specifically, each inception module consists of 4 branches:

1. 1x1 convolution branch: This branch performs a 1x1 convolution on the input feature maps. The purpose of this branch is to reduce the dimensionality of the input and capture low-level features.
2. 3x3 convolution branch: This branch performs a 3x3 convolution on the input feature maps. The purpose of this branch is to capture mid-level features.
3. 5x5 convolution branch: This branch performs a 5x5 convolution on the input feature maps. The purpose of this branch is to capture high-level features.
4. Max pooling branch: This branch performs max pooling on the input feature maps. The purpose of this branch is to capture features that are invariant to spatial translations.

All the outputs of these branches are concatenated along the channel dimension to form the output of the inception module. This output is then fed to the next layer in the network.

In addition to the inception modules, Inception V1 also includes other layers, such as convolutional layers with stride 2 for downsampling, pooling layers for spatial pooling, and fully connected layers for classification.

Overall, Inception V1 is a powerful CNN architecture that combines multiple parallel convolutional layers to capture features at multiple scales, while also reducing the number of parameters. It has been widely used for image classification tasks and has inspired many other CNN architectures.



GoogleNet/Inception V1 Summary

Following were the developments

1. Inception Block contains
    1. 1 × 1 convolutional layer
    2. 1 × 1 convolutional layer + 3 × 3 convolutional layer
    3. 1 × 1 convolutional layer + 5 × 5 convolutional layer
    4. 3 × 3 pooling layer + 1 × 1 convolutional layer
2. 1x1 conv layers are used for depth dimensionality reduction thus reducing the floating point operations
3. Uses Global Average Pooling instead of Flatten
4. Uses Auxillary classifiers for prediction and gradient propagation at intermediate parts of the network

Auxiliary Classifiers

What are auxillary classifiers in simple language

Auxiliary classifiers are like additional checkpoints in a convolutional neural network (CNN) that help to improve its performance in recognizing images. These classifiers are placed at certain points within the network and act like mini-classifiers that provide additional guidance and supervision to the network during training. This helps the network to learn better representations of the input data, which can improve its overall performance on image classification tasks. Think of it like having extra teachers at different stages of your learning journey, who help you to better understand the material and improve your performance on tests.

One of the key features of Inception V1 is the use of auxiliary layers, which are added to the network at various depths. These auxiliary layers are designed to help combat the problem of vanishing gradients, which can occur in very deep neural networks.

The auxiliary layers are essentially smaller, shallower versions of the full Inception network that are inserted at various stages of the network. They consist of a few convolutional layers followed by a pooling layer, and are connected to a separate loss function. During training, the outputs of these auxiliary layers are used to compute a separate loss, which is then combined with the overall loss of the network.

ResNet

Explain Resnet Architecture in details

ResNet (short for Residual Network) is a deep neural network architecture designed to address the problem of vanishing gradients in very deep neural networks. It was proposed by researchers from Microsoft Research in 2015 and has become a very popular architecture in the field of deep learning.

The basic idea behind ResNet is to use skip connections (also known as shortcut connections) to enable the flow of information from earlier layers to later layers in the network. This helps to overcome the problem of vanishing gradients, where the gradients become too small to propagate through the network during backpropagation.

The core building block of a ResNet is the residual block, which consists of two convolutional layers and a skip connection that adds the input to the output of the second convolutional layer. The skip connection allows the gradient to bypass the convolutional layers if needed, making it easier for the network to learn the identity mapping.

In addition to the residual blocks, ResNet also uses pooling layers, batch normalization, and a global average pooling layer at the end of the network to produce the final output. The pooling layers help to reduce the spatial dimensions of the feature maps, while the batch normalization layers help to normalize the activations and improve the stability of the network.

ResNet comes in several different versions, with the number of layers ranging from a few dozen to hundreds. The most popular version is ResNet-50, which has 50 layers and achieved state-of-the-art performance on several computer vision tasks, including image classification, object detection, and semantic segmentation.

Overall, ResNet is a powerful deep learning architecture that has proven to be very effective for a wide range of computer vision tasks. Its ability to handle very deep networks with skip connections has been a major factor in its success, and it is likely to continue to be an important architecture for years to come.

ResNet is a type of deep neural network that helps solve the problem of the "vanishing gradient" in very deep networks. The idea is to use "skip connections" to allow information to flow from earlier layers to later layers, which makes it easier for the network to learn. ResNet achieves this by using a special building block called a "residual block", which includes two convolutional layers and a shortcut connection that adds the input to the output of the second layer. ResNet has become very popular for image recognition tasks because it can handle very deep networks with many layers, and has achieved state-of-the-art performance on several benchmarks.

Explain short cut path and Main path of the Residual blocks

Residual Networks (ResNets) are a type of neural network architecture that were introduced in 2015 and are commonly used in computer vision tasks such as image classification. ResNets are characterized by the use of residual blocks, which consist of a shortcut path and a main path.

The main path of a residual block typically consists of two or more convolutional layers, followed by batch normalization and a non-linear activation function such as ReLU. The output of this main path is then added to the input of the residual block, which is passed through a shortcut path.

The shortcut path in a residual block is a direct connection from the input of the block to the output. This shortcut connection is typically achieved through the use of a 1×1 convolutional layer that matches the number of channels in the input and output feature maps. If the dimensions of the input and output feature maps do not match, additional padding or pooling layers may be used to ensure that they are compatible.

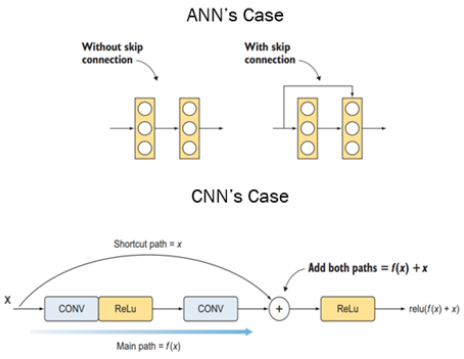
The purpose of the shortcut path in a residual block is to allow the network to learn residual functions, or the difference between the input and output of the block. This can help to overcome the problem of vanishing gradients, which can occur in very deep neural networks. By using the shortcut connection, the gradients can flow directly back to the earlier layers of the network, which can improve the overall training performance and accuracy.

Microsoft's approach to Vanishing Gradient

To solve the vanishing gradient problem, He et al. created a shortcut that allows the gradient to be directly backpropagated to earlier layers. These shortcuts are called as skip connections.

They are used to flow information from earlier layers in the network to later layers, creating an alternate shortcut path for the gradient to flow through.

Skip connections also allow the model to learn an identity function which ensures that the layer will perform at least as well as the previous layer



1. He et al. decided to do dimension downsampling using bottleneck 1 × 1 convolutional layers, similar to the Inception network
2. Residual blocks starts with a 1 × 1 convolutional layer to downsample the input dimension volume, and a 3 × 3 convolutional layer and another 1 × 1 convolutional layer to downsample the output.
3. This is a good technique to keep control of the volume dimensions across many layers. This configuration is called a bottleneck residual block
4. Since the volume dimensions are changing due to the bottleneck layers, we also sometimes downsample the shortcut path as well using the bottleneck layer.

Note: ResNets do not have a pooling layer in the network, they instead use strides=2 to downsample the width and height of the image in the reduce layer itself.

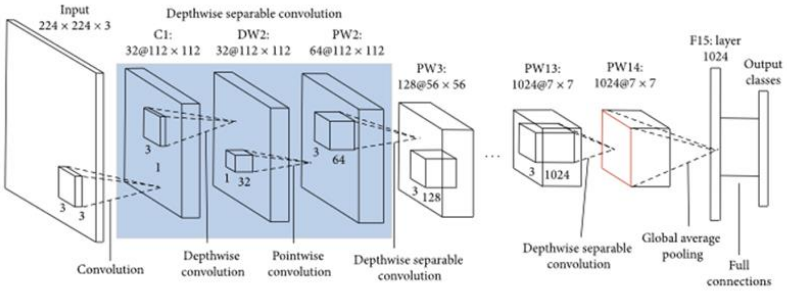
ResNet Summary

Residual blocks contain two paths

1. The shortcut path and the main path.
2. The main path consists of three convolutional layers, and we add a batch normalization layer to them:
   1. 1 × 1 convolutional layer
   2. 3 × 3 convolutional layer
   3. 1 × 1 convolutional layer
3. There are two ways to implement the shortcut path:
   1. **Regular shortcut**: Add the input dimensions to the main path.
   2. **Reduce shortcut**: Add a convolutional layer in the shortcut path before merging with the main path.

MobileNet

MobileNet



MobileNet: Google's approach at light weight models

A class of efficient models for mobile and embedded vision applications

Mobilenets make use of **Depth Wise Separable Convolutions along with reduce layer (1x1 convolution)** to build light weight deep neural networks

Two simple hyperparameters introduced that efficiently trade off between latency and accuracy

The core idea behind MobileNet is to use depthwise separable convolutions instead of traditional convolutions to reduce the computational cost of the network. Depthwise separable convolutions are composed of two separate operations: a depthwise convolution that applies a single filter to each input channel, followed by a pointwise convolution that applies a 1x1 filter to combine the output channels. This approach reduces the number of parameters and computations required by the network while still maintaining accuracy.

The MobileNet architecture consists of a series of depthwise separable convolutional layers followed by global average pooling and a fully connected layer. The depthwise separable convolutions are arranged in a sequence of building blocks, each of which contains one or more depthwise separable convolutional layers with a ReLU activation function.

In addition, MobileNet also incorporates a technique called "width multiplier", which reduces the number of channels in each layer of the network. The width multiplier is a hyperparameter that scales the number of channels in each layer by a factor of less than one, such as 0.5 or 0.25. This reduces the computational cost and memory requirements of the network while still maintaining its accuracy.

what are depthwise convolutions

Depthwise convolutions are a type of convolutional operation used in deep learning neural networks. They are different from traditional convolutions in that they apply a single filter to each input channel separately, rather than using a filter that combines information across all input channels.

More specifically, a depthwise convolution applies a filter of a certain size to each channel of the input, producing a feature map for each channel. These feature maps are then concatenated together to produce the final output.

The main advantage of depthwise convolutions is that they are much more computationally efficient than traditional convolutions, since they require fewer parameters and operations to perform the same operation. This makes them particularly well-suited for mobile and embedded devices with limited processing power and memory.

what are pointwise convolutions

Pointwise convolutions, also known as 1x1 convolutions or network-in-network layers, are a type of convolutional operation that involves applying a filter of size 1x1 to every pixel in a feature map.

Unlike traditional convolutional layers, which typically involve larger filters, pointwise convolutions have a receptive field of only one pixel. This means that they operate only on the channel dimension of the input data, and do not consider spatial relationships between pixels.

Pointwise convolutions are often used in convolutional neural networks (CNNs) to perform dimensionality reduction or to increase the number of channels in a feature map. They can also be used in conjunction with other types of convolutional layers, such as depthwise separable convolutions, to create more efficient and accurate network architectures.

Overall, pointwise convolutions are a powerful tool for improving the performance of CNNs, and are widely used in modern deep learning models.

The key difference between a normal convolutional layer and a depthwise convolution is that the depthwise convolution applies the convolution along only one spatial dimension (i.e. channel) while a normal convolution is applied across all spatial dimensions/channels at each step.

MobileNet's Two Model Lightening Hyperparameters

1. **Width Multiplier**: Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster.
   - The role of the width multiplier α is to thin a network uniformly at each layer.
   - For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN. The value of α is typically between [0,1]
2. **Resolution Multiplier**: A resolution multiplier ρ is used on the input image and the internal representation of every layer is subsequently reduced by the same multiplier. In practice we implicitly set ρ by setting the input resolution. Value of ρ is typically between [0,1] but as mentioned, in practice the input resolution itself is set typically - 224, 192, 160 or 128

What Do CNNS see?

What do CNNs see ?

1. Class Activation Maps – Naïve and Restricted Approach
2. Gradient Class Activation Maps (Grad-CAM) – Appropriate for all kinds of tasks
   - Image Classification
   - Image Captioning
   - Visual QnA
   - Image Segmentation

Class Activation Maps

what are class activation maps

Class activation maps (CAMs) are visualizations that highlight the most important regions of an image that are relevant to a specific classification decision made by a deep learning model. CAMs are typically generated using convolutional neural networks (CNNs), which are widely used in computer vision tasks such as image classification, object detection, and segmentation.

The basic idea behind CAMs is to use the last convolutional layer of a CNN, which produces a feature map that captures high-level abstract features of an input image. The CAM algorithm then generates a heatmap by weighting the feature map using the learned weights of the final fully connected layer of the network. This heatmap highlights the regions of the input image that contributed the most to the network's classification decision.

CAMs can be used to gain insights into how a deep learning model is making decisions and to identify areas of an image that are most important for classification. They have been used in a variety of applications, including object recognition, image segmentation, and medical image analysis.

Grad-CAMS

What are Grad Cams

Grad CAM (Gradient-weighted Class Activation Mapping) is a technique used in computer vision and deep learning to generate a visual explanation of the regions of an image that were most important in determining the output of a neural network. Like CAMs, Grad CAMs use the last convolutional layer of a CNN to create a heat map that highlights the relevant parts of an input image. However, unlike CAMs, which rely on the weights of the final fully connected layer, Grad CAMs use the gradients of the output with respect to the feature maps of the last convolutional layer.

The Grad CAM algorithm computes the gradients of the final output of the network with respect to the feature maps of the last convolutional layer. These gradients are then weighted by the global average pooling of the gradients. The resulting weighted gradients are then passed through a ReLU activation function to obtain the Grad CAM heatmap.

The advantage of Grad CAMs over CAMs is that they are more interpretable and can provide more fine-grained information about the regions of an image that are important for classification. Grad CAMs have been used in a variety of applications, including object recognition, image segmentation, and visual question answering.

Grad-CAMs: Binary Image Classification

1. Create a custom model in and deep learning framework tf.keras where
   1. Input: image
   2. Output: Gives last Convolution Output and Classification Probability score
2. Get the Classification probability score: Sigmoid activation value
3. Calculate the gradient of this classification activation value wrt to the last Convolution Output
4. If class probability score/sigmoid activation >= 0.5
   1. Take the average of gradient matrix
   2. Else if its < 0.5 put a negative sign on the gradient matrix and then take its average
5. Project this averaged gradient on the last convolutional feature map to form a image importance gradient heatmap
6. We only want the positive impact of this gradient heatmap
   1. Pass the gradient heatmap through a relu function
   2. Normalize the gradient heatmap
7. Resize the gradient heatmap and project it on to the image

Grad-CAMs: Multiclass Image Classification

1. Create a custom model in and deep learning framework tf.keras where
   1. Input: image
   2. Output: Gives last Convolution Output and Classification Probability score
2. Get the Classification probability score: Softmax activation value
3. Calculate the gradient of this classification activation value wrt to the last Convolution Output
4. Get the highest voted class probability/softmax activation using max function
   1. Take the average of gradient matrix
5. Project this averaged gradient on the last convolutional feature map to form a image importance gradient heatmap
6. We only want the positive impact of this gradient heatmap
   1. Pass the gradient heatmap through a relu function
   2. Normalize the gradient heatmap
7. Resize the gradient heatmap and project it on to the image

PPT 7

Visual Embeddings

Image Embeddings: An image embedding is a lower-dimensional representation of the image. In other words, it is a dense vector representation of the image which can be used for different computer vision tasks. E.g., Classification

Usually is the output of the layer before the final layer (last fully connected layer)

This way, comparisons between images can be performed by measuring their pairwise distances in this embedding space.

An image embedding is a vector representation of an image that captures its high-level features and semantic information in a lower-dimensional space. These embeddings can be obtained by passing an image through a pre-trained neural network, which extracts the features of the image in a compressed

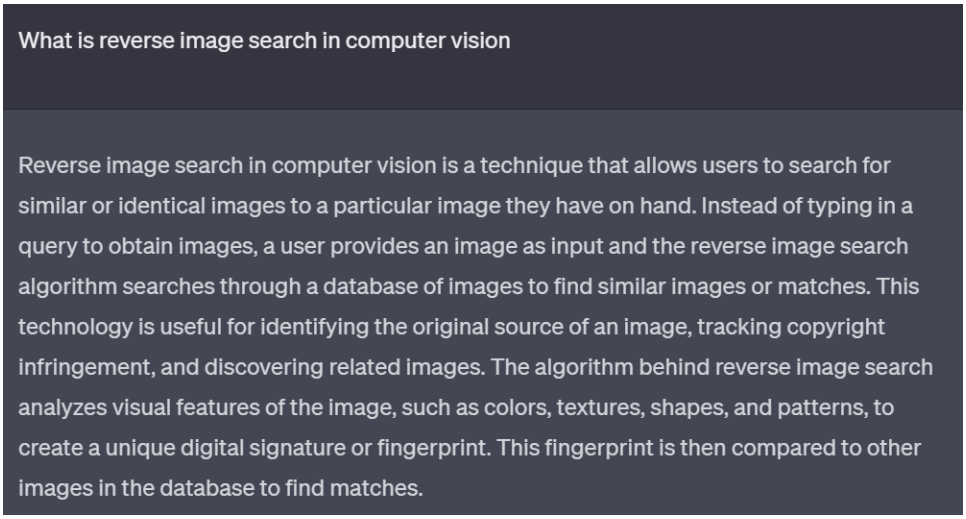
form. The resulting vector is a numerical representation of the image that can be used for tasks such as image classification, object detection, and image retrieval.

These are basically the output that we get during the training but just before the final layer.

Applications of Visual Embeddings

- Image Recommendation Systems
- Reverse Image Search Engines
- Face Recognition
- Object Re-Identification

Reverse Image Search/Image Recommendation Approach



Algorithm of Visual Embeddings

- Extract visual embeddings using a pretrained network
- Store the vector embeddings in a database
- Provide input image -> calculate visual vector embedding
- Calculate distance between input image embeddings and all the image embeddings in the database
- Display the k-Nearest Neighbours based on the distance calculated

Subjective metrics such as hit-rate and A/B tests are used to evaluate the success of these tasks

One Shot Learning: Face Recognition

One Shot Learning: Face Recognition

One-shot learning is a type of machine learning where a model is trained to recognize objects or patterns after being presented with just a single example. One area where one-shot learning has been applied is face recognition.

In face recognition, one-shot learning can be used to identify individuals from just one image of their face. This can be useful in situations where only a single image of a person is available, such as in surveillance footage or social media profile pictures.

One approach to one-shot learning for face recognition is to use a siamese neural network. This type of network takes two input images and learns to output a similarity score between them, indicating how similar the two images are. During training, the network is presented with pairs of images, with the correct pairs having a high similarity score and the incorrect pairs having a low similarity score.

After training, the network can be used for one-shot learning by comparing a new image to a set of reference images, and outputting the similarity score for each comparison. The reference image with the highest similarity score is then identified as the matching individual.

Another approach to one-shot learning for face recognition is to use a generative model to generate additional images of an individual from a single reference image. These generated images can then be used to train a traditional face recognition model, such as a convolutional neural network. This approach can be useful when there are very few reference images available for a particular individual.

Overall, one-shot learning has the potential to improve the accuracy of face recognition in scenarios where only a single image is available, although further research is needed to fully evaluate its effectiveness.

Mode of Face Recognition

1. Face Verification
2. Face Identification

There are two main modes of face recognition: verification and identification.

1. Verification: Verification, also known as authentication or one-to-one recognition, involves comparing a person's facial features with a stored template or database to determine if they are who they claim to be. In verification mode, the system only needs to compare the input image or video to a single template or identity, and the system either accepts or rejects the claimed identity. This mode is commonly used in security applications, such as unlocking a smartphone or accessing a secure area.
2. Identification: Identification, also known as one-to-many recognition, involves searching a database of stored templates or identities to find a match for the input image or video. In identification mode, the system compares the input image or video to all the templates in the database and returns a list of possible matches, ranked in order of similarity. This mode is commonly used in applications such as law enforcement, where a suspect's identity needs to be determined from a large database of known individuals.

Face Verification

Face Verification

1. Face image embeddings of users stored in a database
2. User provides and input image, saying he is person A
3. Distance = L2(User's face embeddings, Person A's face embeddings)
4. If Distance < Distance threshold
    - The person is verified as person A
5. Else:
    - It is not person A

Face Identification

What is face identification in computer vision

Face identification in computer vision is the process of identifying and verifying the identity of an individual by analyzing their facial features. This process involves capturing an image or a video of a face and analyzing the unique features of the face such as the distance between the eyes, the shape of the nose, and the contour of the jawline.

These features are then compared to a database of known faces to determine the identity of the individual. Face identification can be used for a variety of applications such as access control, surveillance, and biometric authentication.

Face Identification

1. Face image embeddings of all users stored in a database
2. Person of interest identified
3. Person's face embeddings calculated
4. Distance = L2(person's face embeddings, face embeddings in DB)
5. Find the person id from the database with lowest distance
6. If distance < distance thresh
   - Person is person id with the lowest distance
7. Else
   - Person's image is not present in the database

Face Recognition

Face Recognition

1. Face Detection
   - Haar Cascades
   - Object Detection based models – MTCNN – Covered in Object Detection
   - Mediapipe – To be covered in next lecture
2. Face Alignment
3. Embedding Extraction
4. Face Identification/Verification

Siamese Network

What is a siamese network

A Siamese network is a type of neural network architecture that consists of two or more identical subnetworks that share the same weights and architecture. This network architecture is often used in tasks that involve comparing two input sequences, such as in face recognition, signature verification, and image matching.

The Siamese network takes two input images and feeds them through two identical subnetworks. Each subnetwork consists of multiple layers of convolutional neural networks (CNNs) and pooling layers that extract high-level features from the input images. The two subnetworks then produce two feature vectors that represent the input images.

Siamese network is used to solve tasks involving similarity or matching.

The siamese network takes two input images and extracts features by processing them with the same set of convolutional and pooling layers. A fully connected layer receives the characteristics from both input images after they have been concatenated, and it produces a single value that indicates how similar the two input images are.

A siamese network is trained by feeding it pairs of input images along with labels indicating whether they are the same or different. The network then learns to increase distance if two photos are identified as "different," and minimize distance if two images are labelled as "same."

For tasks including image identification, object tracking, face recognition, and similarity-based retrieval systems, the siamese network design is frequently utilized.

Contrastive Loss:

One common loss function used in Siamese networks is the contrastive loss function. This loss function penalizes the network for misclassifying similar pairs of images as dissimilar, and vice versa. The contrastive loss function is defined as follows:

$$L = (1/2m) * sum(y * d^2 + (1 - y) * max(margin - d, 0)^2)$$

where m is the batch size, y is the binary label (0 for dissimilar pairs and 1 for similar pairs), d is the distance between the two feature vectors, and margin is a hyperparameter that controls the distance threshold at which two feature vectors are considered similar or dissimilar.

Triplet Loss:

Another common loss function used in Siamese networks is the triplet loss function. This loss function requires three input images: an anchor image, a positive image (an image similar to the anchor), and a negative image (an image dissimilar to the anchor). The goal is to learn a feature representation that maximizes the distance between the anchor and negative images while minimizing the distance between the anchor and positive images. The triplet loss function is defined as follows:

$$L = max(0, d(a, p) - d(a, n) + margin)$$

where d(a, p) is the distance between the anchor and positive images, d(a, n) is the distance between the anchor and negative images, and margin is a hyperparameter that controls the distance threshold between the positive and negative images.

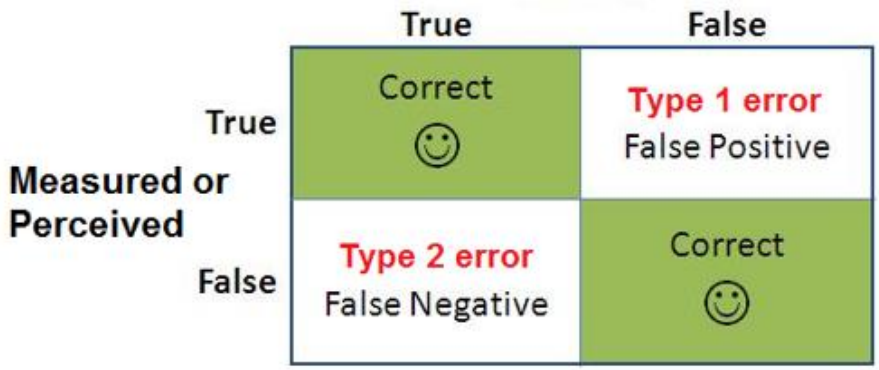
Important Aspects of Face Recognition Systems

- Fairness
- Transparency
- Accountability
- Non-discrimination
- Notice and Consent
- Lawful Surveillance

Applications of Face Recognition

- Auto Tagging on social media platforms
- Authentication via Face Verification
- Missing Person Identification

PPT – 8



You decide to get tested for COVID-19 based on mild symptoms. There are two errors that could potentially occur:

• Type I error (false positive): the test result says you have coronavirus, but you actually don't.

• Type II error (false negative): the test result says you don't have coronavirus, but you actually do.

You build a model for predicting if a credit card user is going to default or not. The bank that uses this model might want to take some precautionary measures based on the prediction.

• Type I error (false positive): the model result says user will default, but he actually doesn't.

Type II error (false negative): the model result says user will not default, but he actually does

Recall Sensitivity True positive rate (TPR)	$\frac{TP}{FN+TP}=\frac{TP}{P}$
False positive rate (FPR) False alarm rate	$\frac{FP}{TN+FP}=\frac{FP}{N}$
Specificity True negative rate (TNR)	$\frac{TN}{TN+FP}=\frac{TN}{N}=1-FPR$
Precision	$\frac{TP}{TP+FP}$
False negative rate (FNR)	$\frac{FN}{FN+TP}=\frac{FN}{P}$
Accuracy	$\frac{TP+TN}{P+N}=\frac{TP+TN}{TP+TN+FP+FN}$

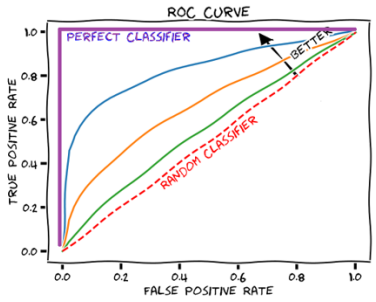
An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

predicted → / real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

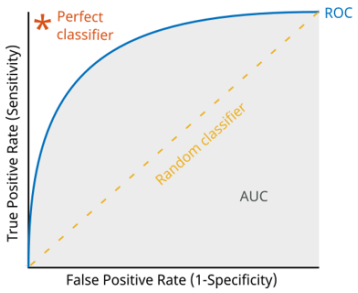
$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN}+\text{FP}}$$



AUC provides an aggregate measure of performance across all possible classification thresholds.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.



Binary Cross Entropy

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

Categorical Cross Entropy

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Video Processing Pipelines using Classification Models

Naïve Video Processing Pipeline

1.Give a video input source

2.Read each frame of the video

3.Classify each frame in the video (Bottleneck)

4.Display the prediction on the video

Multi-threading v/s Multi-processing

Multi-threading refers to a technique where a single process creates multiple threads of execution that can run concurrently within the same address space. Each thread shares the same memory and resources of the process but can perform different tasks simultaneously.

Multi-processing, on the other hand, involves creating multiple independent processes that run concurrently, each with its own memory space and resources. Each process can perform its own task independently of the other processes.

In simple terms, multi-threading is like having multiple workers in a single room working on different tasks at the same time, while multi-processing is like having multiple workers in separate rooms working on their own tasks independently.

PPT-9

Object Detection

Object detection is a computer vision task that involves locating and identifying objects of interest in digital images or video streams. The goal of object detection is to automatically and accurately detect and localize multiple objects within an image, and often classify them into predefined categories.

Localizing an object in an image involves identifying the position of the object within the image by drawing a bounding box around it. This is typically done using a deep learning model, such as a convolutional neural network (CNN)

Bounding Box

Bounding box regression: Bounding box regression is a computer vision technique used in object detection tasks to predict the coordinates of a bounding box around an object of interest within an image. The bounding box is typically represented as a rectangular box that tightly encloses the object, and it is defined by four coordinates: the x and y coordinates of the top-left corner, and the width and height of the box.

The goal of bounding box regression is to accurately predict these four coordinates that define the bounding box around the object.

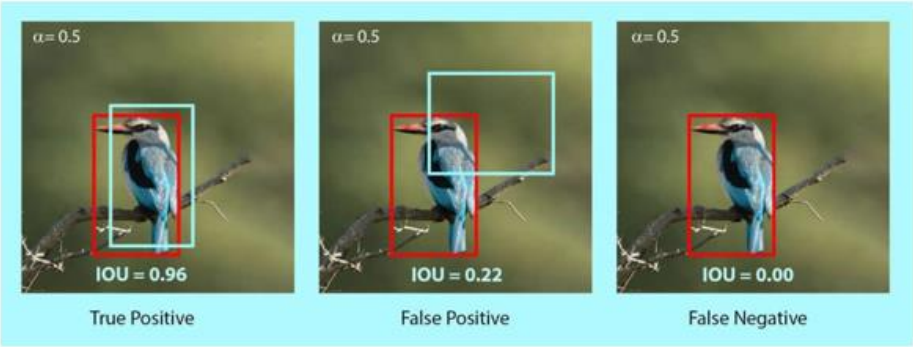
- That is why we normalize the coordinates
  - xmin: 83/233 (xmin/width)
  - ymin: 29/350 (ymin/height)
  - xmax: 197/233 (xmax/width)
  - ymax: 142/350 (ymax/height)
- If we resize the above image to 224x224x3
  - xmin = int(xmin*224)
  - ymin = int(ymin*224)
  - xmax = int(xmax*224)
  - ymax = int(ymax*224)

Bounding box regression with classification module works out fine but If you have multiple objects of a similar class in an image it will only detect one of them which it is most confident about.

Intersection over Union:

Intersection over Union: Intersection over Union (IoU) is a common evaluation metric used in object detection and image segmentation tasks. It measures the overlap between the predicted bounding box and the ground truth regions. It is expressed as a ratio, ranging from 0 to 1, where 0 indicates no overlap and 1 indicates a perfect match. A higher IoU indicates a better overlap between the predicted and

ground truth regions, and therefore a more accurate prediction. A typical threshold for object detection tasks is 0.5, meaning that a predicted bounding box with an IoU of 0.5 or higher with the ground truth bounding box is considered a correct detection.



RCNN

RCNN stands for region-based Convolutional Neural Network. It is a type of deep learning architecture used for object detection tasks in computer vision. It was one of the first large, successful applications of convolutional neural networks to the problem of object detection and localization.

The four components you mentioned in the context of RCNN are important stages in the object detection pipeline:

Region Proposal (Extract Regions of Interest (ROI)): This component is responsible for generating potential region proposals, which are bounding box proposals that may contain objects. The region proposal module identifies regions in the input image that are likely to contain objects and generates a set of region proposals. These proposals are typically generated using techniques such as selective search

Feature Extraction Module: Once the region proposals are generated, the next step is to extract features from these proposals. The feature extraction module typically involves passing the region proposals through a convolutional neural network (CNN) to extract meaningful features that can represent the visual content of the region proposals. The CNN is typically pre-trained on a large dataset, such as ImageNet, and can learn high-level representations of visual features

Classification Module: After feature extraction, the RCNN architecture includes a classification module that is responsible for predicting the class labels of the objects within the region proposals. The features extracted from the region proposals are fed into a classifier, such as a fully connected neural network, which is trained to predict the class labels of the objects. The classifier typically has softmax activation.

Since Selective Search takes care of the localization using its own proposed regions bounding box we do not explicitly perform the bounding box regression step.

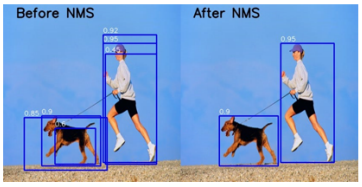
But to improve the localization step further we include the bounding box regression to give more refined coordinates of the bounding box.

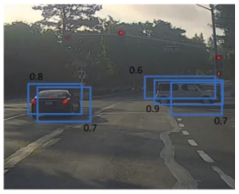
Disadvantages of RCNN:

- Object Detection is very slow
- High computational and memory requirements
- Training complexity
- Training is expensive
- 2 stage pipeline
  A) region proposal
  B) object detection on region proposal

Multiple Bbox single object: NMS

- Non max suppression is a technique used mainly in object detection that aims at selecting the best bounding box out of a set of overlapping boxes
- Sort the bboxes in descending order of class confidence/probability score
- remove the boxex which have a confidence score < threshold confidence score (e.g. remove boxes with score < 0.5)
- Loop over the remaining boxes
  - Start with the box which has the highest confidence
  - Discard other boxes if they have an IoU > predefined IoU thresh with the highest confidence box
  - Repeat the steps until all the boxes are either taken as output prediction or discarded

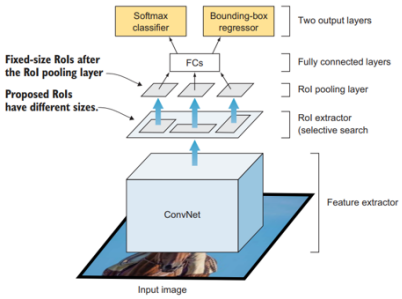




Fast RCNN

Fast RCNN

1. Pass the input image to the feature extractor just once
2. Pass the input image to selective search algorithm to get RoIs/Region Proposals
3. Feature maps are produced by the pooling layer of the pretrained network
4. RoIs from step 2 are projected onto the feature maps
5. RoI pooling is performed on the projected RoIs
6. RoI pooling output is passed to the fully connected layers
7. Fully connected layer is connected to the multi-head output
   1. Classifier
   2. Bbox Regressor



Fast R-CNN (Region-based Convolutional Neural Network) is a popular object detection framework introduced by Ross Girshick in 2015 as an improvement over the original R-CNN approach. Fast R-CNN is designed to be faster and more efficient by addressing some of the limitations of R-CNN, which was computationally expensive and slow.

The main idea behind Fast R-CNN is to perform both region proposal generation and object classification in a single neural network, which makes it end-to-end trainable and eliminates the need for computationally expensive intermediate steps.

RoI Projection: After generating region proposals using the Region Proposal Network (RPN), the RoI projection step maps the region proposals from the input image coordinates to the corresponding feature map coordinates. This allows the network to extract features from the feature map for each region proposal.

ROI Pooling: RoI pooling is used to extract fixed-size feature maps from the region proposals. It divides each region proposal into a fixed number of bins and applies max pooling within each bin to obtain a fixed-size feature map. This fixed-size feature map is then used as input for subsequent layers in the network.

Classification and Bounding Box Regression: The fixed-size feature maps obtained from RoI pooling are fed into separate fully connected layers for object classification and bounding box regression. The classification layer predicts class probabilities for each region proposal, and the bounding box regression layer predicts refined bounding box coordinates for each region proposal to improve the accuracy of the object detections.

Loss Function: Fast R-CNN uses a multi-task loss function to train the network. The classification loss measures the difference between predicted class probabilities and ground truth class labels, and the bounding box regression loss measures the difference between predicted bounding box coordinates and ground truth bounding box coordinates.

Loss function

Fast R-CNN

- Multi-task loss

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

Where $L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i)$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

ADVANTAGES

1. Requires only one propagation for each image unlike 2000 in RCNN
2. Three stage training from RCNN reduced to one stage training
3. Faster in inference time than RCNN: 2 seconds (0.32 seconds for CNN + time required for selective search)

DISADVANTAGES

1. It is still a two stage pipeline
   1. Selective Search Region Proposal
   2. CNN
2. Selective search still remains to be a bottleneck as it is very slow to generate region proposals. 2 seconds of inference time is too much for real time object detection in videos.

Faster RCNN

Faster R-CNN (Region-based Convolutional Neural Network) is a deep learning model for object detection. It is an extension of the original R-CNN model and is faster and more accurate in detecting objects in an image. We now know that the major bottleneck with region based CNNs till now is the traditional and slow region proposal method (selective search). Thus, we need to replace the selective search method with some other region proposal method.

Faster RCNN: Region Proposal Aspect Ratios

- To detect most kinds of objects you typically use one of the 3 aspect ratios given below
  - Vertical rectangle
  - Square
  - Horizontal rectangle
- Most of the object will be obstructing the one behind and hence you need overlapping ROIs
- Objects maybe of different size therefore you need ROIs of different scale

The RPN generates object proposals by sliding a small network (usually a few convolutional layers) over the feature map output of the backbone network (usually a deep CNN). The network takes a set of anchor boxes of different scales and aspect ratios and predicts the probability of each anchor box being an object and the offset between the anchor box and the ground-truth object.

Aspect ratios refer to the ratio of the width to the height of the anchor boxes. Faster R-CNN uses anchor boxes of different aspect ratios to handle objects with different shapes. For example, objects like bicycles and cars are usually wider than they are tall, while objects like humans and dogs are more or less square. By using anchor boxes of different aspect ratios, Faster R-CNN can better fit the object shapes and achieve better detection accuracy.

Faster RCNN: Components

Faster R-CNN (Region-based Convolutional Neural Network) is a deep learning model that is widely used for object detection tasks. It is an improved version of the original R-CNN and Fast R-CNN models, and it achieves higher accuracy and faster speed by introducing several key components. The main components of the Faster R-CNN model are:

1. Backbone Network: This is a convolutional neural network (CNN) that is used to extract feature maps from the input image. Popular backbone networks used in Faster R-CNN include VGG, ResNet, and Inception.
2. Region Proposal Network (RPN): The RPN generates object proposals by sliding a small network over the feature map output of the backbone network. The RPN takes as input the feature map and generates a set of object proposals along with a corresponding objectness score.
3. Anchor Boxes: The RPN uses anchor boxes to generate object proposals. Anchor boxes are pre-defined boxes of different scales and aspect ratios that are placed at various locations on the feature map. The RPN predicts the offset and objectness score for each anchor box, which is used to generate the final object proposals.
4. RoI (Region of Interest) Pooling: The RoI pooling layer extracts a fixed-size feature map from the feature map output of the backbone network for each object proposal generated by the RPN. This allows the network to use a fixed-size input for each object proposal, regardless of its original size, making it easier for the network to learn.
5. Classifier and Box Regressor: The RoI-pooled feature maps are fed into two separate fully connected layers, one for object classification and the other for bounding box regression. The classifier predicts the class of the object in the proposal, while the box regressor predicts the offset between the proposal and the ground-truth object.

Region Proposal Network

Explain RPN for Faster RNN

The Region Proposal Network (RPN) is a key component of the Faster R-CNN (Region-based Convolutional Neural Network) model for object detection. It generates object proposals by sliding a small network over the feature map output of the backbone network and predicting the probability of each anchor box being an object and the offset between the anchor box and the ground-truth object.

The RPN consists of two main components: the anchor boxes and the convolutional layers. The anchor boxes are pre-defined boxes of different scales and aspect ratios that are placed at various locations on the feature map. These boxes act as potential regions of interest for the RPN to evaluate.

The convolutional layers are used to process the feature maps generated by the backbone network and predict the objectness score and the bounding box regression offset for each anchor box. The objectness score measures the likelihood that the anchor box contains an object, while the bounding box regression offset is used to adjust the size and position of the anchor box to better fit the object.

To generate object proposals, the RPN selects a set of anchor boxes with high objectness scores and applies the bounding box regression offsets to these boxes to generate new proposals. The proposals are then passed through a Region of Interest (RoI) pooling layer to generate fixed-size feature maps for each proposal, which are used by the classification and regression layers of the Faster R-CNN model to predict the class and location of the object in each proposal.

Explain Anchor Box in details

Anchor boxes are a key component of the Faster R-CNN (Region-based Convolutional Neural Network) model for object detection. They are pre-defined boxes of different scales and aspect ratios that are used to generate object proposals by the Region Proposal Network (RPN) component of the Faster R-CNN model.

The anchor boxes are placed at various locations and scales on the feature map output of the backbone network, which is typically a convolutional neural network (CNN) used for feature extraction. Each anchor box represents a potential object region that the RPN will evaluate to generate object proposals.

The anchor boxes are defined by their center coordinates and width and height dimensions. To generate object proposals, the RPN slides a small network over the feature map and evaluates each anchor box by predicting the probability that it contains an object and the offset between the anchor box and the ground-truth object.

The anchor boxes are defined with different scales and aspect ratios to handle objects of different sizes and shapes. For example, objects like bicycles and cars are usually wider than they are tall, while objects like humans and dogs are more or less square. By using anchor boxes of different aspect ratios, Faster R-CNN can better fit the object shapes and achieve better detection accuracy.

How are the region proposed bounding boxes predicted

The region proposed bounding boxes are predicted by the Region Proposal Network (RPN) component of the Faster R-CNN (Region-based Convolutional Neural Network) model for object detection. The RPN generates object proposals by evaluating pre-defined anchor boxes and predicting the offsets between the anchor boxes and the ground-truth object.

To predict the region proposed bounding boxes, the RPN uses two sets of outputs for each anchor box: the objectness score and the bounding box regression offsets. The objectness score measures the probability that the anchor box contains an object, while the bounding box regression offsets are used to adjust the size and position of the anchor box to better fit the object.

The objectness score and the bounding box regression offsets are predicted by a set of convolutional layers that are applied to the feature map output of the backbone network. The convolutional layers have a small receptive field, typically 3×3, and slide over the feature map to generate feature vectors for each anchor box. The feature vectors are then passed through two sets of fully connected layers to predict the objectness score and the bounding box regression offsets.

The objectness score is computed using a binary classification task, where the output is either 1 or 0 to indicate whether the anchor box contains an object or not. The bounding box regression offsets are computed using a regression task, where the output is a set of four values that indicate the x and y offset, width, and height of the predicted bounding box relative to the anchor box.

To generate region proposed bounding boxes, the RPN selects a set of anchor boxes with high objectness scores and applies the bounding box regression offsets to these boxes to generate new proposals. The proposals are then passed through a Region of Interest (RoI) pooling layer to generate fixed-size feature maps for each proposal, which are used by the classification and regression layers of the Faster R-CNN model to predict the class and location of the object in each proposal.

Overall, the prediction of region proposed bounding boxes in Faster R-CNN is an efficient and effective way to generate object proposals and improve the accuracy and speed of object detection tasks. By using pre-defined anchor boxes and a small convolutional network to predict objectness scores and bounding box regression offsets, the RPN can quickly and accurately generate object proposals with high recall and low computational cost.

RPN: Loss Function

$$\mathscr{L}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i \mathscr{L}_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}} \sum_i p_i^* \mathscr{L}_{reg}(t_i, t_i^*)$$

$$\begin{aligned} p_i^* &= 1 \quad \text{if anchor box contains ground truth object} \\ &= 0 \quad \text{otherwise} \\ p_i &= \text{predicted probability of anchor box containing an object} \\ N_{cls} &= \text{batch-size} \\ N_{reg} &= \text{batch-size} \times k \\ k &= \text{anchor boxes} \end{aligned}$$

Mean Average Precision (mAP)

Mean Average Precision (mAP) is a common performance metric used in object detection to evaluate the accuracy of an object detection model. It measures the precision and recall of the model's predictions for each object class, and then computes the average precision across all classes.

Precision: It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model. In other words, it measures the percentage of times the model correctly predicted a positive outcome among all the times it predicted a positive outcome.

P = TP/ (TP + FP)

= TP / Total True Predicted

The value ranges from 0 to 1. With 1 indicating perfect precision

Recall: It is defined as the ratio of true positive predictions to the total number of actual positives in the dataset. In other words, it measures the percentage of times the model correctly identified a positive outcome among all the times that it should have identified a positive outcome.

R = TP / (TP + FN)

= TP / Total Ground Truths

The value of Recall also ranges from 0 to 1. With 1 indicating perfect recall

Average Precision: It is defined as the area under the precision-recall curve.

Mean Average Precision: Thus, Mean Average Precision or mAP is the average of AP over all detected classes.

mAP = 1/n * sum(AP), where n is the number of classes.

PPT 10

YOLO V1

YOLOv1, or You Only Look Once version 1, is a real-time object detection algorithm that was introduced by Joseph Redmon et al. in 2015. It is a convolutional neural network (CNN) based algorithm that is known for its speed and accuracy in detecting objects in images and videos. The main idea behind YOLOv1 is to divide an image into a fixed grid and directly predict the bounding boxes and class labels of objects within each grid cell in a single pass. The network only looks at the image once to detect multiple objects. Thus, it is called YOLO, You Only Look Once.

Grid-based Division: YOLOv1 divides the input image into a fixed grid of cells, typically, say, 7x7 or 13x13 cells, depending on the configuration. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Bounding Box Prediction: For each grid cell, YOLOv1 predicts the coordinates of the bounding box that encloses the object, relative to the cell's location. Each grid cell predicts B bounding boxes (B=2 in the paper) and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object. i.e., the probability of prediction of the box

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width w and height h are predicted relative to the whole image. The confidence represents the Intersection Over Union (IOU) between the predicted box and any ground truth box.

Objectness Score: YOLOv1 predicts an "objectness" score for each bounding box, which represents the likelihood of an object being present inside the box. The objectness score is used to filter out irrelevant bounding boxes that do not contain objects.

1. YOLOv1 was originally trained on PascalVOC2007 dataset which had 20 classes so in that case C=20
2. S = 7, B = 2, C = 20
3. S x S x (B x 5 + C) = 7x7x(2x5+20) = 7x7x30 is the output shape of the network

Class Prediction: YOLOv1 also predicts the class label of the object present in each bounding box.

The model consists of 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers.

Input Size: 448x448x3, Output size: 7x7x30 (PascalVOC2007)

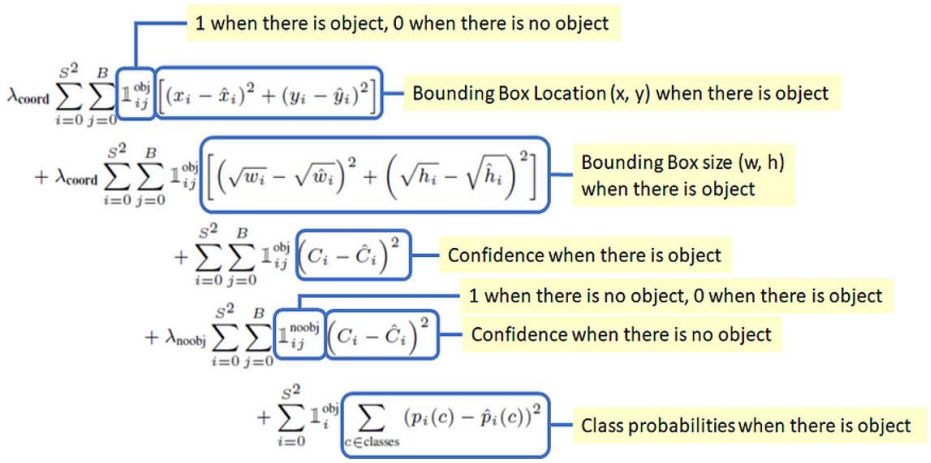
Loss Function: The loss function in YOLOV1 is a mathematical way to measure how accurate the predictions of the object detection model are. It includes two main components: the localization error and the classification error.

For localization, the model tries to predict the position and size of the object within a grid cell. If the cell contains an object, the model computes the squared difference between the predicted and ground truth coordinates and size of the bounding box. If there is no object in the cell, the localization loss is not computed to avoid pushing the confidence scores of empty cells towards zero.

For classification, the model predicts the probability of the object belonging to different classes. The classification loss is computed as the squared difference between the predicted and ground truth class probabilities for the cells that contain objects.

To avoid the model becoming unstable and over-fitting on empty cells, the loss function also includes a penalty for incorrect confidence scores for empty cells. This helps to balance the loss from cells containing objects and those without objects.

Overall, the loss function in YOLOV1 aims to improve the accuracy of object detection predictions while maintaining a balance between empty and occupied grid cells.



Difference b/w RCNN, Fast-RCNN, Faster-RCNN, and YOLOV1

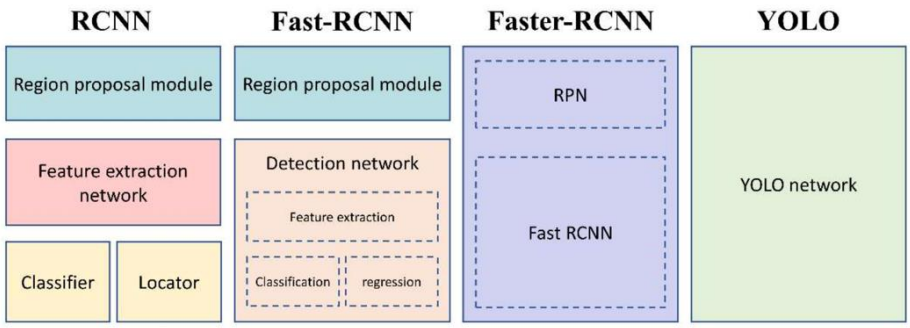
RCNN: The RCNN model proposes regions of interest (RoIs) in the image, then extracts features from each proposed region using a convolutional neural network (CNN). The features are then classified using a support vector machine (SVM).

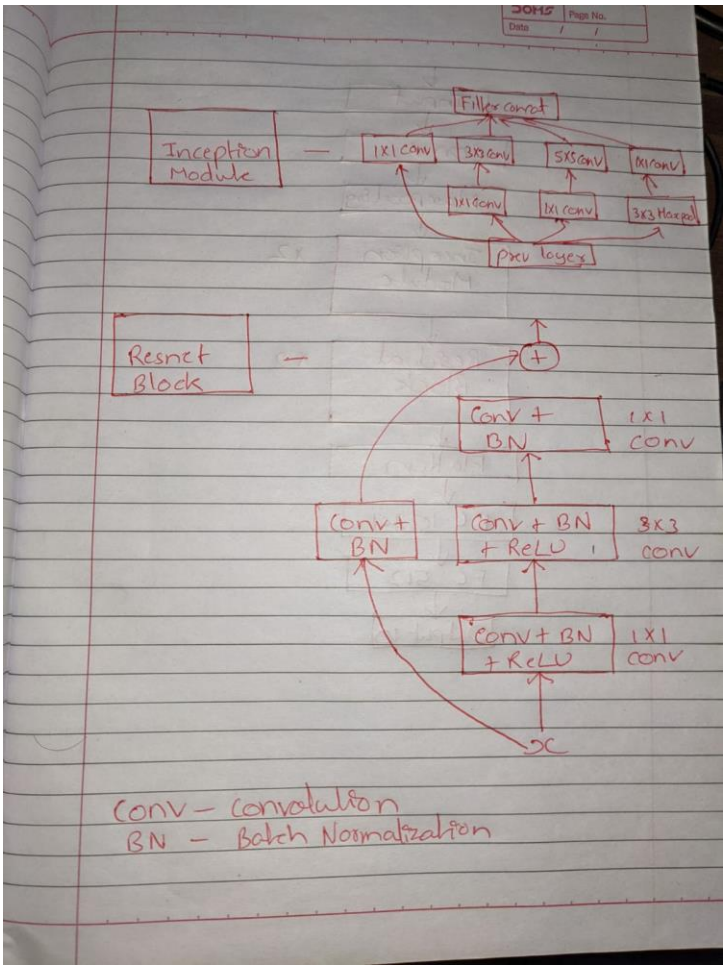
Fast-RCNN: The Fast-RCNN model improves on RCNN by sharing computation across the RoIs, allowing for faster processing. Instead of processing each region separately, the entire image is processed by the CNN, and the features for each region are extracted from the CNN's feature map.

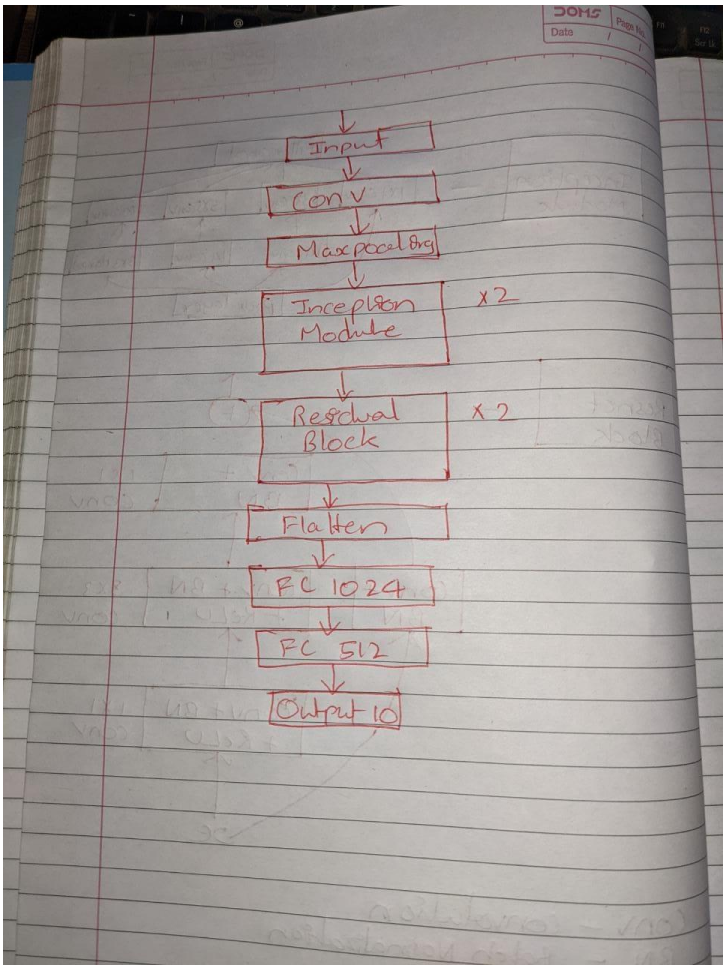
Faster-RCNN: The Faster-RCNN model improves on Fast-RCNN by adding a Region Proposal Network (RPN) to the architecture. The RPN generates region proposals directly from the CNN's feature map, which allows for even faster processing and better accuracy.

YOLOV1: YOLO (You Only Look Once) is a different approach to object detection. It divides the image into a grid and predicts the class probabilities and bounding boxes for each grid cell. The predictions are made using a single neural network that simultaneously predicts the class probabilities and the bounding boxes.

RCNN Family vs YOLOv1




Inception Module
Filter concat
1X1 conv
3X3 conv
5X5 conv
1X1 conv
1X1 conv
1X1 conv
1X1 conv
3X3 Max pool
prev layer
Resnet Block
+
Conv + BN
1X1 conv
Conv + BN
Conv + BN + ReLU
3X3 conv
Conv + BN + ReLU
1X1 conv
Conv - convolution
BN - Batch Normalization


Input
Conv
Maxpool 8x8
Inception Module
x2
Residual Block
x2
Flatten
FC 1024
FC 512
Output 10