Mid-Term Report

Summer of Science - Computer Vision

Anuj Gupta

Roll No. 21D070014

B.Tech Electrical Engineering

Mentor - Harsh Poonia

What is Computer Vision?

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs and take actions or make recommendations based on that information.
If AI enables computers to think, computer vision enables them to see, observe and understand.
Computer vision works much the same as human vision, except humans have a head start.
Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyze thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities.

Roadmap Followed

Computer Vision has two approaches - Classical way and using Deep Learning. Classical way uses well developed algorithm to extract the desired features from the images whereas Deep Learning form of computer vision directly trains an end to end model to extract the features. Deep Learning way is more recent and quite robust to implement on various situations. Briliiant models can be trained on availability of huge amount of data and if not then Transfer Learning comes to save.

- Started with learning Neural Networks & its roots and then carried on leaning in details about tuning the hyperparameters.
- learnt **Convolutional Neural Network (CNN)** to get insights about the Deep Learning approach to Computer Vision.

Neural Networks

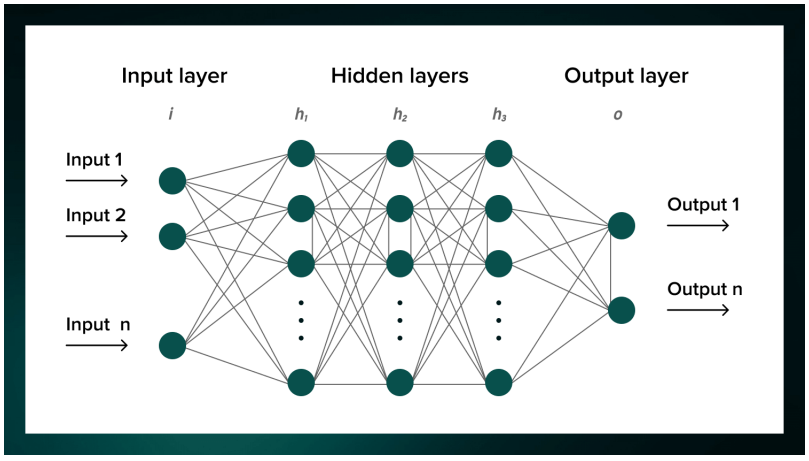

Figure 1: Deep Neural Network

Neural networks are a class of machine learning models inspired by the structure and functioning of the human brain. They are composed of interconnected nodes, called neurons, organized into layers. Each neuron receives inputs, performs a computation, and produces an output,

which is then passed to the next layer. The connections between neurons, often referred to as synapses, carry weights that determine the strength and impact of the input signals.
Neurons are the basic building blocks of neural networks. They are mathematical units that receive input signals, perform computations, and produce output signals. The collective behavior of neurons in a network enables the network to learn and make predictions.
Neural Networks is made up of various key building blocks which are as follows :-

- **Input** : Neurons receive input signals from other neurons or from the external environment. These inputs can be real-valued numbers representing features or activation values from the previous layer of the network.
- **Weights** : Each input signal is associated with a weight, which determines the importance or influence of that input on the neuron's computation. Weights are initially assigned random values and are adjusted during the training process to optimize the network's performance.
- **Activation Function** : After receiving inputs, a neuron applies an activation function to produce an output. The activation function introduces non-linearity to the network, allowing it to model complex relationships between inputs and outputs. Common activation functions include the sigmoid, ReLU (Rectified Linear Unit), and tanh (hyperbolic tangent) functions.
- **Summation Function** : The inputs to a neuron are multiplied by their corresponding weights, and the results are summed up. The summation function combines the weighted inputs, typically with an additional bias term, to compute the total input to the neuron.
- **Bias** : A bias term is added to the weighted sum of inputs before passing it through the activation function. The bias allows the neuron to shift the activation function's threshold, controlling the neuron's sensitivity to inputs.
- **Output** : The output of a neuron is the result of applying the activation function to the computed sum of weighted inputs plus the bias. This output is then passed to the next layer of neurons in the network.
- **Connection** : Neurons are interconnected through connections, which transmit the output of one neuron as input to another. Each connection has an associated weight that determines the strength or influence of the signal transmitted across the connection.

Neurons work collectively in layers to process and transform data in a neural network. The input layer receives the initial data, which is then passed through one or more hidden layers, and finally, the output layer produces the network's predictions or outputs. The weights and biases of the neurons are adjusted during the training process using various optimization algorithms, such as gradient descent, to minimize the difference between predicted and target outputs and improve the network's performance.

Hyperparameters, Regularisation and Optimization

Hyperparameters in a neural network are the settings and configuration choices made by the developer or researcher that affect the behavior and performance of the network during training and inference. These parameters are typically set before training begins and are not learned by the network itself. Here are some commonly used hyperparameters in a neural network:

- **Learning Rate** : This determines the step size at each iteration of the optimization algorithm (e.g., gradient descent) and affects how quickly the network learns. A high learning rate may cause the network to converge quickly but might lead to overshooting, while a low learning rate may result in slow convergence or getting stuck in a suboptimal solution.
- **Number of Hidden Layers** : The number of hidden layers refers to the layers between the input and output layers. It determines the depth and capacity of the network. Increasing the number of hidden layers can allow the network to learn more complex representations but may also increase the risk of overfitting.
- **Number of Neurons per Hidden Layer** : The number of neurons in each hidden layer affects the network's capacity and ability to learn complex patterns. A higher number of neurons can potentially capture more intricate relationships in the data but might also increase the computational complexity and risk overfitting.
- **Activation Functions** : Activation functions introduce non-linearities to the network and determine the output of a neuron. Popular choices include ReLU (Rectified Linear Unit), sigmoid, and tanh. The choice of activation function depends on the problem at hand, and different activation functions can yield different network behaviors
- **Batch Size** : During training, the data is divided into batches, and the network updates its weights based on the gradients calculated on each batch. The batch size determines the number of samples in each batch. A smaller batch size can lead to noisy updates but allows for faster convergence, while a larger batch size provides a more stable estimate of the gradients but may require more memory.
- **Regularization** Techniques : Regularization methods such as L1 or L2 regularization and dropout can be used to prevent overfitting. These techniques introduce additional terms or modifications to the loss function to penalize overly complex models or encourage generalization.
- **Optimizer** : The optimizer determines how the network's weights are updated based on the calculated gradients during training. Popular choices include stochastic gradient descent (SGD), Adam, RMSprop, and AdaGrad. Each optimizer has its own update rules and hyperparameters that affect convergence speed and accuracy.
- **Number of Epochs** : An epoch refers to one complete pass of the entire training dataset through the network. The number of epochs determines how many times the network will see the data during training. Too few epochs may result in underfitting, while too many epochs can lead to overfitting.

The selection of appropriate hyperparameters often requires experimentation and tuning to achieve optimal performance for a given task.

Convolutional Neural Network

Convolution Operation

The convolution operation is a fundamental operation in Convolutional Neural Networks (CNNs) that plays a crucial role in extracting features from input data. In the context of CNNs, the convolution operation involves applying a set of learnable filters (also called kernels or convolutional filters) to the input data, producing a feature map as the output.

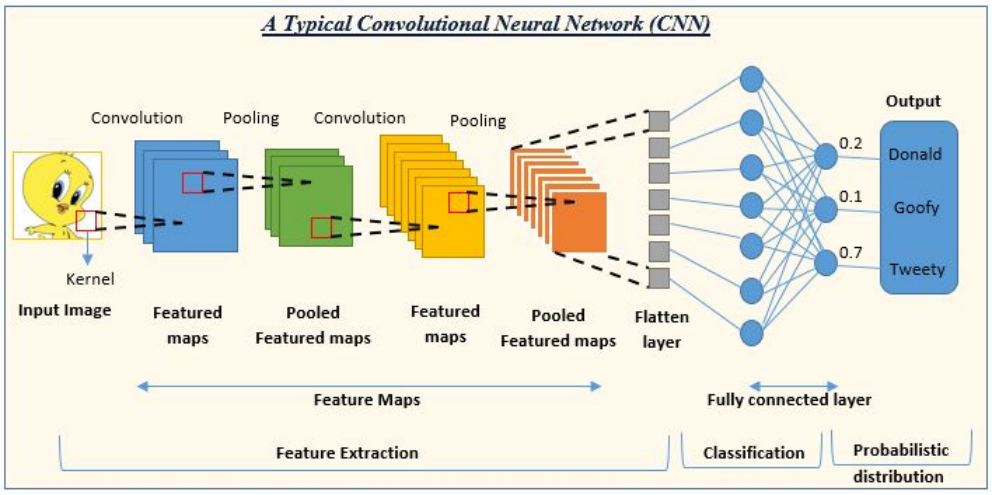

Figure 2: Convolutional Neural Network

The convolution operation can be understood as a sliding window operation, where the filter is systematically applied to different regions of the input data. Here are the key steps involved in the convolution operation:

- **Filter Initialization** : Before the convolution operation, the filters are initialized with random values. During training, these filters are learned through backpropagation to capture important patterns or features in the input data.
- **Sliding Window** : The filter is moved across the input data in a systematic manner, typically with a defined stride. The stride determines the step size at which the filter moves horizontally and vertically. The filter is applied to each region of the input, resulting in a corresponding region in the output feature map.
- **Element-wise Multiplication and Summation** : At each position of the filter, element-wise multiplication is performed between the filter values and the values of the input data that fall within the filter's receptive field. The element-wise products are then summed to obtain a single value. This process is repeated for each position of the filter, generating a single value for each position.
- **Bias Addition** : After the element-wise multiplication and summation, a bias term is added to the resulting value. The bias term provides an additional learnable parameter that can help account for any constant offset or bias in the data.
- **Non-linear Activation** : Typically, an activation function is applied to the output of each convolution operation. Common activation functions include Rectified Linear Unit (ReLU), sigmoid, or hyperbolic tangent (tanh). The activation function introduces non-linearities to the network, enabling it to learn complex relationships and capture non-linear patterns in the data.
- **Output Feature Map** : The resulting values from the convolution operation and activation function form the elements of the output feature map. Each value in the output feature map represents a learned feature or pattern that the filter has detected in the input data.

In a CNN, multiple filters are applied simultaneously to the input data, each producing its own output feature map. These feature maps are stacked together along the depth dimension to form the output tensor of the convolutional layer.

The convolution operation is repeated across different layers of the CNN, allowing the network to learn hierarchical representations of the input data. Deeper layers capture more abstract and complex features by combining information from multiple lower-level feature maps.
Through training, the filters in the convolutional layers learn to extract discriminative features from the input data, enabling the CNN to perform tasks such as image classification, object detection, and image segmentation.

Padding

Padding in CNN refers to the technique of adding extra border pixels (padding) around the input image before applying convolutional operations. The purpose of padding is to preserve the spatial dimensions of the input image or feature maps throughout the convolutional layers, allowing the network to capture information at the borders of the image.
When convolutional layers are applied to an input image, the spatial dimensions of the feature maps typically decrease due to the shrinking effect caused by the convolution operation. This reduction in size can be problematic in certain scenarios, especially when the spatial information at the edges or corners of the image is crucial.
Padding helps to mitigate this issue by adding extra pixels around the borders of the input image. The padding can be added symmetrically, where the same number of pixels is added to each side of the image, or asymmetrically, which allows for more control over the padding in each dimension.
The added border pixels are typically filled with zeros (zero-padding), but other padding values can also be used. By including these extra pixels, the spatial dimensions of the feature maps are preserved or reduced to a lesser extent, depending on the padding size.

Superficial structure of a typical CNN

A Convolutional Neural Network (CNN) is a type of neural network commonly used for image and video processing tasks. It is designed to automatically learn and extract meaningful features from input data through a series of specialized layers. CNNs are particularly effective in handling grid-like data, such as images, due to their ability to capture spatial relationships. The structure of a typical CNN consists of the following components:

- **Input Layer** : The input layer receives the raw input data, which is usually an image represented as a grid of pixel values. The dimensions of the input layer are determined by the size of the input image.

- **Convolutional Layers** : Convolutional layers are the core building blocks of a CNN. Each layer applies a set of learnable filters (also called kernels) to the input data using a convolution operation. The filters detect local patterns or features, such as edges or textures, by sliding across the input spatially and producing a feature map. The depth (number of filters) in each layer determines the number of feature maps generated.

- **Activation Function** : Typically, an activation function (e.g., ReLU) follows the output of each convolutional layer. The activation function introduces non-linearities to the network, allowing it to learn complex relationships and model more sophisticated features.

- **Pooling Layers** : Pooling layers reduce the spatial dimensions (width and height) of the feature maps while preserving their important features. Popular pooling techniques include max pooling, which selects the maximum value within a region, and average pooling, which calculates the average value. Pooling helps in reducing the computational complexity and provides a form of translation invariance.

- **Flattening** : Before connecting to the fully connected layers, the feature maps need to be flattened into a 1D vector. This flattening operation reshapes the feature maps into a suitable format for the fully connected layers.
- **Fully Connected Layers** : Fully connected layers, also known as dense layers, are typically placed after the convolutional and pooling layers. These layers connect every neuron in the previous layer to every neuron in the next layer. They are responsible for learning high-level representations by combining features learned in the earlier layers. The final fully connected layer usually produces the desired output, such as class probabilities or regression values.
- **Output Layer** : The output layer provides the final predictions or outputs of the CNN. The number of neurons in this layer depends on the specific task. For example, in classification tasks, the output layer may have neurons corresponding to different classes, while in regression tasks, it may have a single neuron or multiple neurons for different output variables.

In addition to these components, CNNs can also include various regularization techniques like dropout or batch normalization to improve generalization and prevent overfitting. The overall architecture and the number of layers in a CNN can vary depending on the specific problem and dataset, and it is common to stack multiple convolutional and pooling layers to learn hierarchical features at different scales and abstractions.
Training a CNN involves forward propagation, where the input data passes through the layers, and backward propagation (backpropagation), where the network adjusts its weights based on the computed loss to minimize the error. The weights are updated using optimization algorithms such as stochastic gradient descent (SGD) or its variants.
By leveraging the hierarchical nature of CNNs, these networks have achieved state-of-the-art performance in various computer vision tasks, including image classification, object detection, and image segmentation.

References

[1] https://www.coursera.org/learn/neural-networks-deep-learning/home/week/1

[2] https://www.coursera.org/learn/deep-neural-network/home/week/1

[3] https://www.coursera.org/learn/convolutional-neural-networks/home/week/1

[4] Google for Images

Modified Plan of Action

Week 5	Studying some classic CNN networks and learning object detection and localisation
Week 6	Studying feature extraction techniques like edge detection (e.g., Canny edge detection) and corner detection (e.g., Harris corner detection) and exploring image segmentation algorithms such as thresholding, region growing, or watershed
Week 7	Learning about feature descriptors such as SIFT, SURF, or ORB. Understanding how to match and compare feature descriptors between images. Studying object detection techniques, such as Haar cascades or HOG (Histogram of Oriented Gradients)
Week 8	Diving into advanced topics, such as image registration, optical flow, or camera calibration. Studying more specialized areas like image stitching, panoramic image creation, or 3D reconstruction
	Final Report and Presentation