# UNIT III THIRD-GENERATION NEURAL NETWORKS

Spiking Neural Networks-Convolutional Neural Networks-Deep Learning Neural Networks-Extreme Learning Machine Model-Convolutional Neural Networks: The Convolution Operation – Motivation – Pooling – Variants of the basic Convolution Function – Structured Outputs – Data Types – Efficient Convolution Algorithms – Neuroscientific Basis – Applications: Computer Vision, Image Generation, Image Compression.

## 1.SPIKING NEURAL NETWORK (SNN)

**Artificial neural networks** that closely mimic natural neural networks are known as spiking neural networks (SNNs). In addition to neuronal and synaptic status, SNNs incorporate time into their working model. The idea is that neurons in the SNN do not transmit information at the end of each propagation cycle (as they do in traditional multi-layer perceptron networks), but only when a membrane potential – a neuron's intrinsic quality related to its membrane electrical charge – reaches a certain value, known as the threshold.

### Spiking Neural Network Work

#### *Key Concepts*

What distinguishes a traditional ANN from an SNN is the information propagation approach. SNN aspires to be as close to a biological neural network as feasible. As a result, rather than working with continually changing time values as ANN does, SNN works with discrete events that happen at defined times. SNN takes a set of spikes as input and produces a set of spikes as output (a series of spikes is usually referred to as spike trains).

#### *Spike Based Neural Codes*

Artificial spiking neural networks are designed to do neural computation. This necessitates that neural spiking is given meaning: the variables important to the computation must be defined in terms of the spikes with which spiking neurons communicate. A variety of neuronal information encodings have been proposed based on biological knowledge:

- ***Binary Coding:***

Binary coding is an all-or-nothing encoding in which a neuron is either active or inactive within a specific time interval, firing one or more spikes throughout that time frame. The finding that

physiological neurons tend to activate when they receive input (a sensory stimulus such as light or external electrical inputs) encouraged this encoding.

- ***Rate Coding:***

Only the rate of spikes in an interval is employed as a metric for the information communicated in rate coding, which is an abstraction from the timed nature of spikes. The fact that physiological neurons fire more frequently for stronger (sensory or artificial) stimuli motivates rate encoding.

- ***Fully Temporal Codes***

The encoding of a fully temporal code is dependent on the precise timing of all spikes. Evidence from neuroscience suggests that spike-timing can be incredibly precise and repeatable. Timings are related to a certain (internal or external) event in a fully temporal code (such as the onset of a stimulus or spike of a reference neuron).

- ***Latency Coding***

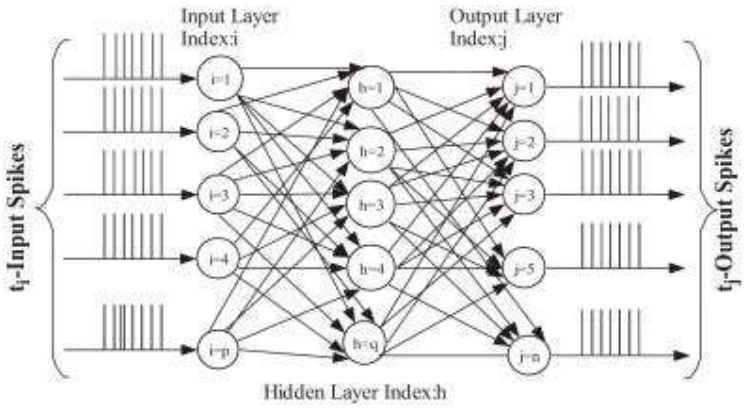
The timing of spikes is used in latency coding, but not the number of spikes. The latency between a specific (internal or external) event and the first spike is used to encode information. This is based on the finding that significant sensory events cause upstream neurons to spike earlier.

***SNN Architecture***

Spiking neurons and linking synapses are described by configurable scalar weights in an SNN architecture. The analogue input data is encoded into the spike trains using either a rate-based technique, some sort of temporal coding or population coding as the initial stage in building an SNN.

A biological neuron in the brain (and a simulated spiking neuron) gets synaptic inputs from other neurons in the neural network, as previously explained. Both action potential production and network dynamics are present in biological brain networks.

APEC




***Learning Rules in SNN's***

Learning is achieved in practically all ANNs, spiking or non-spiking, by altering scalar-valued synaptic weights. Spiking allows for the replication of a form of bio-plausible learning rule that is not possible in non-spiking networks. Many variations of this learning rule have been uncovered by neuroscientists under the umbrella term spike-timing-dependent plasticity (STDP).

- ***Unsupervised Learning***

Data is delivered without a label, and the network receives no feedback on its performance. Detecting and reacting to statistical correlations in data is a common activity. Hebbian learning and its spiking generalizations, such as STDP, are a good example of this. The identification of correlations can be a goal in and of itself, but it can also be utilized to cluster or classify data later on.

- ***Supervised Learning***

In supervised learning, data (the input) is accompanied by labels (the targets), and the learning device's purpose is to correlate (classes of) inputs with the target outputs (a mapping or regression between inputs and outputs). An error signal is computed between the target and the actual output and utilized to update the network's weights.

**Application of Spiking Neural Networks**

In theory, SNNs can be used in the same applications as standard ANNs. SNNs can also stimulate the central nervous systems of biological animals, such as an insect seeking food in an unfamiliar environment. They can be used to examine the operation of biological brain networks due to their realism.

**Advantages and Disadvantages of SNN**

***Advantages***

- SNN is a dynamic system. As a result, it excels in dynamic processes like speech and dynamic picture identification.
- When an SNN is already working, it can still train.
- To train an SNN, you simply need to train the output neurons.
- Traditional ANNs often have more neurons than SNNs; however, SNNs typically have fewer neurons.
- Because the neurons send impulses rather than a continuous value, SNNs can work incredibly quickly.
- Because they leverage the temporal presentation of information, SNNs have boosted information processing productivity and noise immunity.

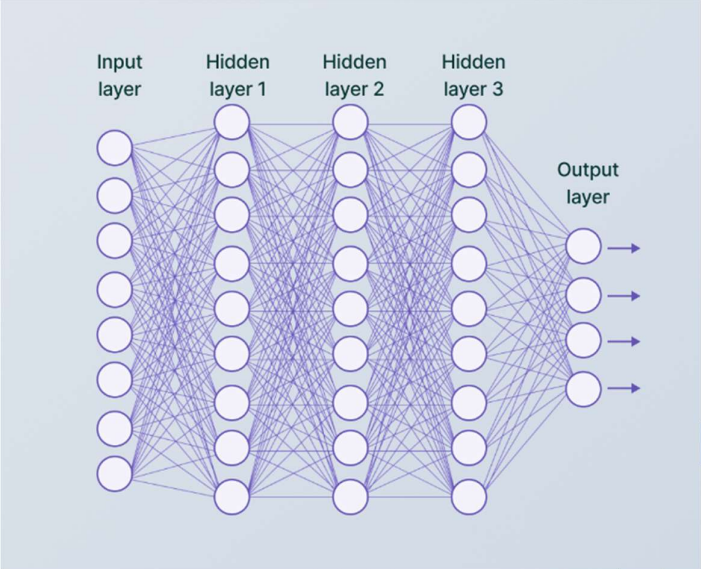
***Disadvantages***

- SNNs are difficult to train.
- As of now, there is no learning algorithm built expressly for this task.
- Building a small SNN is impracticable.

**2. CONVOLUTIONAL NEURAL NETWORKS:**

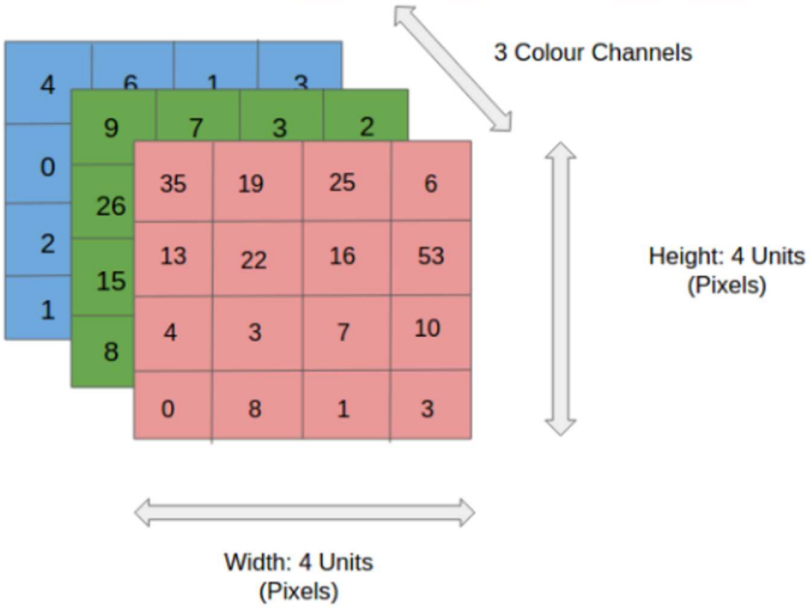
In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

APEC

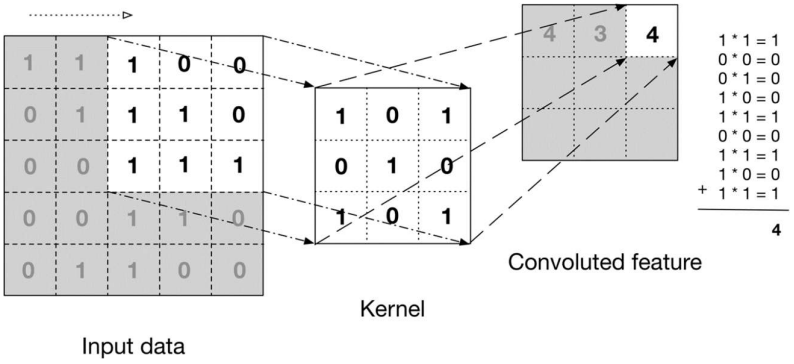



**CNN work?**

working of Convolutional neural networks (CNN), let's cover the basics, such as what an image is and how it is represented. An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.




For simplicity, let's stick with grayscale images as we try to understand how CNNs work.




**Background of Convolutional neural networks (CNNs)**

CNN's were first developed and used around the 1980s. The most that a Convolutional Neural Networks (CNN) could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc. The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources. This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.

In 2012 Alex Krizhevsky realized that it was time to bring back the branch of deep learning that uses multi-layered neural networks. The availability of large sets of data, to be more specific ImageNet datasets with millions of labeled images and an abundance of computing resources enabled researchers to revive CNNs

**Limitations of Convolutional neural networks (CNNs)**

Despite the power and resource complexity of CNNs, they provide in-depth results. At the root of it all, it is just recognizing patterns and details that are so minute and inconspicuous that it goes unnoticed to the human eye. But when it comes to **understanding** the contents of an image it fails.

**CNNS LAYERS:**

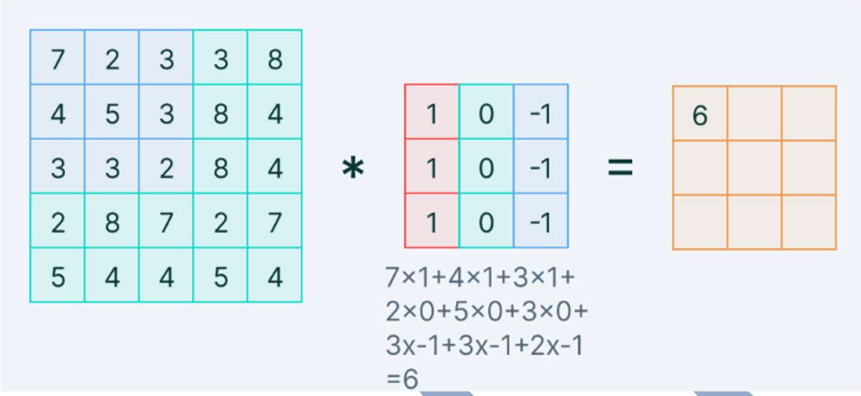
Here's an overview of layers used to build Convolutional Neural Network architectures.

*Convolutional Layer*

CNN works by comparing images piece by piece.

Filters are spatially small along width and height but extend through the full depth of the input image. It is designed in such a manner that it detects a specific type of feature in the input image.

In the convolution layer, we move the filter/kernel to every possible position on the input matrix. Element-wise multiplication between the filter-sized patch of the input image and filter is done, which is then summed.




The translation of the filter to every possible position of the input matrix of the image gives an opportunity to discover that feature is present anywhere in the image.

## 3. DEEP LEARNING NEURAL NETWORKS:

### What is a Neuron in Deep Learning?

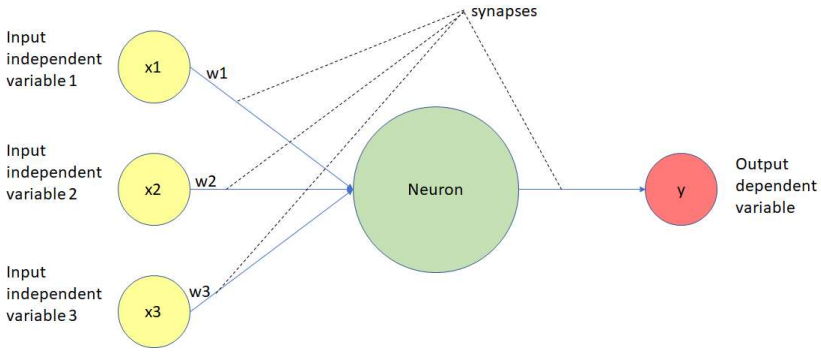
Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Neurons in deep learning models are nodes through which data and computations flow.

Neurons work like this:

- They receive one or more input signals. These input signals can come from either the raw data set or from neurons positioned at a previous layer of the neural net.
- They perform some calculations.
- They send some output signals to neurons deeper in the neural net through a synapse.

Here is a diagram of the functionality of a neuron in a deep learning neural net:

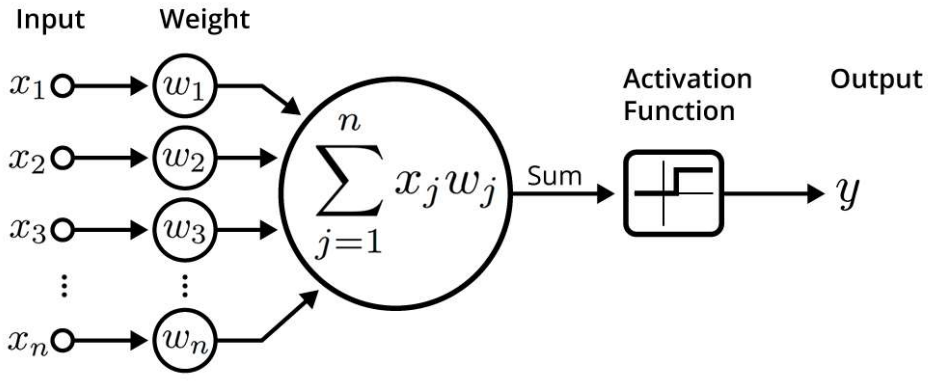



Let's walk through this diagram step-by-step.

As you can see, neurons in a deep learning model are capable of having synapses that connect to more than one neuron in the preceding layer. Each synapse has an associated weight, which impacts the preceding neuron's importance in the overall neural network.

Weights are a very important topic in the field of deep learning because adjusting a model's weights is the primary way through which deep learning models are trained. You'll see this in practice later on when we build our first neural networks from scratch.

Once a neuron receives its inputs from the neurons in the preceding layer of the model, it adds up each signal multiplied by its corresponding weight and passes them on to an activation function, like this:




The activation function calculates the output value for the neuron. This output value is then passed on to the next layer of the neural network through another synapse.

This serves as a broad overview of deep learning neurons. Do not worry if it was a lot to take in – we'll learn much more about neurons in the rest of this tutorial. For now, it's sufficient for you to have a high-level understanding of how they are structured in a deep learning model.

**Deep Learning Activation Functions**

Activation functions are a core concept to understand in deep learning.

They are what allows neurons in a neural network to communicate with each other through their synapses.

In this section, you will learn to understand the importance and functionality of activation functions in deep learning.

**What Are Activation Functions in Deep Learning?**

In the last section, we learned that neurons receive input signals from the preceding layer of a neural network. A weighted sum of these signals is fed into the neuron's activation function, then the activation function's output is passed onto the next layer of the network.

There are four main types of activation functions that we'll discuss in this tutorial:

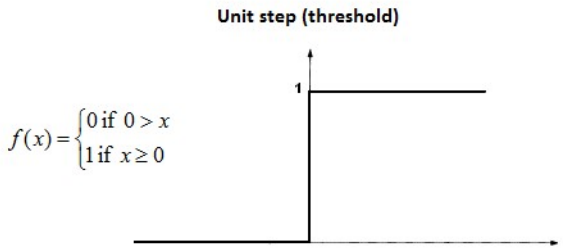
- Threshold functions
- Sigmoid functions
- Rectifier functions, or ReLUs
- Hyperbolic Tangent functions

Let's work through these activations functions one-by-one.

**Threshold Functions**

Threshold functions compute a different output signal depending on whether or not its input lies above or below a certain threshold. Remember, the input value to an activation function is the weighted sum of the input values from the preceding layer in the neural network.

Mathematically speaking, here is the formal definition of a deep learning threshold function:




As the image above suggests, the threshold function is sometimes also called a unit step function.

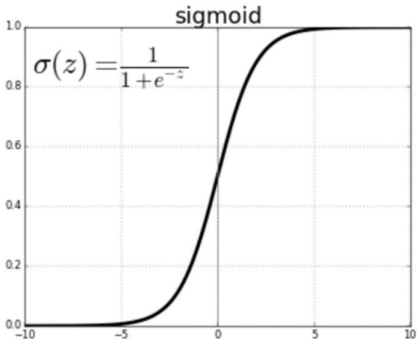
Threshold functions are similar to boolean variables in computer programming. Their computed value is either 1 (similar to True) or 0 (equivalent to False).

**The Sigmoid Function**

The sigmoid function is well-known among the data science community because of its use in logistic regression, one of the core machine learning techniques used to solve classification problems.

The sigmoid function can accept any value, but always computes a value between 0 and 1.

Here is the mathematical definition of the sigmoid function:




One benefit of the sigmoid function over the threshold function is that its curve is smooth. This means it is possible to calculate derivatives at any point along the curve.

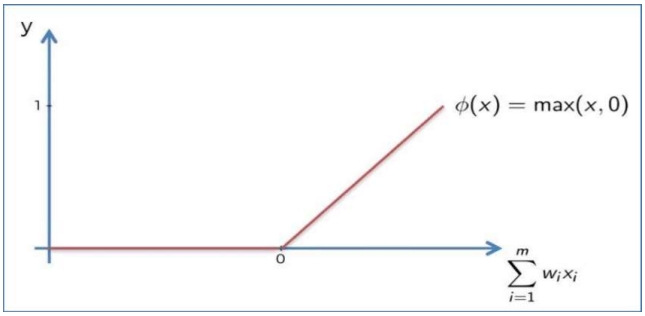
**The Rectifier Function**

The rectifier function does not have the same smoothness property as the sigmoid function from the last section. However, it is still very popular in the field of deep learning.

The rectifier function is defined as follows:

- If the input value is less than 0, then the function outputs 0
- If not, the function outputs its input value

Here is this concept explained mathematically:

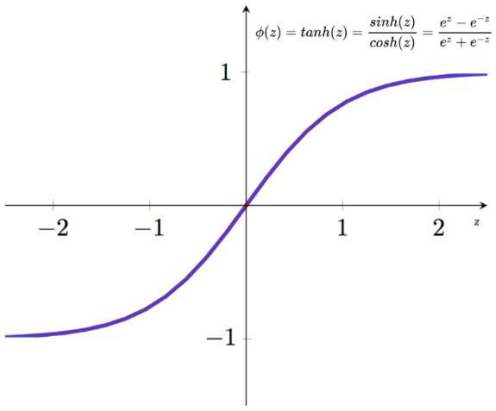



Rectifier functions are often called Rectified Linear Unit activation functions, or ReLUs for short.

**The Hyperbolic Tangent Function**

The hyperbolic tangent function is the only activation function included in this tutorial that is based on a trigonometric identity.

It's mathematical definition is below:



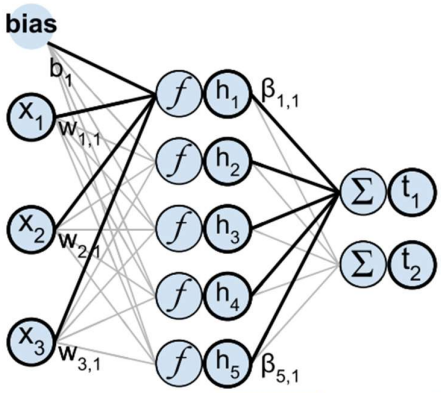

The hyperbolic tangent function is similar in appearance to the sigmoid function, but its output values are all shifted downwards.

## 4. EXTREME LEARNING MACHINE (ELM):

Extreme learning machines are feed-forward neural networks having a single layer or multiple layers of hidden nodes for classification, regression, clustering, sparse approximation, compression, and feature learning, where the hidden node parameters do not need to be modified. These hidden nodes might be assigned at random and never updated, or they can be inherited from their predecessors and never modified. In most cases, the weights of hidden nodes are usually learned in a single step which essentially results in a fast learning scheme.

### Fundamentals of ELM

An ELM is a quick way to train SLFN networks (shown in the below figure). An SLFN comprises three layers of neurons, however, the name Single refers to the model's one layer of non-linear neurons which is the hidden layer. The input layer offers data features but does not do any computations, whereas the output layer is linear with no transformation function and no bias.




The ELM technique sets input layer weights W and biases b at random and never adjusts them. Because the input weights are fixed, the output weights ???? are independent of them (unlike in the Backpropagation training method) and have a straightforward solution that does not require iteration. Such a solution is also linear and very fast to compute for a linear output layer.

Random input layer weights improve the generalization qualities of a linear output layer solution because they provide virtually orthogonal (weakly correlated) hidden layer features. A linear system's solution is always in a range of inputs. If the solution weight range is constrained, orthogonal inputs provide a bigger solution space volume with these constrained weights. Smaller weight norms tend to make the system more stable and noise resistant since input errors are not aggravating in the

output of the linear system with smaller coefficients. As a result, the random hidden layer creates weakly correlated hidden layer features, allowing for a solution with a low norm and strong generalization performance.

**Variants of ELM**

In this section, we will summarize several variants of ELM and will introduce them briefly.

***ELM for Online Learning***

There are numerous types of data in real-world applications, thus ELM must be changed to effectively learn from these data. For example, because the dataset is increasing, we may not always be able to access the entire dataset. From time to time, new samples are added to the dataset. Every time the set grows, we must retrain the ELM.

***Incremental ELM***

To build an incremental feedforward network, **Huang et al**. developed an incremental extreme learning machine (I-ELM). When a new hidden node was introduced, I-ELM randomly added nodes to the hidden layer one by one, freezing the output weights of the existing hidden nodes. I-ELM is effective for SLFNs with piecewise continuous activation functions (including differentiable) as well as SLFNs with continuous activation functions ( such as threshold).

***Pruning ELM***

**Rong et al**. proposed a pruned-ELM (P-ELM) algorithm as a systematic and automated strategy for building ELM networks in light of the fact that using too few/many hidden nodes could lead to underfitting/overfitting concerns in pattern categorization. P-ELM started with a large number of hidden nodes and subsequently deleted the ones that were irrelevant or lowly relevant during learning by considering their relevance to the class labels.

***Error-Minimized ELM***

**Feng et al**. suggested an error-minimization-based method for ELM (EM-ELM) that can automatically identify the number of hidden nodes in generalized SLFNs by growing hidden nodes one by one or group by group. The output weights were changed incrementally as the networks grew, reducing the computational complexity dramatically.

***Evolutionary ELM***

When ELM is used, the number of hidden neurons is usually selected at random. Due to the random determination of input weights and hidden biases, ELM may require a greater number of hidden neurons. **Zhu et al**. introduced a novel learning algorithm called evolutionary extreme learning machine (E-ELM) for optimizing input weights and hidden biases and determining output weights.

To improve the input weights and hidden biases in E-ELM, the modified differential evolutionary algorithm was utilized. The output weights were determined analytically using Moore– Penrose (MP) generalized inverse.

**Applications of ELM**

Extreme learning machine has been used in many application domains such as medicine, chemistry, transportation, economy, robotics, and so on due to its superiority in training speed, accuracy, and generalization. This section highlights some of the most common ELM applications.

***IoT Application***

As the Internet of Things (IoT) has gained more attention from academic and industry circles in recent years, a growing number of scientists have developed a variety of IoT approaches or applications based on modern information technologies.

***Transportation Application***

The application of machine learning in transportation is a popular issue. Scientists, for example, used machine learning techniques to create a driver sleepiness monitoring system to prevent unsafe driving and save lives.

## 5. CONVOLUTIONAL NEURAL NETWORKS:

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node

is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

**convolutional neural networks work?**

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

**Convolutional layer**

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyperparameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

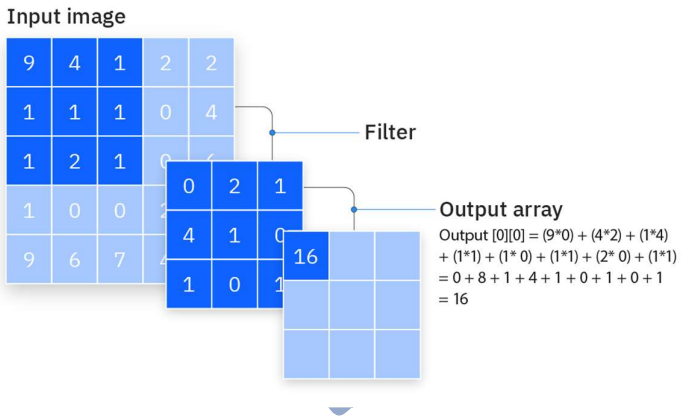
1. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

2. Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

3. Zero-padding is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:

- Valid padding: This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
- Same padding: This padding ensures that the output layer has the same size as the input layer.
- Full padding: This type of padding increases the size of the output by adding zeros to the border of the input.

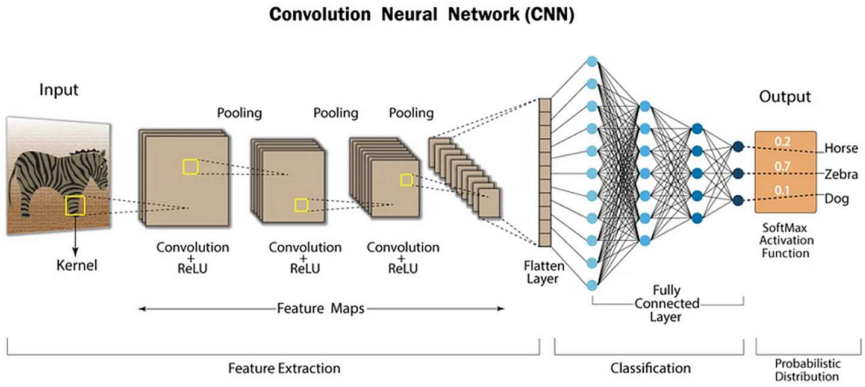
After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.




## 6. CONVOLUTION OPERATION

Convolution neural network is the major building block of deep learning, which helps in image classification, object detection, image recognition, etc of computer vision tasks. We use

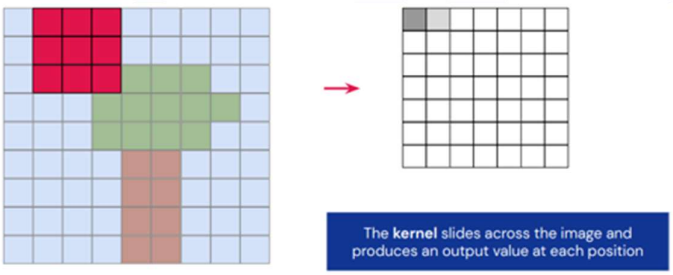
many convolution operation techniques that we will discuss in this article.



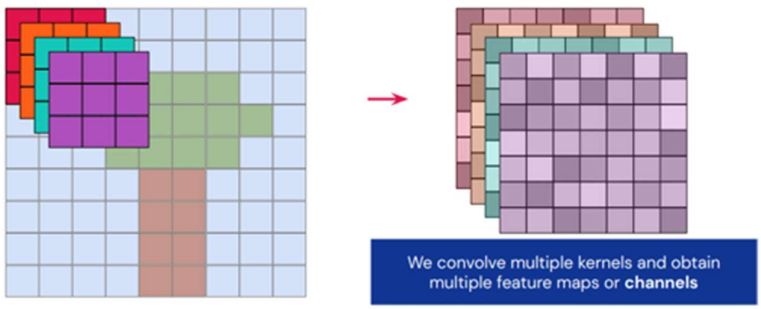

## THE CONVOLUTION OPERATIONS

The convolution operation is the process of implying a combination of two functions that produce the third function as a result, employing filters across the entire input image allows the filter to discover that feature of the image. Which is also called a feature map.

So in Practical set , the convolutional operation is implemented by making The kernel slides across The image and produces an output Value at each position

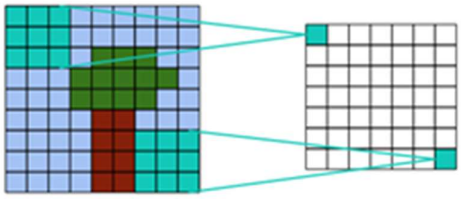



Also we convolve different Kernels and as a result obtain Different feature maps or channels



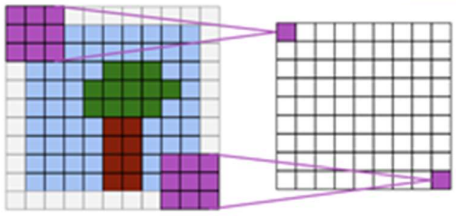
**Variants of The Convolution Operation :**

- **Valid Convolution:** Doesn't used any padding



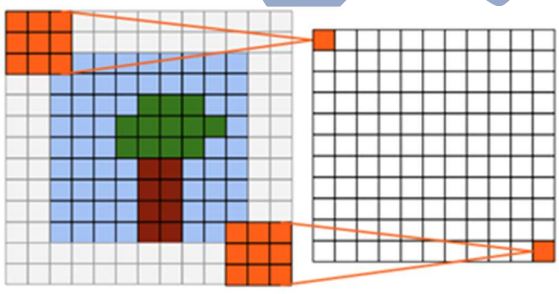
**Valid convolution**: output size = input size – kernel size + 1

- **Same Convolution** :pads in a way that the input size is the same as the output size



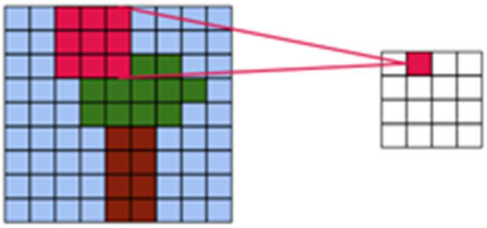
**Same convolution**: output size = input size

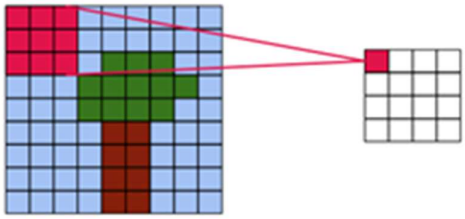
- **Full Convolution :** — : we compute output wherever the kernel and the output overlap by at least 1 pixel



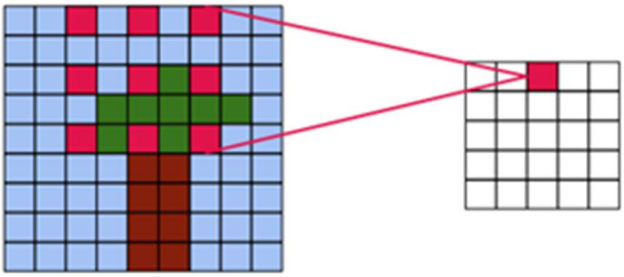
**Full convolution**: output size = input size + kernel size – 1

- **Strided Convolution** : : kernel slides along the image with a step > 1

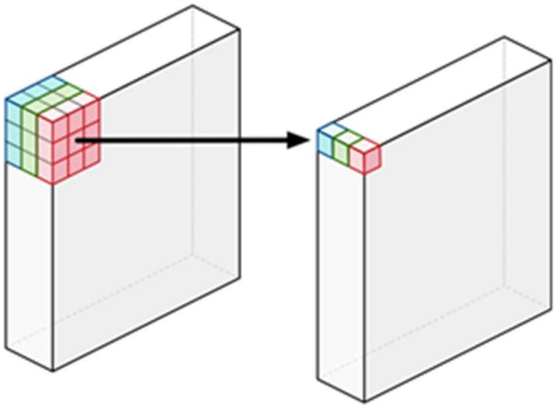




- **Dilated Convolution :** : kernel is spread out, step > 1 between kernel elements



- **Depth wise Convolution** : each output channel is connected only to one input channel



## 7. MOTIVATION BEHIND CONVOLUTION

Convolution leverages three important ideas that motivated computer vision researchers: sparse interaction, parameter sharing, and equivariant representation. Let's describe each one of them in detail.

Trivial neural network layers use matrix multiplication by a matrix of parameters describing the interaction between the input and output unit. This means that every output unit interacts with every input unit. However, convolution neural networks have *sparse interaction.* This is achieved by making kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.

If computing one feature at a spatial point (x1, y1) is useful then it should also be useful at some other spatial point say (x2, y2). It means that for a single two-dimensional slice i.e., for creating one activation map, neurons are constrained to use the same set of weights. In a traditional neural network, each element of the weight matrix is used once and then never revisited, while convolution network has *shared parameters* i.e., for getting output, weights applied to one input are the same as the weight applied elsewhere.

Due to parameter sharing, the layers of convolution neural network will have a property of *equivariance to translation*. It says that if we changed the input in a way, the output will also get changed in the same way.

## 8. POOLING LAYERS

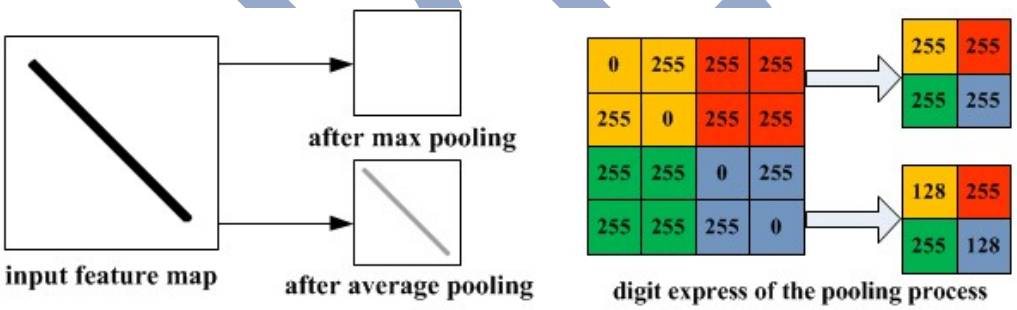
Pooling layers are one of the building blocks of Convolutional Neural Networks. Where Convolutional layers **extract features** from images, Pooling layers **consolidate the features** learned by CNNs. Its purpose is to gradually shrink the representation's spatial dimension to minimize the number of parameters and computations in the network.

Why are Pooling layers needed?

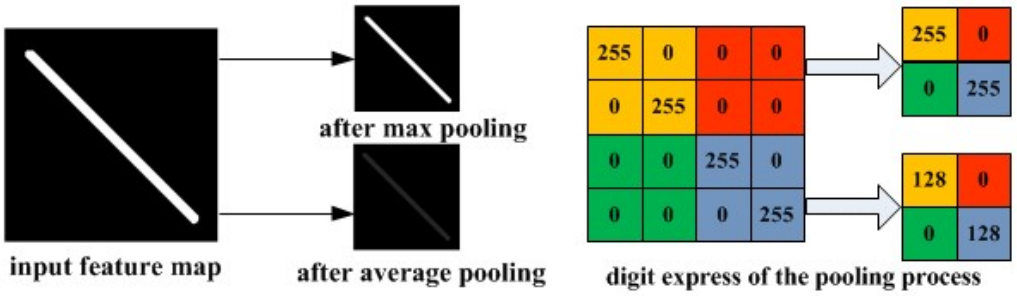
The feature map produced by the filters of Convolutional layers is location-dependent. For example, If an object in an image has shifted a bit it might not be recognizable by the Convolutional layer. So, it means that the feature map records the precise positions of features in the input. What pooling layers provide is "Translational Invariance" which makes the CNN invariant to translations, i.e., even if the input of the CNN is translated, the CNN will still be able to recognize the features in the input.

There are **two types of poolings** that are used:

1. **Max pooling**: This works by selecting the maximum value from every pool. Max Pooling retains the **most prominent** features of the feature map, and the returned image is sharper than the original image.
2. **Average pooling**: This pooling layer works by getting the average of the pool. Average pooling retains the **average values** of features of the feature map. It smoothes the image while keeping the essence of the feature in an image.




**(a) Illustration of max pooling drawback**




**(b) Illustration of average pooling drawback**

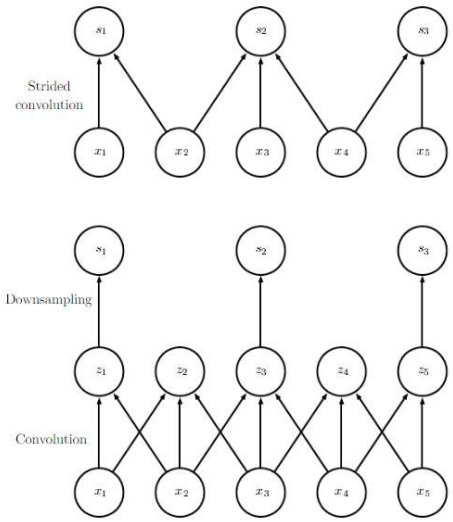
## 9. VARIANTS OF THE BASIC CONVOLUTION FUNCTION

Convolution in the context of NN means an operation that consists of many applications of convolution in parallel.

- Kernel K with element Ki,j,k,l giving the connection strength between a unit in channel i of output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit.
- Input: Vi,j,k with channel i, row j and column k
- Output Z same format as V
- Use 1 as first entry

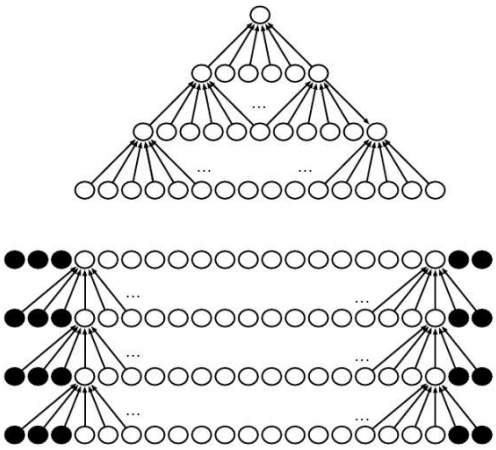
**Full Convolution**

Convolution with a stride greater than 1 pixel is equivalent to conv with 1 stride followed by downsampling:




**Some 0 Paddings and 1 stride**

Without 0 paddings, the width of representation shrinks by one pixel less than the kernel width at each layer. We are forced to choose between shrinking the spatial extent of the network rapidly and using small kernel. 0 padding allows us to control the kernel width and the size of the output independently.



Special case of 0 padding:

- Valid: no 0 padding is used. Limited number of layers.
- Same: keep the size of the output to the size of input. Unlimited number of layers. Pixels near the border influence fewer output pixels than the input pixels near the center.
- Full: Enough zeros are added for every pixels to be visited k (kernel width) times in each direction, resulting width m + k - 1. Difficult to learn a single kernel that performs well at all positions in the convolutional feature map.

Usually the optimal amount of 0 padding lies somewhere between 'Valid' or 'Same'

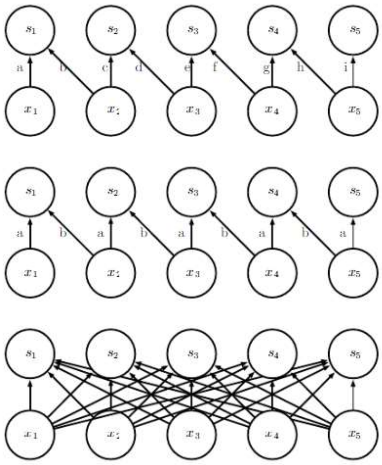
**Unshared Convolution**

In some case when we do not want to use convolution but want to use locally connected layer. We use **Unshared convolution**. Indices into weight W

- i: the output channel
- j: the output row;
- k: the output column
- l: the input channel
- m: row offset within input
- n: column offset within input

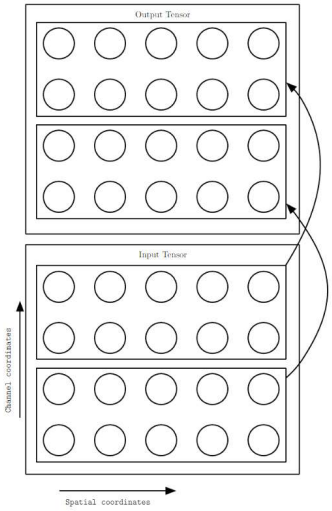
$Z_{i,j,k}=\sum_{l,m,n}[V_{l,i+m-1,j+n-1}W_{i,j,k,l,m,n}]$

Comparison on local connections, convolution and full connection



Useful when we know that each feature should be a function of a small part of space, but no reason to think that the same feature should occur accross all the space. eg: look for mouth only in the bottom half of the image.

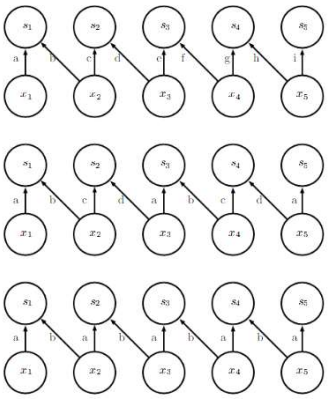
It can be also useful to make versions of convolution or local connected layers in which the connectivity is further restricted, eg: constrain each output channeel i to be a function of only a subset of the input channel.




**Tiled Convolution**

Learn a set of kernels that we rotate through as we move through space. Immediately neighboring locations will have different filters, but the memory requirement for storing the

parameters will increase by a factor of the size of this set of kernels. Comparison on locally connected layers, tiled convolution and stardard convolution:



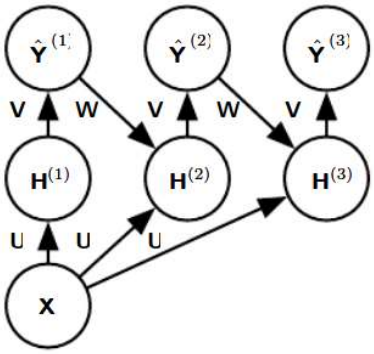
Local connected layers and tiled convolutional layer with max pooling: the detector units of these layers are driven by different filters. If the filters learn to detect different tranformed version of the same underlying features, then the max-pooled units become invariant to the learned transformation.

## 10. STRUCTURED OUTPUTS:

Convolutional networks can be trained to output high-dimensional structured output rather than just a classification score. A good example is the task of image segmentation where each pixel needs to be associated with an object class. Here the output is the same size (spatially) as the input. The model outputs a tensor ***S*** where ***S[i,j,k]*** is the probability that pixel **(j,k)** belongs to class **i**.

To produce an output map as the same size as the input map, only **same-padded** convolutions can be stacked. Alternatively, a coarser segmentation map can be obtained by allowing the output map to shrink spatially.

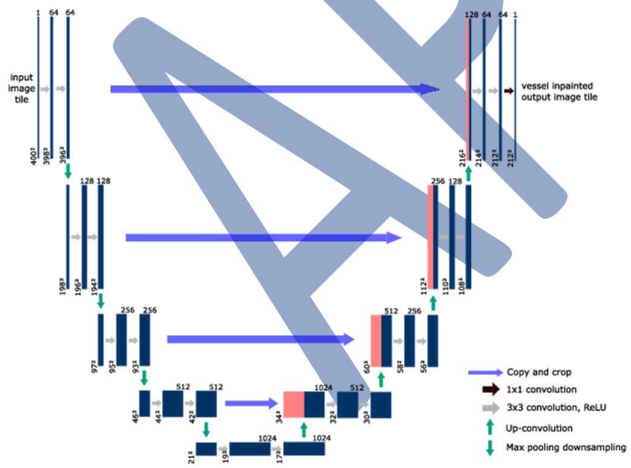
The output of the first labelling stage can be refined successively by another convolutional model. If the models use tied parameters, this gives rise to a type of **recursive model** as shownbelow. ($H^1$, $H^2$, $H^3$ share parameters)




Recursive refinement of the segmentation map

The output can be further processed under the assumption that contiguous regions of pixels will tend to belong to the same label. Graphical models can describe this relationship. Alternately, *CNNs can learn to optimize the graphical models training objective*.

Another model that has gained popularity for segmentation tasks (especially in the medical imaging community) is the *U-Net*. The *up-convolution* mentioned is just a direct upsampling by repetition followed by a convolution with *same padding*.



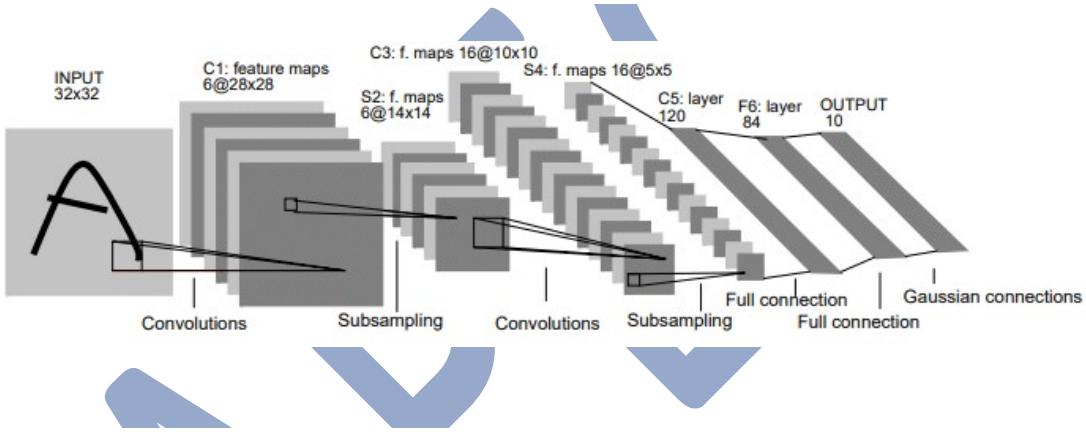

## 11. CONVOLUTIONAL NEURAL NETWORKS DATA TYPES

Convolutional neural networks are known for their superiority over other artificial neural networks, given their ability to process visual, textual, and audio data. The CNN architecture

comprises three main layers: convolutional layers, pooling layers, and a fully connected (FC) layer.

**LeNet**

LeNet is a pioneering CNN designed for recognizing handwritten characters. It was proposed by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner in the late 1990s. LeNet consists of a series of convolutional and pooling layers, as well as a fully connected layer and softmax classifier. It was among the first successful applications of deep learning for computer vision. It has been used by banks to identify numbers written on cheques in grayscale input images.




**VGG**

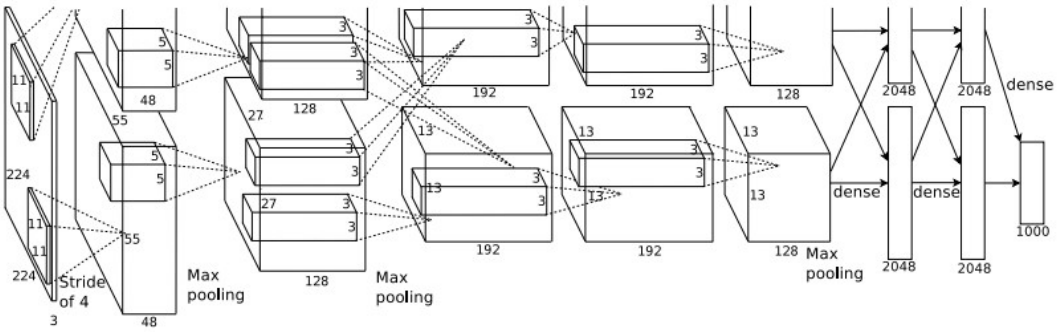
VGG (Visual Geometry Group) is a research group within the Department of Engineering Science at the University of Oxford. The VGG group is well-known for its work in computer vision, particularly in the area of convolutional neural networks (CNNs).

One of the most famous contributions from the VGG group is the VGG model, also known as VGGNet. The VGG model is a deep neural network that achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge in 2014, and has been widely used as a benchmark for image classification and object detection tasks.

**AlexNet**

AlexNet won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, with a significant margin of error reduction compared to the previous state-of-the-art model – down to 15% from 26%. It was designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton at the University of Toronto.

AlexNet consists of five convolutional layers with max pooling, and three FC layers, with ReLU activation functions used throughout. In addition to 3×3 filters, it also has 5×5 and 11×11 convolutional filters.




**ResNet**

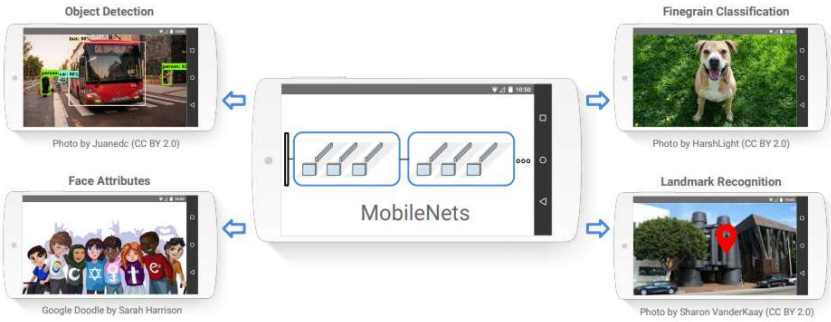
ResNet (short for "Residual Neural Network") is a family of deep convolutional neural networks designed to overcome the problem of vanishing gradients that are common in very deep networks. The idea behind ResNet is to use "residual blocks" that allow for the direct propagation of gradients through the network, enabling the training of very deep networks.

**GoogLeNet**

GoogLeNet is a deep convolutional neural network developed by researchers at Google. It was introduced in 2014 and won the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) that year, with a top-five error rate of 6.67%.

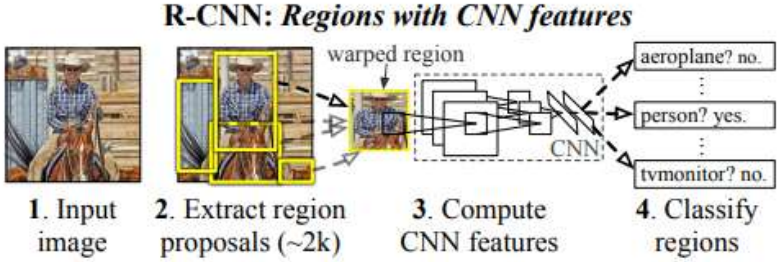
**MobileNet**

MobileNet is a family of convolutional neural networks designed for mobile and embedded devices. They are optimized for efficiency and speed, while still achieving high accuracy in tasks such as image recognition. One of the key features of MobileNet is the use of pointwise and depthwise convolutions.




**R-CNN**

R-CNN (Region-based Convolutional Neural Network) was introduced in 2013, and is a two-stage approach to object detection. First, the input image is segmented into regions of interest (ROIs) using selective search, a computationally expensive algorithm that generates region proposals based on image texture, color, and other features. Then, each ROI is passed through a CNN that produces a fixed-length feature vector. The CNN used in R-CNN is usually a pretrained network, such as AlexNet or VGG.

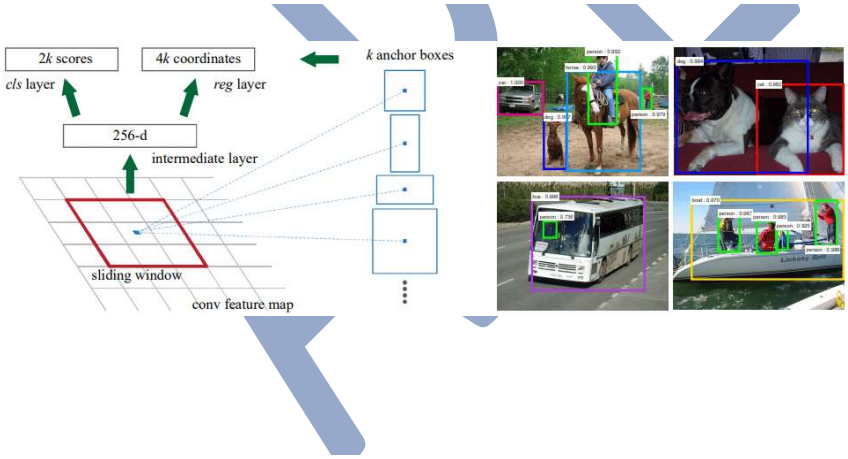


**Fast R-CNN**

Fast R-CNN, introduced in 2015, improved on R-CNN by eliminating the need for a separate feature extraction step for each region proposal. Instead, the entire image is passed through the CNN, producing a feature map that is shared by all the region proposals. Each ROI is then pooled into a feature map with a fixed size through an operation called "Region of Interest (ROI) Pooling", and passed through a sequence of FC layers to produce the final bounding box and classification predictions.

**Faster R-CNN**

Faster R-CNN, introduced in 2015, further improved on Fast R-CNN by replacing selective search with an RPN (Region Proposal Network), which is a small convolutional network that generates region proposals directly from the feature map produced by the CNN. The RPN shares the same convolutional layers as the object detection network, making the entire model more efficient and easier to train.




## 12. EFFICIENT CONVOLUTION ALGORITHMS

**Efficient Convolutional Neural Networks**

The efficient neural-network design approach either searches for an efficient base model from a predefined search pool or modifies the given base model to obtain the final compressed model. Network pruning, data quantization, tensor decomposition, and knowledge distillation approaches use the given base model, whereas more recent approaches, such as NAS,

programmatically search for highly efficient neural network structures. In this section, we detail each of these efficient neural network design approaches.

**Pruning**

Network pruning is a well-known topic that can be traced back to the early 1990s [98,99]. The key idea in pruning is to remove unimportant weights, filters, channels, or even layers from the original DNN, resulting in a reduced number of memory access and computation operations. Weight or connection pruning attempts to eliminate unimportant connections from the original model, whereas more recent techniques use large-scale pruning, such as filter-pruning or channel-pruning. In general, network pruning can be categorized as 1) connection (weight) pruning, also known as unstructured pruning, and 2) filter or channel pruning, also known as structural pruning.

***Weight Pruning***

Some early works, i.e., Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS), use the second derivative (Hessian matrix) of the loss function to prune each non-essential weight. The most popular early works in the deep learning era used a three-step method to prune the weights below the threshold and demonstrated that many connections can be removed from the DNN to achieve a significant compression ratio. First, the network is trained to find important connections, then, unimportant connections are pruned, and, finally, the network is retrained to fine-tune the remaining connections. Iterative connection pruning and fine-tuning gradually produce a compressed network. Dead neurons, for which all connected weights are pruned, can also be safely dropped.

Using similar-weight pruning techniques, the authors in proved that the accuracy of large but pruned models outperforms the accuracy of their smaller but dense counterparts under identical memory footprints. However, the lottery ticket hypothesis shows that dense, randomly-initialized, neural network contains subnetworks (“winning tickets”), when trained in isolation, have an accuracy comparable to original networks with a similar number of iterations. In their iterative pruning, each pruned subnetwork was trained from scratch where the weights are initialized with the same initial weights used in the original model. While the above methods use pruning in the spatial domain, Ref. showed that converting spatial weights into frequency–domain weight coefficients and pruning dynamically in each iteration and different frequency bands achieves a higher compression ratio. The target layers to be pruned are selected based on the quantity of small-magnitude weights in each layer estimated using

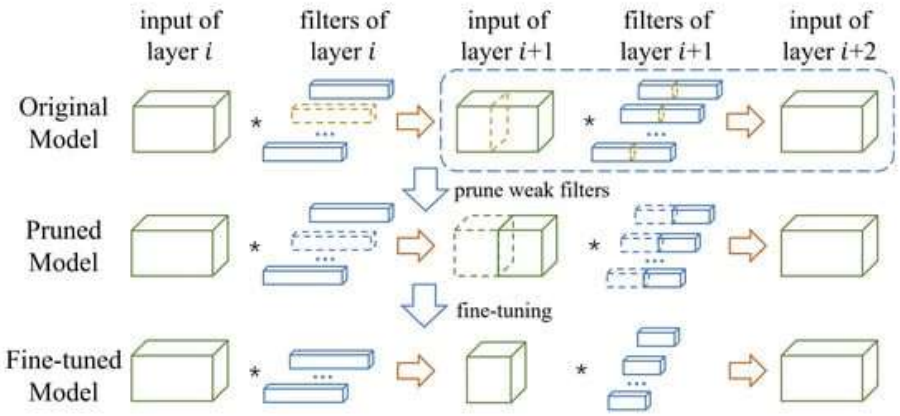
GMM, and the selected layers are pruned with a specific compression rate. In another magnitude-aware pruning method ,energy reduction, rather than only the compression ratio and accuracy loss, is also considered when determining the pruning strategy.

With connection pruning, less important weights or neurons are removed regardless of their position, resulting in an unstructured network. Although a significant number of connections can be pruned with minimal damage to the general capacity of the network, specialized software or hardware is required to support sparse matrix operations and has limited applications on the general purpose hardware.

***Structural Pruning***

In this category, the methods aim to reduce the memory footprint by pruning entire filters, channels, or even layers. After pruning, the input and output shapes of the layers and weight matrices are changed, still permitting dense matrix operations without requiring any specialized hardware or software. Identifying the least important filter and removing it is the most popular scheme in filter pruning.

The filter pruning via geometric median and online filter clustering criteria search for filters with redundancy rather than those with "relatively less" importance. In addition to filter and channel pruning, entire layer pruning methods also exist .Using different criteria, selected layers from the network were removed to obtain a compact structure. These methods claim that the model obtained by layer pruning incurs even less inference time and run-time memory usage with similar accuracy than the model obtained by filter pruning methods.



## 13. NEUROSCIENTIFIC BASIS

**Convolutional Neural Network** may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers

CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps.

**A convolutional network layer is designed to capture three properties of V1:**

1. V1 is arranged in a spatial map. It actually has a two-dimensional structure mirroring the structure of the image in the retina. **Convolutional networks capture this property by having their features defined in terms of two dimensional maps.**
2. V1 contains many simple cells. A simple cell's activity can be characterized by a linear function of the image in a small, spatially localized receptive field. **The detector units of a convolutional network are designed to emulate these properties of simple cells.**
3. V1 also contains many complex cells. These cells respond to features that are similar to those detected by simple cells, but complex cells are invariant to small shifts in the position of the feature. **This inspires the pooling units of convolutional networks.**

**There are many differences between convolutional networks and the mammalian vision system. Some of these differences are -**

1. The human eye is mostly very low resolution, except for a tiny patch called the fovea. Most convolutional networks receive large full resolution photographs as input.
2. The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
3. Even simple brain areas like V1 are heavily impacted by feedback from higher levels. Feedback has been explored extensively in neural network models but has not yet been shown to offer a compelling improvement.

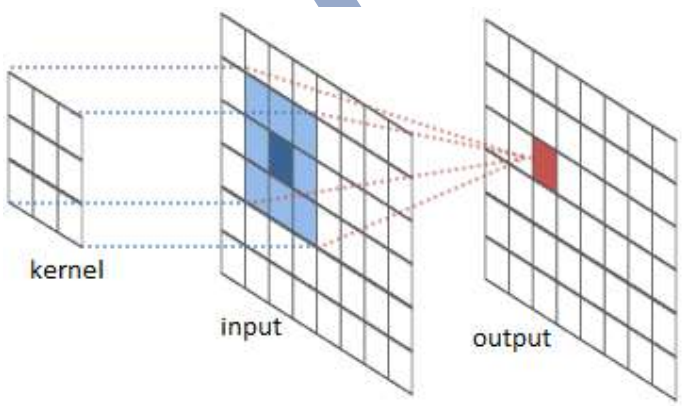
## 14. APPLICATIONS FOR CONVOLUTIONAL NETWORKS

Convolutional Neural Networks (CNNs) are widely used in various applications such as:

1. **Object Detection:** CNN can detect and locate objects in images or videos.
2. **Image Segmentation:** CNNs can segment images into different regions and tag each region with a semantic class.
3. **Create Images:** CNNs can create new images or manipulate existing ones.
4. **Video Analytics:** CNNs can be used for action detection, object tracking, and video scene segmentation.
5. **Natural Language Processing:** CNNs can be used for text classification, sentiment analysis, and language translation tasks.
6. **Autonomous Systems:** CNNs can be used in autonomous systems such as self-driving cars for lane detection, obstacle detection, and traffic sign recognition.

## 15. CONVOLUTION IN COMPUTER VISION

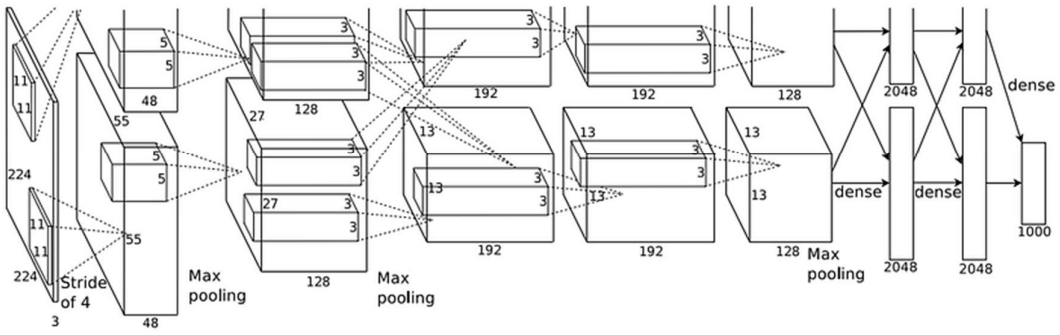
Deep learning is an advanced version of machine learning. It don't require any human to extract feature to trained from it. Take example from computer vision, before deep learning application in computer vision is found, we are still using machine learning techniques like SVM or random forest. These machine learning techniques require us to filter out the features from the image, then use those features to be the parameter of classification. These techniques lead to heavy computation, as it usually use feature of each pixel. After moving to deep learning, using each pixel as the input node leads to huge amount of hidden nodes (nodes between input and output), make computation of an standard 480*480 image take so much time.



Convolutions

Other than that, the algorithms are also overfit. In classification or segmentation, when we shifts the object, the algorithms are unable to achieve good precision anymore.

To deal with this, we can use convolution. Convolution in computer vision means extracting information of an image to a smaller size object called filter. The extractor is called kernel. It is an n*n matrix with n < image size.




AlexNet Algorithm, the most prominent computer vision. It is using convolution in its algorithm

**Why Should Deep Learning Use Convolution?**

- Reducing number of input nodes
  The convolution will filter the initial image until the image is small enough to be flatten and be used in an usual neural network.
- Tolerate small shifts in where the pixel are
  With kernel filtering in convolution, these leads to make the algorithm learn about pixel color information rather than its position.
- Take advantage for correlation in complex image
  One of the correlation we can make use color clustering. In an image, when a pixel is nearby, it will have probability to have the same color.

**Convolution process**

Convolution itself is a simple process,

- First, it will filter an image with a kernel, these kernel is defined usually with n*n and the depth to correspond the channel amount of an image or filter. Then it will create an entity called filter, these filter will be the input for the next convolution or multilevel perceptron system.
- Second, it will de-linearize the data with activation function. Each new pixel in the filter that were yield by the first step will be input for the activation function. The value that returned by the activation function will substitute the direct kernel filtering value to a values that are less linear.

A convolution basically create a new 3d matrix with size that based on the number of kernel, the width of the kernel, stride, and padding that are used in the process.

## 16. IMAGE GENERATION

### The process of image convolution

A convolution is done by multiplying a pixel's and its neighboring pixels color value by a matrix

Kernel: A kernel is a (usually) small matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers produce different results under convolution.

The size of a kernel is arbitrary but 3x3 is often used

Convolution Formula

APEC

$$V = \left| \frac{\sum_{i=1}^{q} \left( \sum_{j=1}^{q} f_{ij} d_{ij} \right)}{F} \right|$$

Where:

| | | |
|---|---|---|
| $f_{ij}$ | = | the coefficient of a convolution kernel at position *i,j* (in the kernel) |
| $d_{ij}$ | = | the data value of the pixel that corresponds to $f_{ij}$ |
| $q$ | = | the dimension of the kernel, assuming a square kernel (if $q$ = 3, the kernel is 3 × 3) |
| $F$ | = | either the sum of the coefficients of the kernel, or 1 if the sum of coefficients is 0 |
| $V$ | = | the output pixel value |

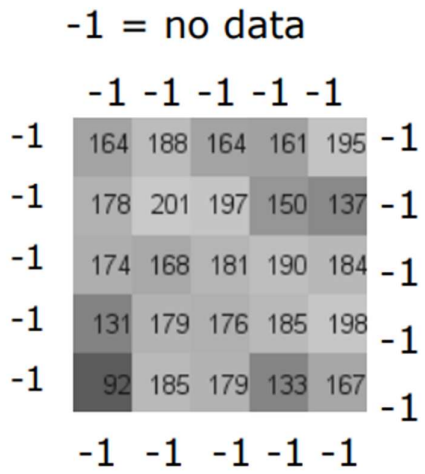
In cases where $V$ is less than 0, $V$ is clipped to 0.

**What do we do with edge pixels?**

Wrap the image

Ignore edge pixels and only compute for those pixels with all neighbors

Duplicate edge pixels so the pixel at (2,n) (where n would be nonpositive will have a value of (2,1)






Start out with an image

The choice of kernel affects the output image

Base your choice of kernel on the desired results for the image (smooth, blur, enhance, sharpen)

Low Pass and high pass filters will be discussed later in the class

**17. IMAGE COMPRESSION**

Mirror the function W in the origin to give function Wm[i,j]=W[−i,−j], then shift the weight function Wm to position (k,l) in the image, pixelwise multiply the function and shifted weight function and. sum all resulting values, this is the result of the convolution at point (i,j).

APEC