PPT 1 - CVA

Computer Vision

Traditional Computer Vision

1. Relies on basic image processing techniques
2. Handcrafted feature extraction/Mathematical feature extraction
3. Classic Machine Learning techniques e.g. Image classification using SVM, Naïve Bayes, Logistic Regression, etc. Image Segmentation using K-Means

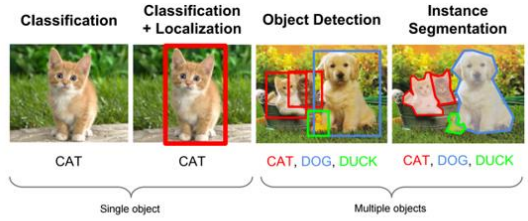
Deep Learning Based Computer Vision

1. Data Driven approach (requires image data)
2. Feature extraction and pattern recognition is **learnt** during the training process
3. Training neural networks on a dataset of images for the desired task (classification, segmentation, object detection, etc)

Typical Tasks in Computer Vision

Recognition

1. Object Classification/Image Classification
2. Object Identification
3. Object Detection
4. Content Based Image Retrieval (Reverse Image Search)
5. Pose estimation
6. OCR – Optical Character Recognition
7. Face Recognition



Difference between Object Detection and Image Classification

Let us break down these techniques, to know the difference between them. When you look at a picture of a dog you can instantly say it's an image of an animal i.e. tell what the image is about. This is what image classification is all about. As long as there is only one object, image classification techniques can be used. But if we have multiple objects, that's when the concept of Object Detection comes into play. By building rectangular boxes around the object of interest we can help the machine recognize the object each box contains. We can also indicate the exact location of the objects using this method. It is possible for a single picture to contain many objects, so multiple bounding boxes may be shown. Object detection applications are limitless, but they generally identify and detect the real-objects such as human beings, buildings, cars and many more. Additionally, a machine needs a lot of labeled data of different kinds of objects for it to recognize those objects in the future. This means the ML model being trained on that labeled dataset will have a better chance to make accurate predictions. Moreover, there are several companies which offer data annotation services. You just have to choose the right one based on your requirements. This technique is widely applied in people/object tracking applications, video surveillance cameras which I will elaborate further.

> Motion analysis is an overarching task that can be broken down into many components.

- **Object Detection**
- **Object Localisation**
- **Object Segmentation**
- **Tracking**
- **Motion Detection**
- **Pose estimation**

The phrase ‘motion analysis’ is self-descriptive; in layman terms, it’s generating an understanding of how an object moves within an environment.

But let’s describe motion analysis with more technical terms

> Motion analysis is the study of locomotion and trajectory of objects bodies.

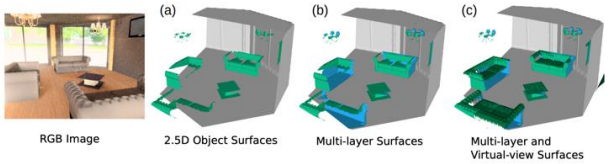
Now we have an overview of what motion analysis is, but where is it used?

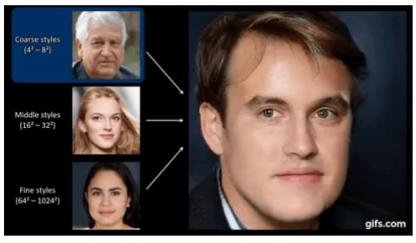
Motion analysis can be applied in various industries and disciplines. In medicinal related institutions, motion analysis is used as a non-intrusive method of observing the movement of patients with mobility impairments.

Typical Tasks in Computer Vision

Scene Reconstruction and Image Generation

1. Given one or (typically) more images of a scene, or a video, scene reconstruction aims at computing the 3D model of the scene
2. Generation of new synthetic instances of images that can pass for real images

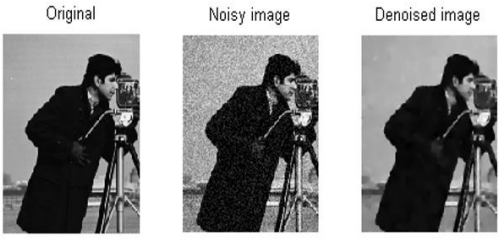




Typical Tasks in Computer Vision

Image Restoration

1. Aim is to remove noise from images
2. Restoring the erroneous parts of the images



Applications of Computer Vision

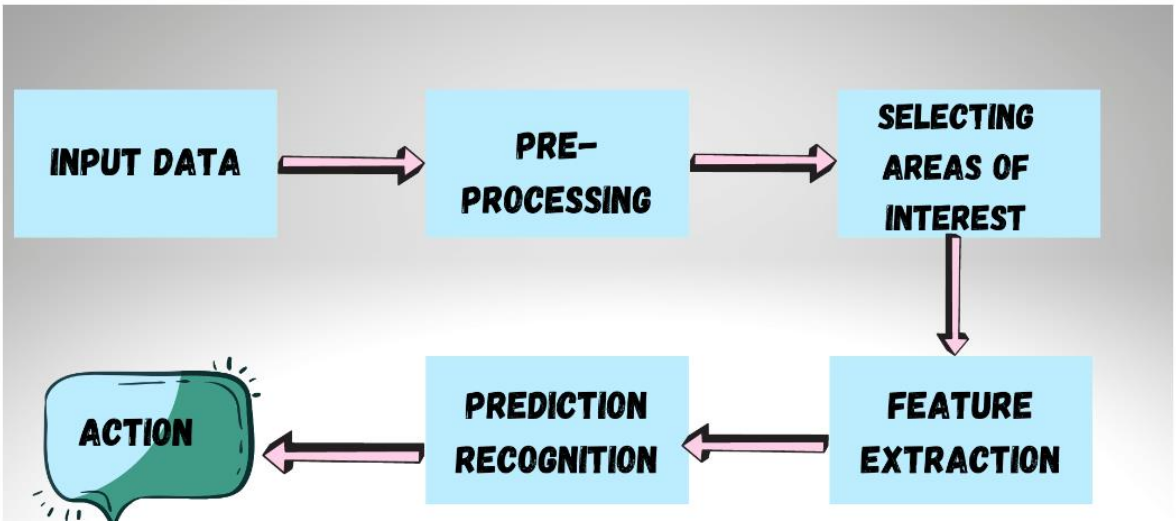
1. Transportation
   - Self Driving Cars, Pedestrian Detection, Traffic Flow Analysis
2. Healthcare
   - Xray Analysis, Cancer Detection, Movement Analysis, CT and MRI
3. Manufacturing
   - Reading Text and Barcodes, Defect Inspection
4. Agriculture
   - Aerial survey and imaging, livestock health monitoring,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

Computer Vision Pipeline/System

A pipeline is **a series of processes that migrate data from a source to a destination.**

1. Image acquisition/collection
   - Cameras, radar, ultra sonic cameras, etc
2. Preprocessing
   - Resampling to assure the image coordinate system, noise reduction, contrast enhancement, augmentation
3. Feature Extraction
   - Lines, edges, ridges, corners, blobs, etc
4. Detection/Segmentation/Different Operation
5. Decision Making
   - E.g. generating report of a medical diagnosis, automated vehicle changing its direction, etc.

The General Pipeline is pictured below:



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.

Region Based Image Operations in computer vision

Region-based image operations are a fundamental technique in computer vision that involves applying image processing operations to specific regions or regions of interest within an image, rather than the entire image.

These operations are commonly used for tasks such as image segmentation, object recognition, and tracking. The regions of interest can be defined in various ways, including manually by a user, using predefined templates, or by automatic segmentation algorithms.

Region-based image operations can be used to extract features from an image, such as color, texture, and shape, which can then be used for classification or recognition purposes. Examples of region-based operations include region growing, region splitting and merging, and region-based morphological operations.

Region-based image operations can be applied using various tools and libraries in computer vision, such as OpenCV and MATLAB. These tools provide a range of functions for performing region-based operations, including thresholding, filtering, and segmentation.

Pixel Based Image Operations in computer vision

Pixel-based image operations are a basic technique in computer vision that involve applying image processing operations to individual pixels within an image. These operations are commonly used for tasks such as noise reduction, contrast enhancement, and image sharpening.

Pixel-based image operations are typically performed using mathematical operations on the pixel values of the image. These operations can be used to modify the brightness, contrast, color balance, and other properties of the image.

Examples of pixel-based image operations include image thresholding, which involves converting an image to binary form based on a certain threshold value; edge detection, which involves identifying edges in an image by detecting changes in pixel values; and image filtering, which involves applying mathematical operations to pixel values to remove noise or enhance image features.

what are pixels in an image

In digital imaging, a pixel (short for "picture element") is the smallest unit of a digital image that can be displayed or manipulated. A digital image is made up of a grid of tiny squares called pixels, each of which represents a different color or shade of gray.

Difference between Region and Pixel based Image Operations

Region-based image operations and pixel-based image operations are two different approaches for manipulating digital images.

Pixel-based operations modify the value of individual pixels in the image, without regard to their surrounding neighbors. Examples of pixel-based operations include adjusting the brightness or contrast of an image, or applying a filter to remove noise. Pixel-based operations can be implemented efficiently using matrix operations, making them fast and suitable for real-time image processing.

Region-based operations, on the other hand, consider groups of neighboring pixels together as a region and apply an operation to that region. This approach allows for more complex manipulations, such as image segmentation or object detection. Region-based operations are computationally more complex than pixel-based operations, but they can often produce more visually pleasing results, especially for tasks such as image enhancement or restoration.

Difference between image segmentation and object detection

Image segmentation and object detection are two distinct computer vision tasks that involve analyzing images to identify the different objects or regions within them.

Image segmentation involves dividing an image into multiple regions or segments, where each segment represents a separate object or region in the image. The goal of image segmentation is to partition an image into meaningful regions, which can then be used for further analysis, such as object recognition or tracking. Image segmentation can be performed using a variety of techniques, including thresholding, clustering, and edge detection.

Object detection, on the other hand, involves identifying and localizing specific objects within an image. The goal of object detection is to detect and locate all instances of a particular object class within an image, such as cars, people, or animals. Object detection typically involves training a machine learning model on a large dataset of annotated images, where each image is labeled with the location and class of all objects present in the image.

what is image annotation

Image annotation is the process of labeling or tagging images with metadata that describes the objects or regions of interest within the image. Image annotation is an essential step in many computer vision applications, including object detection, image classification, and image segmentation.

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.

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.

What is Edge Detection

Edge detection is a fundamental technique in computer vision that involves identifying the boundaries or edges of objects in an image. The edges of an object represent the points where there is a significant change in the intensity or color of an image.

Edge detection algorithms typically analyze the changes in the gradient of an image, which refers to the rate at which the intensity or color of an image changes in different directions. The edges of an object correspond to the areas where the gradient is highest, indicating a sharp change in intensity or color.

There are many different edge detection algorithms, each with its strengths and weaknesses. Some of the most popular algorithms include the Canny edge detector, the Sobel operator, and the Laplacian of Gaussian (LoG) operator. These algorithms are often used in combination with other computer vision techniques, such as image segmentation and object recognition, to extract useful information from images.

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.

What is Thresholding in computer vision

Thresholding is a fundamental technique in computer vision that involves segmenting an image into regions based on the intensity or color of the pixels. The basic idea of thresholding is to separate the pixels of an image into two classes: foreground and background, based on a threshold value.

The threshold value is chosen such that all pixels below the threshold are considered part of the background, while all pixels above the threshold are considered part of the foreground. This technique is often used to remove noise or to extract regions of interest in an image.

What is noise in an image

In digital image processing, noise refers to unwanted variations or distortions in the image that are not part of the original scene being captured. Noise can be introduced at various stages of the image acquisition and processing pipeline, such as during image capture, transmission, or processing.

What is Erosion in computer vision

In computer vision, erosion is a morphological operation that is used to reduce the size of the foreground objects in an image, while at the same time increasing the size of the background objects.

Erosion works by sliding a structuring element (a small matrix of 0s and 1s) over the image pixel by pixel, and checking whether the pixels covered by the structuring element are all white or not. If all the pixels covered by the structuring element are white, then the center pixel is also set to white; otherwise, it is set to black.

The size and shape of the structuring element used for erosion determine the amount of erosion applied to the image. A larger structuring element will cause more erosion, while a smaller structuring element will cause less erosion.

Erosion can be useful in various image processing tasks such as noise reduction, segmentation, and feature extraction. For example, it can be used to remove small details in an image or to separate overlapping objects in an image. However, erosion can also result in the loss of useful information in the image, and care should be taken when applying this operation to avoid over-eroding the image.

What is Dilation in computer vision

In computer vision, dilation is a morphological operation that is used to expand the size of the foreground objects in an image, while at the same time reducing the size of the background objects.

Dilation works by sliding a structuring element (a small matrix of 0s and 1s) over the image pixel by pixel, and checking whether any of the pixels covered by the structuring element are white or not. If any of the pixels covered by the structuring element are white, then the center pixel is also set to white; otherwise, it is set to black.

The size and shape of the structuring element used for dilation determine the amount of dilation applied to the image. A larger structuring element will cause more dilation, while a smaller structuring element will cause less dilation.

Dilation can be useful in various image processing tasks such as noise reduction, segmentation, and feature extraction. For example, it can be used to fill in small gaps or holes in an image, or to connect broken regions of an image. However, dilation can also result in the over-growth of objects in the image, and care should be taken when applying this operation to avoid over-dilating the image.

What is bitwise operation in computer vision in simple language and in 1 paragraph

In computer vision, bitwise operations are used to perform logical operations on binary images. A binary image consists of pixels with values of either 0 or 1, where 0 usually represents the background and 1 represents the foreground or the object of interest. Bitwise operations involve comparing the binary values of the corresponding pixels in two binary images, and producing a new binary image based on the result of the comparison. There are four types of bitwise operations: AND, OR, XOR, and NOT. The AND operation produces a new binary image where the output pixel is 1 only if both corresponding pixels in the input images are also 1. The OR operation produces a new binary image where the output pixel is 1 if at least one of the corresponding pixels in the input images is also 1. The XOR operation produces a new binary image where the output pixel is 1 only if the corresponding pixels in the input images have different values. The NOT operation produces a new binary image where the output pixel is the complement of the input pixel. These operations are used in various computer vision applications such as image filtering, object detection, and segmentation to manipulate binary images and extract useful information from them.

What is gray scale image

A grayscale image is an image in which the color or intensity of each pixel is represented by a single value, typically ranging from 0 (black) to 255 (white) in an 8-bit image. A grayscale image has only shades of gray, without any color information. The value of each pixel in a grayscale image represents the brightness or luminance of the corresponding point in the image.

A grayscale image can be represented as a matrix where each element of the matrix corresponds to a pixel in the image, and the value of the element represents the intensity of the pixel. The dimensions of the matrix are typically the same as the dimensions of the image. Grayscale images are commonly used in computer vision tasks such as image processing, image analysis, and computer graphics, as they are simpler to work with than color images and contain sufficient information for many applications.

what is Rgb image

An RGB (Red, Green, Blue) image is a digital image in which the color of each pixel is represented by a combination of red, green, and blue values. In an 8-bit RGB image, each color channel has 256 possible values, which are represented by integers from 0 to 255. The color of each pixel in an RGB image is determined by the combination of these three values. By varying the intensities of red, green, and blue, a wide range of colors can be generated. RGB images are commonly used in computer vision, image processing, and computer graphics applications, and are the most common type of color image used in digital photography and display. They can be represented as a three-dimensional matrix, where the first dimension represents the rows of the image, the second dimension represents the columns, and the third dimension represents the three color channels.

what is Hsv image

HSV stands for Hue, Saturation, and Value, which are three components used to represent colors in an HSV color space. An HSV image is a color image representation where each pixel's color is defined using these three values.

The Hue component describes the dominant color of the pixel, ranging from 0 to 360 degrees, where 0 and 360 degrees correspond to red, 120 degrees corresponds to green, and 240 degrees corresponds to blue.

The Saturation component describes the intensity or purity of the color, ranging from 0 to 100 percent. A saturation of 0 corresponds to a grayscale image, while a saturation of 100 percent corresponds to a fully saturated color.

The Value component describes the brightness or luminance of the color, ranging from 0 to 100 percent. A value of 0 corresponds to black, while a value of 100 percent corresponds to white.

HSV is a useful color space in computer vision applications such as image processing, object detection, and segmentation. Unlike RGB, HSV separates the color information from the brightness information, making it easier to manipulate color without affecting brightness and vice versa.

What are color channels

Color channels refer to the individual color components that make up a color image. In a color image, each pixel is composed of multiple color values, usually three in the case of an RGB image (red, green, and blue). Each of these color values is known as a color channel.

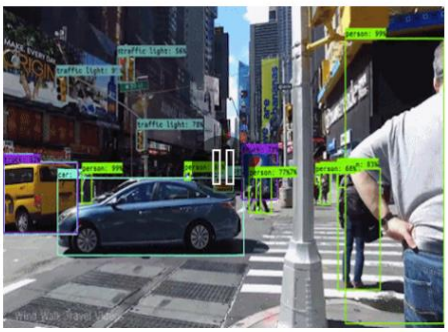
In an RGB image, each pixel is composed of three color channels, each representing the intensity of red, green, and blue, respectively. These three channels combine to create a full-color image, with each pixel's color determined by the combination of the three channel values.

In other color spaces such as CMYK or HSV, there can be different numbers of color channels used to represent an image. For example, in a CMYK image, each pixel is composed of four color channels, representing the intensity of cyan, magenta, yellow, and black.

PPT 4 5 – DL

Deep Learning Phase Outline

1. What do CNNs Learn ?
2. One Shot Learning (Face Recognition)
    1. Siamese Network
3. Image Segmentation
    1. FCN
    2. U-Net (Optional)
4. Object Detection (Architecture details and training framework)
    1. RCNN
    2. Fast RCNN
    3. Faster RCNN
    4. YOLO



What is ANN

ANN stands for Artificial Neural Network. It is a type of machine learning algorithm that is modeled after the structure and function of the human brain.

ANNs consist of interconnected processing nodes, which are organized into layers. Each node performs a simple mathematical operation on its inputs and passes the result to the next layer. The nodes in the output layer produce the final result of the network's computation.

ANNs are trained using a process called backpropagation, which adjusts the connections between nodes to minimize the difference between the network's output and the desired output for a given input. Once trained, ANNs can be used for a variety of tasks, including pattern recognition, classification, regression, and forecasting.

ANNs have been used successfully in a wide range of applications, including image recognition, natural language processing, and game playing.

What is the Use of ANN

Artificial Neural Networks (ANNs) have a wide range of applications in many fields, including:

1. Pattern recognition: ANNs can be used to recognize patterns in images, speech, and text. For example, ANNs can be used to classify images of handwritten digits, recognize faces, or transcribe speech to text.
2. Prediction and forecasting: ANNs can be used to predict future values based on historical data. For example, ANNs can be used to forecast stock prices, weather patterns, or traffic flow.
3. Control systems: ANNs can be used to control systems that have complex nonlinear dynamics. For example, ANNs can be used to control robotic arms, autonomous vehicles, or power grids.
4. Natural language processing: ANNs can be used to analyze and understand natural language text. For example, ANNs can be used for sentiment analysis, language translation, or question-answering systems.
5. Optimization: ANNs can be used to solve optimization problems. For example, ANNs can be used to optimize the design of products, minimize costs in manufacturing processes, or optimize energy consumption in buildings.

How does ANN work

Artificial Neural Networks (ANNs) work by simulating the behavior of the human brain. ANNs consist of interconnected processing nodes, which are organized into layers. There are typically three types of layers in an ANN:

1. Input layer: The input layer receives input data from the outside world. For example, if the ANN is being used to recognize handwritten digits, the input layer would receive an image of a handwritten digit.
2. Hidden layers: The hidden layers process the input data and perform calculations. There can be one or more hidden layers in an ANN.
3. Output layer: The output layer produces the final result of the network's computation. For example, if the ANN is being used to recognize handwritten digits, the output layer would produce a probability distribution over the 10 possible digits.

The nodes in each layer are interconnected by weighted connections. Each node in a hidden layer takes the weighted sum of its inputs, applies an activation function to the result, and passes the output to the nodes in the next layer. The weights on the connections between nodes are adjusted during training to minimize the difference between the network's output and the desired output for a given input.

Nodes in ANNs are also referred to as neurons or processing units. Each node performs a simple mathematical operation on its inputs, which typically involves multiplying the input by a weight, summing the results, and passing the output through an activation function. The weights on the connections between nodes are adjusted during training to minimize the difference between the network's output and the desired output for a given input.

Overall, the layers and nodes in ANNs work together to process input data and produce output that can be used for a wide range of applications, such as pattern recognition, prediction, and control.

What are weights and biases in ANN

Weights and biases are important components of Artificial Neural Networks (ANNs). They are used to model the behavior of neurons and their interactions in the network.

Weights are parameters that determine the strength of the connections between neurons. Each connection between neurons is assigned a weight, which represents the strength of the connection. During training, the weights are adjusted to minimize the difference between the network's output and the desired output for a given input. The weights are initialized randomly before training and are updated using an optimization algorithm, such as gradient descent or backpropagation.

Biases are parameters that determine the behavior of individual neurons in the network. A bias is added to the weighted sum of the inputs to a neuron before the activation function is applied. Biases are used to control the output of a neuron when all of its inputs are zero. Like weights, biases are also initialized randomly and are updated during training.

Whats the use of activation function in ANN

The activation function is a critical component of Artificial Neural Networks (ANNs). It is used to introduce non-linearity to the output of a neuron by transforming the weighted sum of the inputs to a neuron into the neuron's output.

The activation function is applied to the output of each neuron in the network. Without an activation function, the output of the neuron would be a simple linear function of the weighted sum of the inputs. A linear function can only model linear relationships between the inputs and outputs, which is too limiting for many real-world problems. Non-linear activation functions, on the other hand, enable the network to model more complex relationships between the inputs and outputs.

Forward propagation is the process of computing the output of a neural network for a given input. It involves passing the input forward through the network, layer by layer, and computing the output of each layer using the weights, biases, and activation function of the neurons in that layer. The output of one layer becomes the input to the next layer until the output of the final layer is computed, which is the output of the neural network for the given input.

The forward propagation algorithm can be broken down into the following steps:

1. The input is fed into the input layer of the network.
2. The input is multiplied by the weights of the connections between the input layer and the first hidden layer.
3. The bias for each neuron in the first hidden layer is added to the weighted sum of the inputs.
4. The output of each neuron in the first hidden layer is computed by applying the activation function to the sum of the weighted inputs and bias.
5. Steps 2-4 are repeated for each hidden layer until the output layer is reached.
6. The output of the final layer is computed by applying the activation function to the weighted sum of the inputs and bias of the output layer.
7. The output of the final layer is the output of the neural network for the given input.

---

what is backward propagation

Backpropagation (short for "backward propagation of errors") is an algorithm used to train artificial neural networks (ANNs) by computing the gradient of the loss function with respect to the weights of the network. It is a key component of most neural network training algorithms, including gradient descent.

The backpropagation algorithm works by computing the gradient of the loss function with respect to the weights of the network, starting at the output layer and working backwards through the network. The gradient is then used to adjust the weights in the opposite direction of the gradient, which minimizes the loss function.

What is gradient descent

Gradient descent is an optimization algorithm used to minimize the loss function of a neural network. In the context of neural networks, the loss function measures the difference between the predicted output of the network and the true output for a given input. The goal of training a neural network is to minimize the value of the loss function over the entire training set.

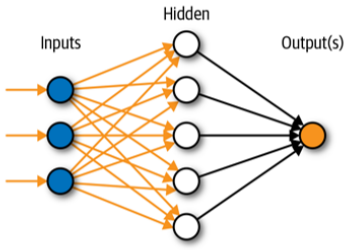
The gradient descent algorithm works by iteratively adjusting the weights of the network in the direction of the negative gradient of the loss function with respect to the weights. The negative gradient is used because we want to minimize the loss function, and the gradient tells us which direction the function is decreasing.

The learning rate is an important hyperparameter in the gradient descent algorithm, as it determines how quickly the algorithm will converge to the minimum of the loss function. If the learning rate is too large, the algorithm may overshoot the minimum and fail to converge. If the learning rate is too small, the algorithm may take a long time to converge, or may get stuck in a local minimum.

In summary, gradient descent is an optimization algorithm used to minimize the loss function of a neural network by iteratively adjusting the weights of the network in the direction of the negative gradient of the loss function with respect to the weights. It is a key component of most neural network training algorithms, including backpropagation.

A revisit to an old friend: NNs

1. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain
2. Used for non linear pattern recognition
3. Always count the hidden layers + output layers
   - This is a 2 layers neural network
4. Hyperparameters need to be configured:-
   1. No. of layers
   2. No. of neurons in each layer
   3. No. of epochs
   4. Learning rate
   5. Loss function, Optimizer
   6. Batch size
5. How do you count the total parameters in your neural network ?



What are parameters in a Neural Network

In a neural network, the parameters are the variables that the network learns from the training data. The two main types of parameters in a neural network are the weights and biases.

1. Weights: The weights are the parameters that determine the strength of the connections between the neurons in the network. Each connection between two neurons has a weight associated with it, which determines how strongly the output of one neuron influences the input of the other neuron. The weights are typically initialized with small random values and are adjusted during training using an optimization algorithm such as gradient descent.
2. Biases: The biases are the parameters that determine how easy it is for a neuron to fire. They are added to the input of each neuron and allow the network to learn a more complex decision boundary. The biases are also typically initialized with small random values and are adjusted during training using an optimization algorithm such as gradient descent.

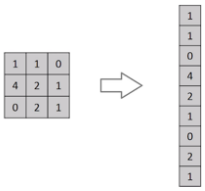
A revisit to an old friend: NNs

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

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

A visit to a new friend: CNNs

Remember Convolution operation ?

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

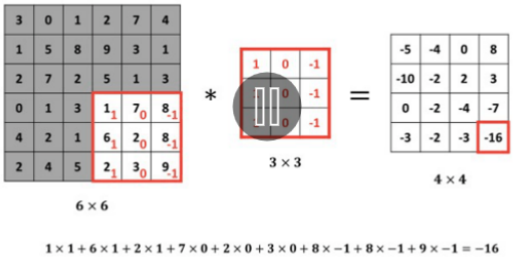
How to decide weights of kernels ?

Make the kernels learnable

Treat the kernels as trainable weights

What about nonlinear patterns in images ?

Add activation functions to the kernels



$$1\times1+6\times1+2\times1+7\times0+2\times0+3\times0+8\times-1+8\times-1+9\times-1=-16$$

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.

A CNN works by using a series of convolutional layers, pooling layers, and fully connected layers to extract and classify features from input data, typically images. Here is a brief overview of how a CNN works:

1. Input data: The input data is usually a set of images that are represented as arrays of pixel values.
2. Convolutional layer: The first layer in a CNN applies a set of filters (also called kernels or weights) to the input data. Each filter convolves over the input image, producing a feature map that highlights a specific pattern in the image.
3. ReLU activation: After each convolutional layer, a Rectified Linear Unit (ReLU) activation function is applied to the output of the layer. ReLU sets all negative values in the feature map to zero, which helps to increase the non-linearity of the network.
4. Pooling layer: A pooling layer is used to downsample the feature maps generated by the convolutional layers, reducing the size of the data and making it more manageable. Common pooling functions include max pooling and average pooling.
5. Repeat: Steps 2-4 are repeated with additional convolutional and pooling layers, each layer extracting increasingly complex features from the input image.
6. Fully connected layer: The final layers of a CNN are fully connected layers, which take the flattened output of the convolutional and pooling layers and use it to make predictions about the input image.
7. Output: The output of the fully connected layer is usually passed through a softmax activation function to generate a probability distribution over the possible output classes.

During the training phase, the weights of the CNN are adjusted using backpropagation to minimize the error between the predicted output and the actual output. This process is repeated for many iterations until the network is able to accurately classify the input data.

Here's how convolutional layers work in more detail:

1. Filter initialization: The weights of the filters are initialized randomly.
2. Convolution operation: Each filter convolves over the input image, producing a feature map that highlights a specific pattern in the image. The convolution operation involves taking the dot product between the filter and a small region of the input image at a time, sliding the filter over the entire input image to produce the feature map.
3. Non-linear activation: After the convolution operation, a non-linear activation function is applied to the feature map to introduce non-linearity into the network. The most common activation function used in CNNs is the Rectified Linear Unit (ReLU) function.
4. Repeated convolution and activation: This process of applying a filter to the input image, followed by a non-linear activation, is repeated with multiple filters to produce multiple feature maps that capture different patterns in the image.
5. Pooling: Once the feature maps have been produced, they are often fed through a pooling layer, which downsamples the feature maps to reduce the computational complexity of the network. Common pooling functions include max pooling and average pooling.

What are feature Maps

A feature map, also called an activation map, is a representation of the output of a convolutional layer in a Convolutional Neural Network (CNN). Each feature map corresponds to a specific filter applied to the input data, and the values in the map represent the degree of activation of that filter across the input data.

Feature maps are created by convolving a filter with the input data and applying a non-linear activation function such as the Rectified Linear Unit (ReLU) to the result. This produces a set of values that indicate the presence or absence of a particular feature in the input data.

CNNs typically use multiple filters in each convolutional layer, resulting in multiple feature maps for each layer. Each feature map captures a different aspect of the input data, allowing the network to learn increasingly complex features as information flows through the network.

Pooling layers are an essential component of Convolutional Neural Networks (CNNs), and they serve two main purposes:

1. 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.
2. Introduce spatial invariance: Pooling layers introduce a degree of spatial invariance into the network by making the output of the previous layer more robust to small translations or distortions in the input data. This is achieved by taking the maximum or average value over a small window of the input data, which effectively summarizes the features present in that region.

The two most common types of pooling layers are max pooling and average pooling:

- 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.

Whats the use of pooling layers in simple language

Pooling layers are used in Convolutional Neural Networks (CNNs) to reduce the size of the input data and introduce a degree of spatial invariance to the network. They do this by summarizing small regions of the input data using techniques such as taking the maximum or average value. This reduces the number of parameters in the network and helps the network to recognize features regardless of their position in the input data. Overall, pooling layers make CNNs more effective for image processing tasks like object recognition and segmentation.

---

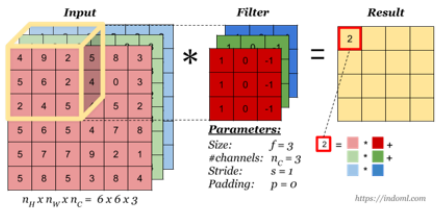
spatial invariance

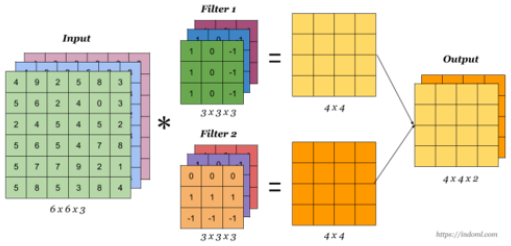
Spatial invariance refers to the property of a system that remains unchanged under spatial transformations such as translation, rotation, scaling, or distortion. In the context of image processing, spatial invariance means that a system can recognize objects or features in an image regardless of their position or orientation in the image. This is an important property for Convolutional Neural Networks (CNNs) because it allows the network to learn features that are invariant to small changes in the position or orientation of an object, making the network more robust to variations in the input data. Pooling layers in CNNs help to introduce spatial invariance by summarizing small regions of the input data and reducing the impact of small spatial transformations on the output of the network.

Convolutions over Volumes

Convolutions on RGB image/over volumes

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





CNNs: Understanding Dimensions

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

CNNs: Understanding Dimensions

Lets 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

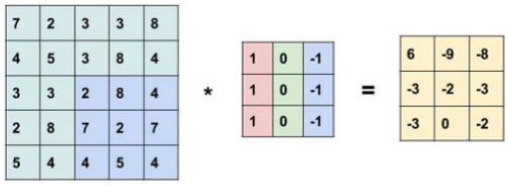
W2: W1 – F + 1 -> 5 – 3 + 1 = 3

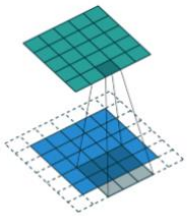
H2: H1 – F + 1 -> 5 – 3 + 1 = 3

Lets say we have a 7x7 image and kernel of size 3x3

7 – 3 + 1 = 5

Note: The output result dimensions are smaller than input dimensions





CNNs: Understanding Dimensions

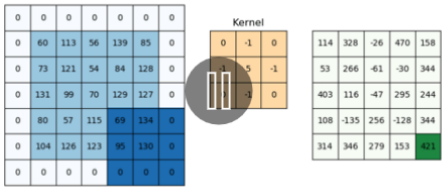
What if we want the output of convolution to be of same size as input ?

Pad input image with appropriate number of inputs so that you can now apply the kernel on input image corners as well. There are different methods for padding but the most commonly used one is zero padding. No. of Padding units (P)

If P is set to 1 then it will pad rows and columns of 0's to the top, bottom, left and right of the image

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

5 – 3 + 2x1 + 1 = 5



what is 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."

Padding is typically used to preserve the spatial dimensions of the input data after convolutional layers. Without padding, the dimensions of the output feature maps will be smaller than the input feature maps, which can lead to information loss at the borders. By adding border pixels, the output feature maps will have the same dimensions as the input feature maps.

Whats the use of strides

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

The stride determines the number of pixels by which the filter is shifted along each dimension of the input data. For example, a stride of 1 means that the filter is shifted one pixel at a time, while a stride of 2 means that the filter is shifted two pixels at a time.

The main use of strides is to reduce the spatial dimensions of the output feature maps. When the stride is greater than 1, the output feature maps will be smaller than the input feature maps. This can help to reduce the number of parameters in the network and improve its efficiency.

However, using larger strides can also result in information loss, since the filter is not applied to all of the pixels in the input feature maps. In addition, larger strides can make it more difficult for the network to learn complex features.

Strides are often used in combination with other techniques such as padding and pooling to achieve the desired balance between spatial dimensions and feature learning.

Regenerate response

CNNs: Understanding Dimensions

What does stride do ?

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

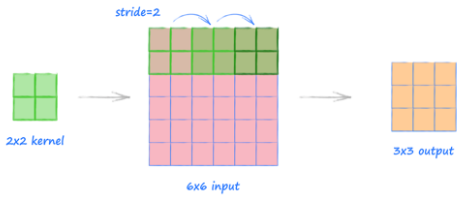
Results in an output of smaller dimensions

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



CNNs: Understanding Dimensions

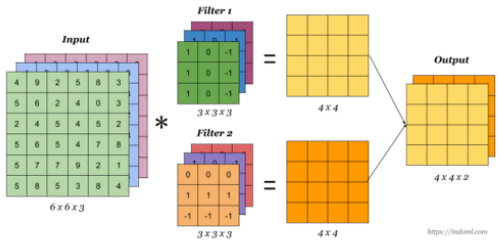
Lets talk about the depth of the output now.

Each kernel/filter gives us a 2D output

If we use K filters then we will get K such 2D outputs

The resulting dimensions would then be W2xH2xK

Thus, D2 = K



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

Artificial Neural Networks (ANNs), particularly Fully Connected Neural Networks (FNNs), faced several challenges in image tasks. These include:

1. High-dimensional input data: Images typically have a large number of pixels, resulting in high-dimensional input data. FNNs have difficulty processing such high-dimensional input data efficiently because of the sheer number of parameters required to learn the features in the image.
2. Spatial structure: Images have spatial structure, where neighboring pixels have a correlation with each other. FNNs do not take into account the spatial structure of the image, making them inefficient in learning features that are dependent on the spatial context of the input.
3. Translation invariance: FNNs are not translation invariant, meaning that if an object in an image is shifted or rotated, the network may not recognize it as the same object. This is because FNNs are sensitive to the exact position of the features in the image.
4. Overfitting: FNNs are prone to overfitting, particularly when dealing with high-dimensional input data such as images. Overfitting occurs when the network learns the noise in the training data rather than the underlying patterns. This can result in poor performance on unseen data.

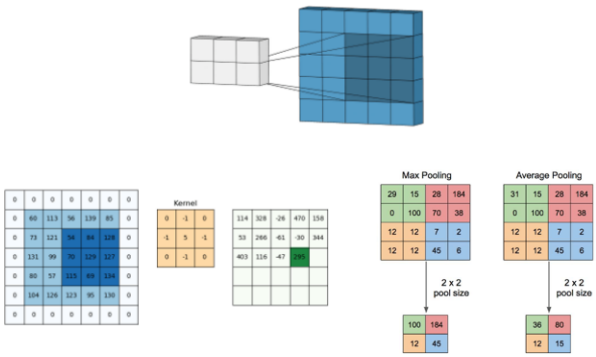
What were the challenges faced by ANNs aka FNNs in image tasks ?

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 obvious way for network to learn features at different places in an input image (similar edges, corners or blobs)
3. Can get computationally expensive for large images

How do CNNs solve these challenges ?

1. Local Receptive Fields: Hidden units are connected to the local patches from the previous layer. This serves 2 main purposes:-
   - Captures local spatial relationships in pixels
   - Reduces the number of parameters significantly
2. Weight Sharing: Also serves 2 purposes
   - Enables rotation, scale and translational invariance to objects in images (no more usage of the painful SIFT algo)
   - Reduces number of parameters in the model
3. 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



What is Local Receptive fields in CNN

Local receptive fields in a Convolutional Neural Network (CNN) refer to the regions of the input data that are used by the convolutional filter to compute the output feature map.

Each neuron in a convolutional layer is connected to a small, contiguous region of neurons in the previous layer. This region is called the local receptive field, and it corresponds to the area of the input data that is used to compute the output value of the neuron.

Local receptive fields are important in CNNs because they allow the network to capture local patterns in the input data. By focusing on small, local regions of the input data, the network is able to learn features that are more specific and localized.

In addition, local receptive fields help to reduce the number of parameters in the network. By sharing weights across the local receptive fields of different neurons, the network can learn a smaller number of parameters that are reused across different regions of the input data. This can greatly reduce the computational complexity of the network and make it more efficient.

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.

What a typical CNN architecture looks like ?

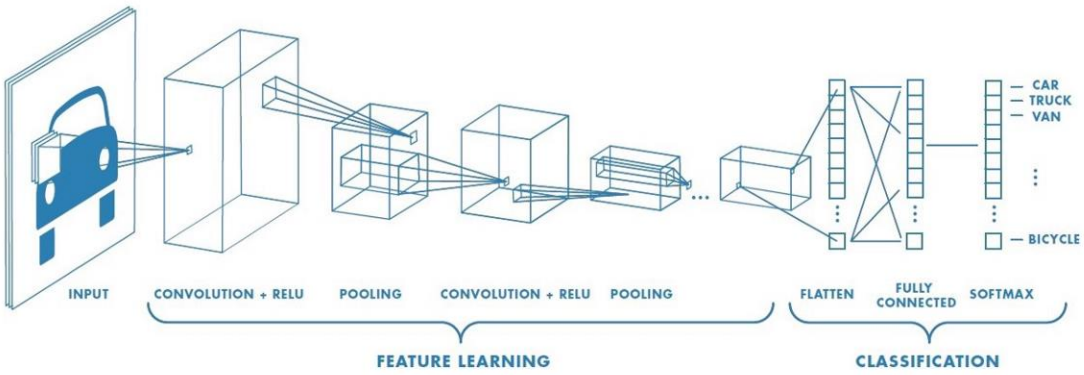
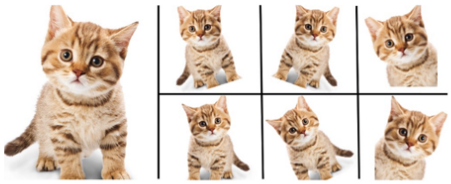


Image Augmentation

Image augmentation is **a technique that is used to artificially expand the data-set**. This is helpful when we are given a data-set with very few data samples. In case of Deep Learning, this situation is bad as the model tends to over-fit when we train it on limited number of data samples.

We use both image filtering and image transformations (affine transformations) to expand the dataset.

Since we know that the weight sharing property of the CNNs make the network rotation, scale and translational invariant we can easily leverage these transformation methods.



Enlarge your Dataset

What is Image Augmentation

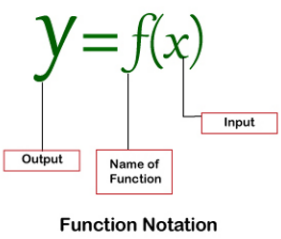
Image augmentation is a technique used to artificially expand a dataset of images by applying various transformations to existing images. The goal of image augmentation is to increase the diversity of the training data, which can help improve the performance and generalization of machine learning models.

Image augmentation techniques can include:

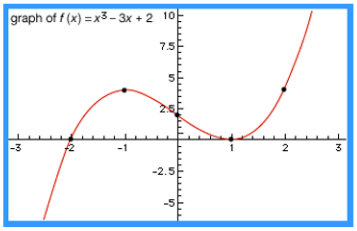
1. Random cropping: randomly selecting a subregion of the image and resizing it to the original size.
2. Rotation: rotating the image by a certain angle, which can help the model learn to recognize objects from different orientations.
3. Flipping: horizontally flipping the image, which can help the model learn to recognize objects from different perspectives.
4. Color jittering: randomly adjusting the brightness, contrast, and saturation of the image to create variations in lighting conditions.
5. Translation: shifting the image in different directions to simulate different positions of the object.
6. Adding noise: adding random noise to the image to simulate natural variation in real-world images.

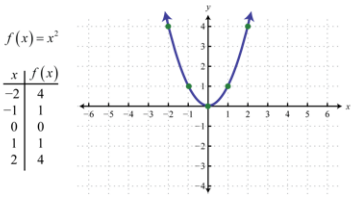
Regenerate response

Neural Networks as Functions



Function Notation





Note: Neural Networks are nothing but trainable parametric non linear functions. The complexity of the function depends on the number of layers of the neural network

Overfitting in Neural Nets

What happens when a function is overtrained on same instances of data ? It overfits on those values and fails to generalize on new instances of data.

Solution: -

1. Regularization
    1. L1 and L2, ElasticNet
    2. Dropout
2. Addressing the difference in data distributions
3. Adding more data (diverse kind not the repeated instances)
4. Stop being greedy and stop the training a little earlier

Overfitting in Neural Networks

Overfitting is a common problem in neural networks that occurs when a model becomes too complex and starts to fit the noise in the training data, rather than the underlying patterns. This leads to poor generalization performance, where the model performs well on the training data but poorly on new, unseen data.

There are several techniques to prevent overfitting in neural networks:

1. Early stopping: This involves monitoring the validation loss during training and stopping the training when the validation loss stops improving. This prevents the model from overfitting to the training data by stopping the training before it has a chance to overfit.
2. Regularization: Regularization techniques such as L1, L2, or dropout can be used to prevent overfitting. L1 and L2 regularization add a penalty term to the loss function, which encourages the model to have smaller weights. Dropout randomly drops out units during training, which helps prevent the model from relying too heavily on any one unit.
3. Data augmentation: Data augmentation involves generating new training data by applying transformations such as rotations, flips, and shifts to the existing training data. This increases the size and diversity of the training data, which can help prevent overfitting.
4. Model simplification: Simplifying the model architecture can also help prevent overfitting. This involves reducing the number of layers, the number of units per layer, or using a smaller model altogether.
5. Increase training data: Overfitting often occurs when there is not enough data to learn the underlying patterns. Increasing the amount of training data can help prevent overfitting by providing the model with more examples to learn from.

L1 regularization, also known as Lasso regularization, is a technique used in machine learning to reduce overfitting of a model by adding a penalty term to the loss function.

The penalty term is proportional to the sum of the absolute values of the model's parameters, which encourages the model to have many small or zero-weight parameters. This leads to a sparse model where many of the model's parameters are set to zero, which can improve the model's interpretability and reduce the risk of overfitting.

Mathematically, the L1 regularization term is expressed as:

$L1 = \lambda * \sum|w|$

where λ is the regularization strength hyperparameter, w is a model parameter, and the sum is taken over all parameters. The L1 term is added to the loss function, and the model is trained to minimize the total loss.

L1 regularization can be used in a variety of models, including linear regression, logistic regression, and neural networks. It is particularly useful when dealing with high-dimensional datasets, where the number of features is much larger than the number of examples. In such cases, L1 regularization can help identify the most important features and remove noise from the data.

L2 regularization, also known as Ridge regularization, is a technique used in machine learning to reduce overfitting of a model by adding a penalty term to the loss function.

The penalty term is proportional to the sum of the squares of the model's parameters, which encourages the model to have small weights. This can improve the generalization performance of the model by reducing the complexity of the model and avoiding large weights that may lead to overfitting.

Mathematically, the L2 regularization term is expressed as:

$L2 = \lambda * \sum w^2$

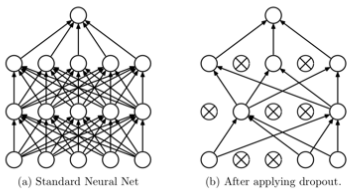
where λ is the regularization strength hyperparameter, w is a model parameter, and the sum is taken over all parameters. The L2 term is added to the loss function, and the model is trained to minimize the total loss.

L2 regularization can be used in a variety of models, including linear regression, logistic regression, and neural networks. It is particularly useful when dealing with datasets with highly correlated features or when the number of features is much larger than the number of examples. L2 regularization can also help prevent numerical instability by ensuring that the matrix used to solve the linear equations in the model is not ill-conditioned.

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 of 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.



(a) Standard Neural Net (b) After applying dropout.

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.

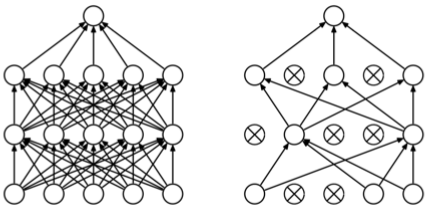
Co-adaptation, also known as co-adaptive overfitting, is a problem that can occur in neural networks when multiple neurons learn to work together to overfit the training data. In other words, co-adaptation occurs when different subsets of neurons become too specialized in memorizing specific aspects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

Dropout During Inference (Test/Val)

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.



(a) Standard Neural Net (b) After applying dropout.

What is feature scaling?

Feature scaling is a preprocessing technique used in machine learning to transform features of a dataset to a common scale.

When working with datasets where the features have different ranges of values, algorithms that use distance measures, such as k-nearest neighbors or support vector machines, can be sensitive to the scale of the input features. This can lead to biased results, where some features are given more weight than others due to their larger scale.

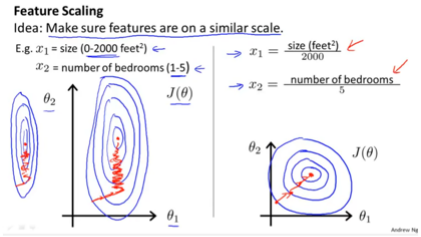
To address this issue, feature scaling is used to normalize the range of values for each feature so that they all have similar scales. This can improve the performance and accuracy of machine learning models by ensuring that each feature contributes equally to the model.

There are two common types of feature scaling:

1. Standardization: In this method, the mean of the feature is subtracted from each data point, and then the result is divided by the standard deviation of the feature. This scales the data to have a mean of 0 and a standard deviation of 1.
2. Normalization: In this method, each data point is scaled to a range between 0 and 1, based on the minimum and maximum values of the feature in the dataset.

Why Feature Scaling is Important ?

We need **input feature scaling** in parametric methods like neural networks because the parameters descend quickly on smaller homogeneous scales and slowly on larger heterogeneous scales.



But what about the scales of features of the hidden layers of the neural network ?

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.

SR

Why is batch normalisation used

Batch normalization is used in deep neural networks for several reasons:

1. Faster training: Batch normalization can help to speed up the training process of neural networks by reducing the internal covariate shift, which is the change in the distribution of activations in each layer during training. By normalizing the inputs to each layer, batch normalization can help to stabilize the optimization process and prevent the vanishing and exploding gradient problems that can occur in deep neural networks, allowing for faster and more efficient training.
2. Improved performance: Batch normalization can also improve the performance of deep neural networks by reducing overfitting and improving generalization. By reducing the internal covariate shift, batch normalization can make the optimization process more stable and robust, allowing for better generalization and reducing the risk of overfitting.
3. Better initialization: Batch normalization can also improve the initialization of deep neural networks. By normalizing the inputs to each layer, batch normalization can help to ensure that the initial weights of the network are not biased towards a particular activation distribution, making it easier for the optimization process to converge.
4. Handling varying input distributions: Batch normalization can also help to handle varying input distributions in neural networks. By normalizing the inputs to each layer, batch normalization can ensure that the network is not sensitive to changes in the input distribution, making it more robust and adaptable to different types of data.

Explain batch normalisation in simple language

Batch normalization is a technique used in deep learning to help neural networks learn better and faster.

In a neural network, each layer takes inputs from the previous layer and applies some calculations to produce outputs that are passed to the next layer. However, during training, the distribution of these inputs can change, which makes it harder for the network to learn and can slow down the training process.

Batch normalization addresses this problem by making sure that the inputs to each layer have a standard distribution, which means the network can learn faster and more effectively. It does this by calculating the mean and variance of the inputs over a small batch of data, and then adjusting the inputs so that they have a mean of zero and a standard deviation of one. This helps to ensure that the inputs to each layer are not too different from each other, which can make it easier for the network to learn and can improve its accuracy.

Overall, batch normalization is a powerful technique that can make deep learning models more effective and efficient. It helps to ensure that the network can learn more quickly and accurately, which can lead to better performance on a wide range of tasks.

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.

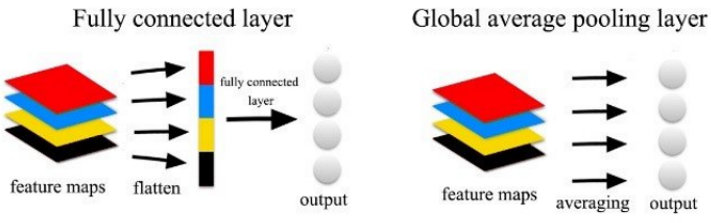
Batch normalization has been shown to be an effective technique for improving the performance of deep neural networks, and it has become a popular technique for training deep learning models. It can help to speed up the training process, improve the accuracy of the model, and make the optimization process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

Global Average Pooling

Reduces the increase in number of params caused by flattening layer

Takes an average of all the feature maps of the layer

E.g. if the last conv layer in the network has 512 filters thus producing 512 feature maps, the global average pooling will produce an output vector of 512



Global Average Pooling (GAP) is a pooling operation used in Convolutional Neural Networks (CNNs) for image classification tasks. It is commonly used as the last layer of a CNN, where it takes the output feature map of the last convolutional layer and reduces its spatial dimensions to a single value per channel. Here is a more detailed explanation of the GAP operation:

1. Input feature map: The input to the GAP layer is the output feature map of the last convolutional layer in the CNN. The feature map is a tensor of shape (H, W, C), where H and W are the height and width of the feature map, and C is the number of channels.
2. Global Average Pooling: The GAP operation computes the average value for each channel across all spatial locations of the feature map. This is done by summing the values of each channel and dividing by the total number of spatial locations. The resulting feature vector is of shape (C,), where C is the number of channels.
3. Output: The output of the GAP layer is a feature vector of shape (C,), where C is the number of channels. Each element of the feature vector represents the global average of the activations over the entire feature map for that channel.

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.

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 has several advantages, including:

1. Reduced training time: Using a pre-trained model as a starting point can significantly reduce the amount of time needed to train a new model, especially when the new dataset is small.
2. Improved performance: Pre-trained models have already learned useful features from a large dataset, which can help to improve the performance of a new model on a smaller dataset.
3. Better generalization: Pre-trained models have learned to recognize and extract features from a wide range of images, which can help to improve the generalization of a new model to new images that it has not seen before.
4. Reduced need for data: Transfer Learning can be effective even when the new dataset is small, as it leverages the pre-trained model's features to learn from fewer examples.

Fine-tuning involves the following steps:

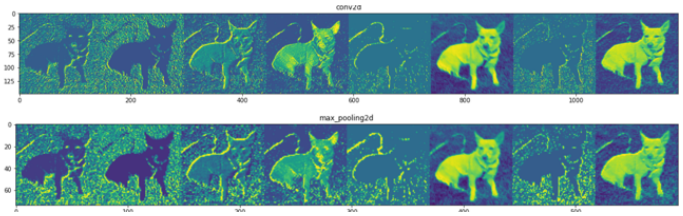
1. Select a Pre-trained Model: The first step in fine-tuning is to select a pre-trained model that is relevant to the new task. The pre-trained model must be trained on a large, general dataset and should have similar features to the new task. For example, a pre-trained model that is trained on the ImageNet 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.

Transfer Learning

1. Transfer Learning is the transfer of the knowledge (feature maps) that the network has acquired from one task, where we have large amount of data, to a new task where the data is not abundantly available.
2. Generally used where a neural network model is trained on a problem similar to the problem that is being solved.
3. The intuition behind transfer learning is that if a model is trained on a large and general enough dataset, the model will effectively serve as a generic function for that problem.
4. We use the feature maps that this model has learnt, and without having to train a new model on a large dataset we can transfer the learning of the trained model's knowledge to our model and use it as a base starting model for our own task

How Transfer Learning works ?

1. What is really being learned by a CNN during training ? Feature Maps
2. How are these features learned ? During backpropagation process the weights are updated until we get to the optimized weights that minimize the error function
3. What is the relationship between features and weights ? Feature maps are a result of passing the weights filter/kernel on the input image during the convolution process
4. What is really being transferred from one network to another ? To transfer features we use the optimized weights of the pretrained network and reuse them as the starting point for the training process to adapt to our new problem

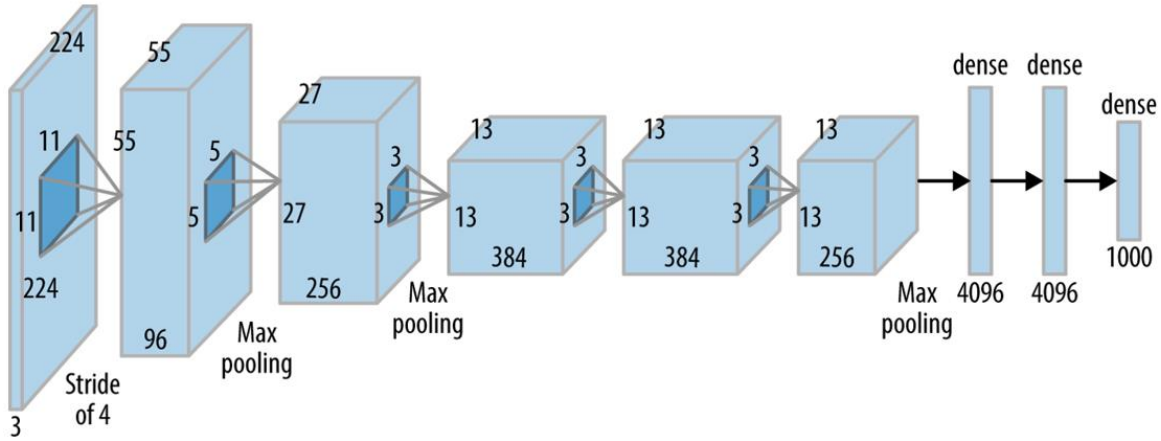


The basic architecture of AlexNet can be summarized as follows:

1. Input Layer: The input layer accepts a color image of size 227x227x3 as input.
2. Convolutional Layers: AlexNet consists of five convolutional layers, where each layer is followed by a max-pooling layer. The convolutional layers use filters of different sizes and depths to extract features from the input image.
3. Activation Function: A rectified linear unit (ReLU) activation function is used after each convolutional layer to introduce non-linearity into the network and make the model more expressive.
4. Local Response Normalization: After the first and second convolutional layers, a local response normalization (LRN) layer is used to normalize the output of the previous layer.
5. Dropout: To reduce overfitting, a dropout layer is used after the last convolutional layer and before the first fully connected layer. Dropout randomly drops out some of the units in the layer during training to prevent the model from relying too heavily on any single feature.
6. Fully Connected Layers: AlexNet has three fully connected layers with 4096 units each, followed by a final output layer with 1000 units, corresponding to the 1000 classes in the ImageNet dataset.
7. Softmax Activation Function: The output layer uses a softmax activation function to compute the probability distribution over the 1000 classes in the dataset.

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

AlexNet



Explain Alex Net architecture in details

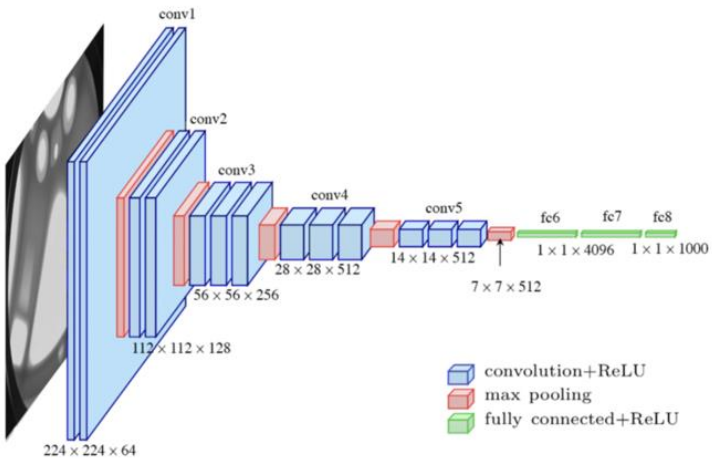
AlexNet is a convolutional neural network architecture that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. The architecture consists of 5 convolutional layers, 3 max-pooling layers, and 3 fully connected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

VGG-16



Explain VGG 16 architecture in 1 paragraph

The VGG16 architecture is a convolutional neural network with 16 layers, consisting of 13 convolutional layers and 3 fully connected layers. It uses a small filter size of 3x3 with a stride of 1 and padding of 1 for better feature extraction, and max-pooling layers to reduce the spatial dimensions of the output. The architecture achieved state-of-the-art performance on various image classification tasks by stacking multiple convolutional layers, increasing depth while maintaining the same filter size and stride, and using dropout regularization to prevent overfitting.

Fine Tuning

1. So far we have seen how to use a pretrained network directly for classification or as a feature extractor. We generally use these approaches when the domain of the target dataset (problem we are interested in solving) is somewhat similar to the source domain (imagenet problem)
2. But what if the target domain is different from the source domain or if its very different ?
3. We just extract the correct feature maps from the source domain pretrained network and fine tune them to the target domain. This is known as fine tuning.
4. Fine tuning can be formally defined as freezing a few of the network layers that are used for feature extraction and transferring all of its high level feature maps (outputs of later convolutional layers) to your domain.

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.

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.

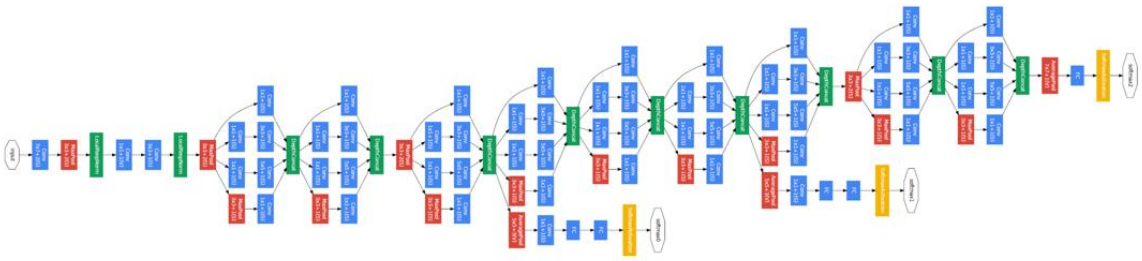
In summary, ROC and AUC are important metrics for evaluating the performance of binary classification models, and they can help you choose the best model for your specific task.

CVA 6 – ADV CNN

Advanced CNN Architecture in 1 paragraph

Advanced CNN architectures such as ResNet, Inception, DenseNet, Xception, and EfficientNet have revolutionized computer vision by enabling machines to see, recognize, and interpret visual data. These architectures have introduced new concepts such as residual learning, multi-scale feature extraction, dense connections, depthwise separable convolutions, and compound scaling, which have improved the accuracy, efficiency, and generalization of CNNs. These models have achieved state-of-the-art performance in various computer vision tasks and have paved the way for more sophisticated applications such as image recognition, object detection, and image segmentation.

GoogleNet/Inception V1



Google's Approach

Configuring the kernel size and positioning the pool layers are decisions that are made by trial and error and experiment

Google RnD Team's idea - ***"Instead of choosing a desired filter size in a convolutional layer and deciding where to place the pooling layers, let's apply all of them all together in one block and call it the inception module."***

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.

Regenerate response

Brainstorming of Inception Module

The naïve version of Inception module has a big computational cost problem due to the usage and processing of larger filters like 5x5

The input volume with dimensions of 32 × 32 × 200 will be fed to the 5 × 5 convolutional layer of 32 filters with dimensions = 5 × 5 × 32.

This means the total number of multiplications that the computer needs to compute is 32 × 32 × 200 multiplied by 5 × 5 × 32, which is more than **163 million** operations. (32x32x200=204800, 5x5x32=800 -> 204800x800 = **163,840,000**

While Google thought that this computation can be performed with modern computers this computation is still expensive.

Dimensionality Reduction on Inception V1

Inception V1 uses dimensionality reduction techniques to reduce the number of parameters and computational cost of the model, while maintaining or even improving the accuracy of the model.

One of the key techniques used for dimensionality reduction in Inception V1 is the use of 1x1 convolutions. 1x1 convolutions are used to reduce the number of channels in the input feature maps, which reduces the computational cost of subsequent convolutional layers. This technique is particularly useful in the inception modules, where multiple parallel convolutional layers are concatenated together. By using 1x1 convolutions, the number of channels in each branch can be reduced before concatenation, which reduces the number of parameters and computational cost.

Another technique used for dimensionality reduction in Inception V1 is the use of average pooling instead of fully connected layers for classification. Instead of using fully connected layers, Inception V1 uses a global average pooling layer, which averages the values of each feature map across its spatial dimensions. This reduces the number of parameters and eliminates the need for flattening the feature maps before the fully connected layers.

In addition to these techniques, Inception V1 also uses regularization techniques, such as L2 regularization and dropout, to prevent overfitting and improve generalization performance.

Overall, the dimensionality reduction techniques used in Inception V1 help to reduce the number of parameters and computational cost of the model, while maintaining or improving its accuracy. This has made Inceptio[illegible]tive CNN architecture for image classification tasks.

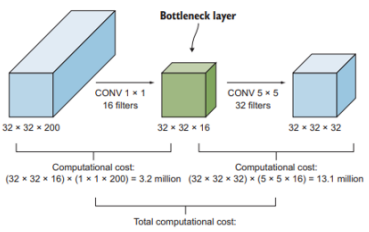
Regenerate response

Dimensionality Reduction Layer (1x1 Conv Layer)

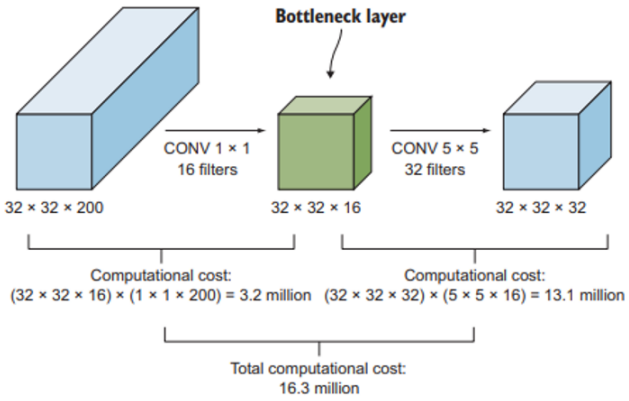
The **1 × 1** convolutional layer can reduce the operational cost of 163 million operations to about a tenth of that. That is why it is called a reduce layer/bottleneck layer.

The idea here is to add a **1 × 1 convolutional layer** before the bigger kernels like the **3 × 3** and **5 × 5** convolutional layers, to reduce their depth, which in turn will reduce the number of operations.

Suppose we have a input feature map of 32x32x200 and we apply a 1x1x16 convolution. This reduces the dimension volume from 200 to 16 channels. We can now apply 5x5 convolution to this output.



Dimensionality Reduction Layer (1x1 Conv Layer)

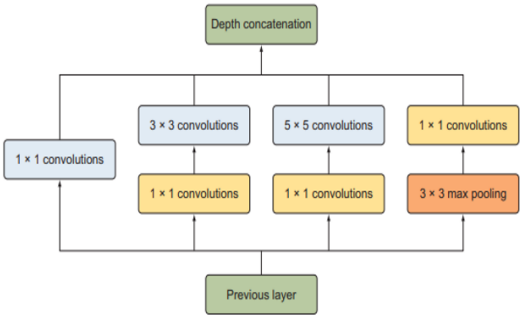


Inception Module with Dimensionality Reduction

Does shrinking the depth representation size so dramatically hurt the performance of neural network ?

Szegedy et al. ran experiments and found that as long as you implement the reduce layer in moderation, you can shrink the representation size significantly without hurting performance and save a lot of computations

Note: We will add a 1 × 1 convolutional reduce layer before the 3 × 3 and 5 × 5 convolutional layers to reduce their computational cost. We will also add a 1 × 1 convolutional layer after the 3 × 3 max-pooling layer because pooling layers don't reduce the depth for their inputs



In deep learning, a tensor is a multi-dimensional array that represents a data structure that can be processed by a neural network. Tensors can have any number of dimensions, from 0 (a scalar) to n (a tensor with n dimensions). For example, a grayscale image can be represented as a 2D tensor, where each element represents a pixel intensity. Similarly, a color image can be represented as a 3D tensor, where each element represents a pixel intensity for each color channel (red, green, and blue).

On the other hand, convolution is a mathematical operation that is used in CNNs to process input data, such as images or videos. In a convolutional layer, a small matrix called a kernel or filter is applied to the input data to produce a new feature map. The kernel is typically smaller than the input data and slides across the input data to compute the convolution operation at each location. Convolutional layers help to extract important features from the input data and are used extensively in CNNs for image and video processing tasks.

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

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.

Microsoft's Approach

MS research team introduced a novel **residual module** architecture with skip connections

The network also features heavy batch normalization for the hidden layers

Upon observing AlexNet, VGGNet, Inception you must have noticed the deeper the network, the larger its learning capacity and the better it extracts features from the images.

This happens because **deeper networks** are able to represent **very complex functions,** which allows the network to learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers).

VGG-19 was 19 layers, GoogleNet was 22 layers can we go deeper ?

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.

---

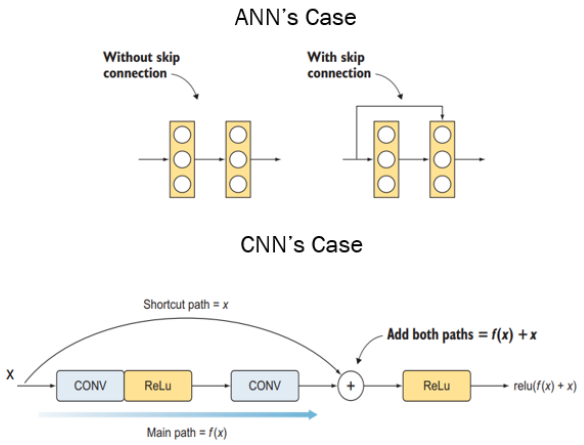
https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/

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



What are skip connections in Resnet in simple language and in details

Skip connections, also known as shortcut connections, are a way to help neural networks learn more effectively, especially in deep networks. In ResNet, skip connections are added by connecting the output of a layer to the input of a layer that is two or more layers deeper in the network.

The basic idea behind skip connections is that they allow the neural network to learn the identity mapping more easily. This means that the network can learn to map the input directly to the output, without having to learn a complex mapping through multiple layers.

In ResNet, a skip connection is created by adding the input of a layer to the output of another layer, skipping one or more layers in between. This is achieved using a residual block, which consists of two convolutional layers followed by a skip connection. The skip connection adds the input of the block to the output of the second convolutional layer, allowing the network to learn the residual mapping (the difference between the input and output) more easily.

Microsoft's approach to Vanishing Gradient

Unlike traditionally just stacking the convolutions just one after the other they also add the original input to the output of the stacked convolutional blocks

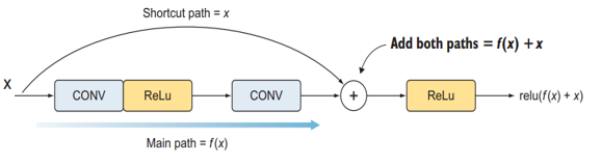
The shortcut arrow/skip connection is added at the end of the convolution layer

The reason is that we add both paths before we apply the ReLU activation function of this layer

We apply the ReLU activation to f(x) + x to produce the output signal: ReLU(f(x) + x).

Note: This combination of the skip connection and convolutional layers is called a residual block. Similar to the Inception network, ResNet is composed of a series of these residual block building blocks that are stacked on top of each other

CNN's Case

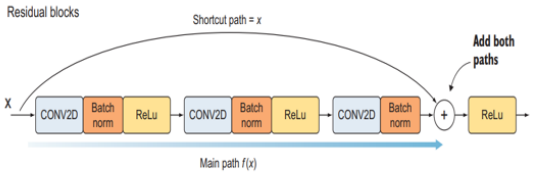


Residual Blocks

Residual blocks consist of two branches

1. **Shortcut path:** Connects the input to an addition of the second branch
2. **Main path:** A series of convolutions and activations. The main path consists of three convolutional layers with ReLU activations. We also add batch normalization to each convolutional layer to reduce overfitting and speed up training. The main path architecture looks like this: [CONV -> BN -> ReLU] × 3.

As mentioned earlier the shortcut path is added to the main path right before the activation function of the last convolution layer. Then we add ReLU function after adding the two paths.



Residual blocks are the basic building blocks of ResNet (Residual Network) architecture. They are used to overcome the problem of vanishing gradients in very deep neural networks by introducing skip connections between layers.

A residual block consists of two convolutional layers followed by a skip connection that adds the input of the block to its output. The output of the first convolutional layer is passed through an activation function (typically ReLU) before being fed into the second convolutional layer. The output of the second convolutional layer is then added to the input of the block using the skip connection. Finally, the output of the block is passed through another activation function and sent to the next layer in the network.

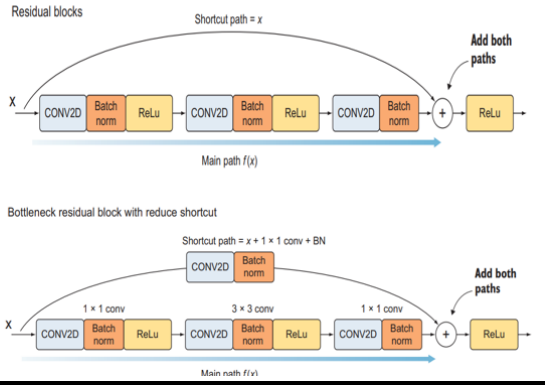
The skip connection allows the information to bypass one or more layers, making it easier for the network to learn the identity mapping, which is the mapping from the input to the output. The skip connection also helps to prevent the gradient from becoming too small to propagate through the network during training.

Residual blocks are used to construct very deep networks in ResNet. The skip connections allow the gradients to flow more easily through the network, enabling the network to train effectively even with a large number of layers.

Residual Blocks

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.

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 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.

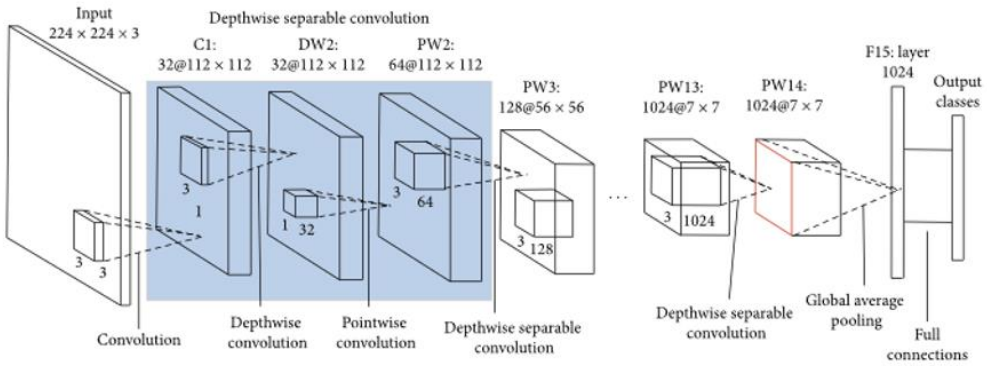
MobileNet: Google's approach at light weight models

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

MobileNet



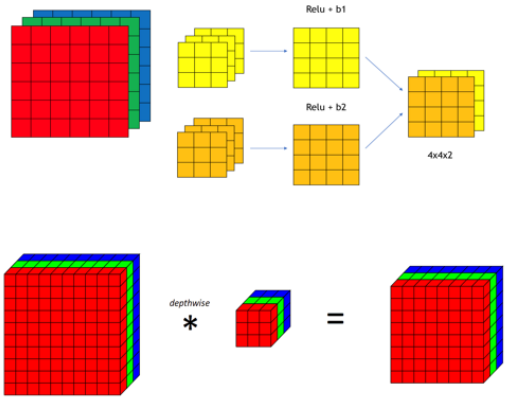
MobileNet Approach: Depthwise Separable Convolutions

In regular convolution a kernel is applied across all channels of the image

A depthwise convolution is basically a convolution along only one spatial channel of the image.

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.

Note: Since we are applying one convolutional filter for each output channel, the number of output channels is equal to the number of input channels. After applying this depthwise convolutional layer, we then apply a pointwise convolutional layer. Here the pointwise convolution will not be known as reduce but as expand layer.

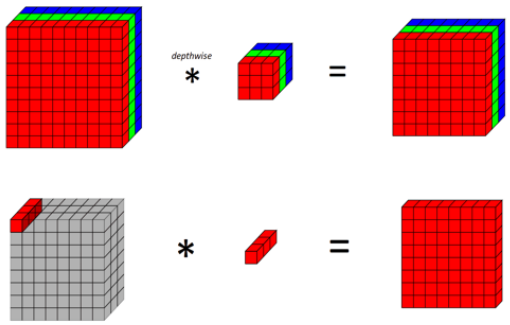


MobileNet Approach: Depthwise Separable Convolution + Pointwise Convolution

Suppose we have a RGB image of size 224x224x3. You want to apply the convolutional layer with 3x3 size kernel with 64 total kernels

1. 3x3x3x64+64=1792 parameters (if directly 64 3x3 kernels are applied)
2. DSC+ PWC
    1. 3x3x1x3+3 = 30 params (DSC)
    2. 1x1x3x64+64 = 256 params (PWC)
3. 30 + 256 = 286 parameters

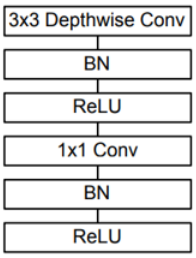
Same operation performed with significantly lesser number of parameters !



Parameters are the values that a machine learning algorithm learns from the training data in order to make predictions. In a neural network, for example, parameters include the weights and biases of the neurons. These values are adjusted during the training process in order to minimize the error between the predicted output and the true output. Parameters are learned automatically by the model during training, and their values are not set by the user.

On the other hand, hyperparameters are values that are set by the user before the training process begins, and they control how the model learns from the data. Hyperparameters include things like the learning rate, regularization strength, and number of layers in a neural network. These values are not learned by the model during training, but are set by the user based on prior knowledge, experimentation, or trial and error.

MobileNet Architecture



Note: MobileNet does not use pooling either. It uses strides of 2 for downsampling width and height. This has been the case for most of the modern architectures

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

what are width multipliers

Width multipliers are a technique used in deep learning to reduce the number of parameters and computations required by a neural network, while still maintaining its accuracy. This technique involves scaling the number of channels in each layer of the network by a factor of less than one, such as 0.5 or 0.25.

In a neural network, each layer consists of a set of neurons or filters that extract features from the input data. The number of neurons in each layer determines the number of parameters in the network, which can be computationally expensive and memory-intensive. By applying a width multiplier to each layer, the number of channels in the layer is reduced, and the number of parameters and computations required by the network are correspondingly reduced.

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

MobileNet Summary

Mobilenet makes use of special type of convolution combined with pointwise convolution

1. Depthwise Separable Convolutions used along with Pointwise Convolutions to reduce the number of parameters as well as computations
2. Width Multiplier and Resolution Multiplier hyperparameters used for further making the model more light weight

---

what are resolution multipliers

Resolution multipliers are a technique used in deep learning to reduce the input resolution of an image or video, in order to decrease the computational cost and memory requirements of a neural network. This technique involves scaling down the size of the input image or video by a factor of less than one, such as 0.5 or 0.25.

In a neural network for image or video processing, the input size can significantly impact the number of parameters and computations required by the network. By applying a resolution multiplier to the input data, the size of the input image or video is reduced, which in turn reduces the number of parameters and computations required by the network.

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

---

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.

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