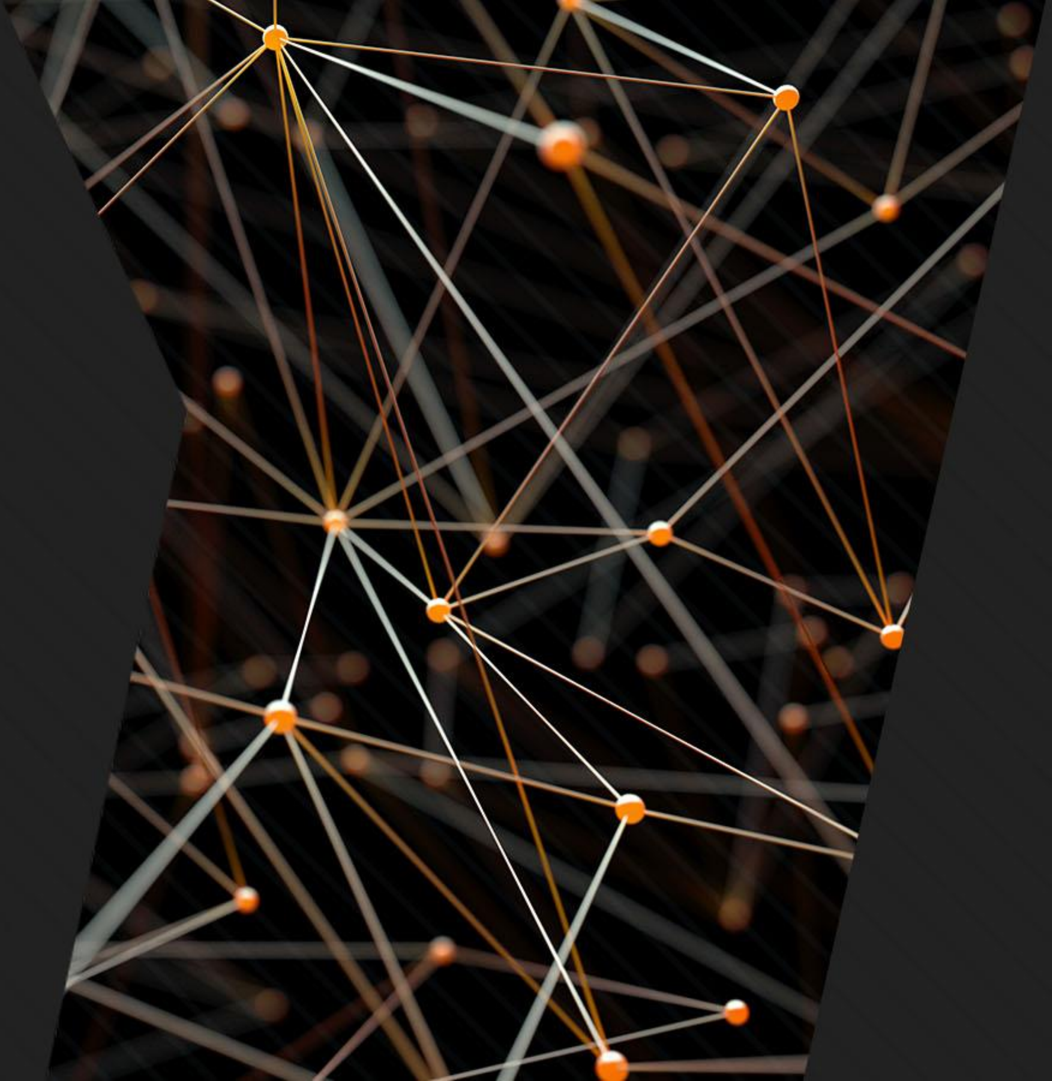
Applications of ANN

Unit-VI



Pattern Classification

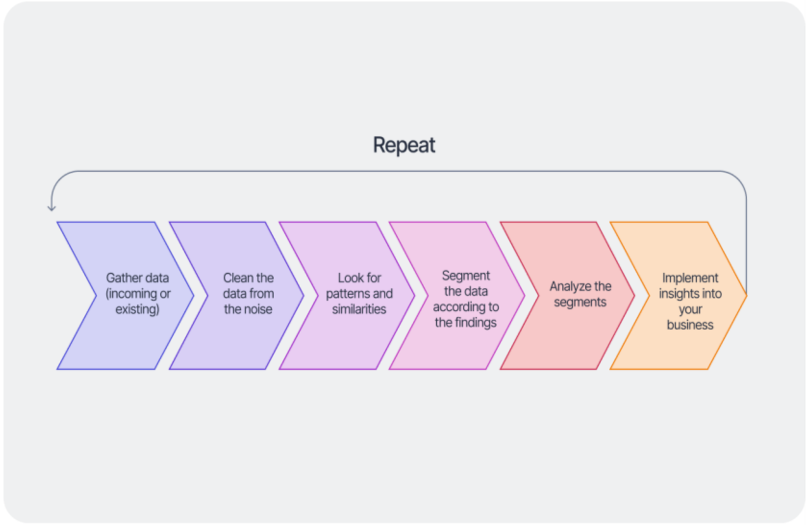
- Also referred to as "Pattern Recognition"
- A pattern can be any entity of interest that one needs to recognize and identify
- A pattern- any regularly repeated arrangement
  - ➔ described by its features - Characteristics of the examples for a given problem
  - ➔ e.g. in a face recognition task some features could be the color of the eyes or the distance between the eyes
- Examples of patterns are
  - ➔ Human faces
  - ➔ colors on the clothes
  - ➔ Speech signal etc.
- Goal of a (supervised) pattern classification task is to
  - ➔ Find a functional mapping between the input data X, used to describe an input pattern, to a class label Y so that Y = f(X)
- Construction of the mapping is based on **training data**
- Important division of pattern classification tasks
  - ➔ Supervised classification
    - Training data consists of training patterns and required labels
    - Identifies the input pattern as a member of a predefined class. (Descriptive)
  - ➔ unsupervised classification –
    - training data provided without labels
    - Unsupervised classification assigns the input pattern to a hitherto undefined class. (Explorative)

Definition and Background

- Process of recognizing patterns by using a Machine Learning or Deep learning Algorithm
- Analyzes incoming data and tries to identify patterns
- Classification of data
  - Based on knowledge already gained or on statistical information extracted from patterns and/or their representation
- Examples: Speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis.
- Raw data is processed and converted into a form that is amenable for a machine to use.
- Involves
  - Classification - used in supervised learning.
  - Cluster of patterns - used in unsupervised learning.
- Features
  - Function of one or more measurements, computed so that it quantifies some significant characteristics of the object
  - May be represented as continuous, discrete, or discrete binary variables
  - Example: consider our face then eyes, ears, nose, etc are features of the face.
  - A set of features that are taken together, forms the features vector.

Process of Finding Patterns in Data

- Collection of digital data
- Cleaning the data from noise
- Examining information for important features or familiar elements
- Grouping of the elements into segments
- Analysis of data sets for insights
- Implementation of the extracted insights


Repeat
Gather data (incoming or existing)
Clean the data from the noise
Look for patterns and similarities
Segment the data according to the findings
Analyze the segments
Implement insights into your business

Features of Pattern Recognition

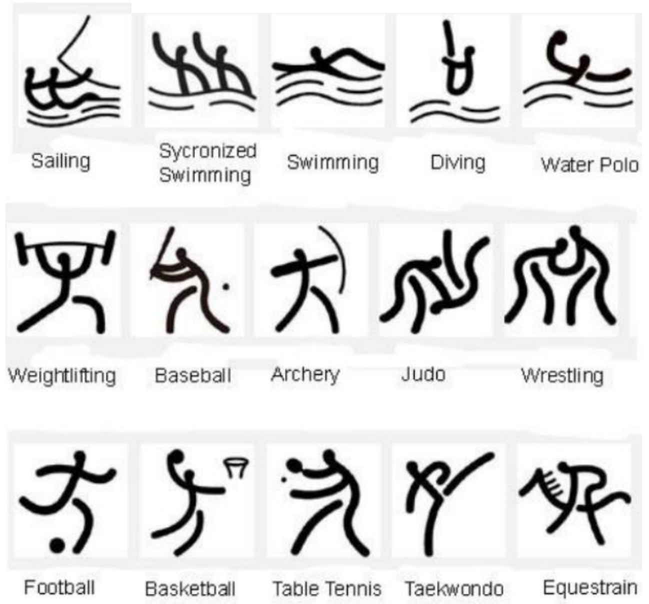
- Pattern recognition system should recognize familiar patterns quickly and accurate
- Recognize and classify unfamiliar objects
- Accurately recognize shapes and objects from different angles
- Identify patterns and objects even when partly hidden
- Recognize patterns quickly with ease, and with automaticity
- Identification of regularities in data used
  - to make predictions
  - categorize information
  - improve decision-making processes.
- Important problems
  - Automatic and machine-based recognition
  - Description
  - Classification
  - Grouping of patterns
- Application Areas-
  - Biology
  - Psychology
  - Medicine
  - Marketing
  - Computer vision and AI

Goal

- The decision-making process of a human being somewhat related to the recognition of patterns
- E.g.,
  - the next move in a chess game is based on the board’s current pattern
  - buying or selling stocks is decided by a complex pattern of financial information.
- Goal-
  - To clarify these complicated mechanisms of decision-making processes
  - To automate these e functions using computers.

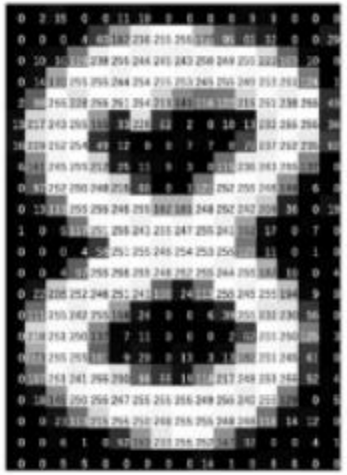
Olympic Games Symbol Recognition

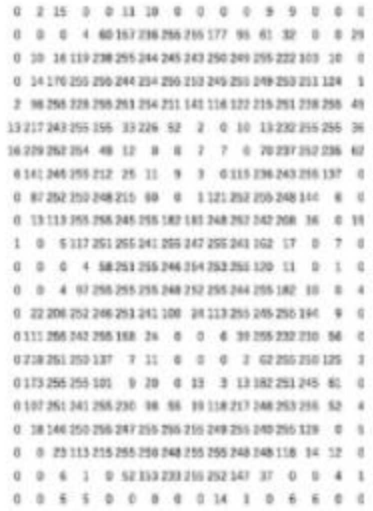
- International Olympic Committee (IOC) uses icons, flags and symbols to elevate the Olympic Games
- Include-
  - Commonly used during Olympic competition
  - Used throughout the years, such as the Olympic flag
- **Step 1- Obtain a dataset of Olympic Games symbols**
  - Gather images of Olympic Games symbols from various sources
    - Official Olympic websites
    - News articles
    - Social media.
- **Step 2 - Preprocess the images**
    - Resizing the images to a standard size
    - Converting them to grayscale or binary format
    - Removing any noise or artifacts present in the images



Olympic Games Symbol Recognition

- **Step 3- Extract features from the images**
  - Edge detection
  - Corner detection
  - Texture analysis
  - Color analysis to identify key features of the symbols





0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	232	245	255	249	253	211	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	26	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	215	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Olympic Games Symbol Recognition

- **Step 4 – Training**
- To classify the symbols based on their features
  - support vector machines (SVMs)
  - Random forests
  - Neural networks
- **Step 5 – Testing**
- Finding Accuracy in recognizing symbols

Goal-Olympic Games Symbol Recognition

- many different symbols
  - that represent different sports, countries, and events
  - Can vary in size, color, and style
- Without pattern classification, it would be very difficult to manually identify and classify each symbol, especially if the dataset is large.
- Allows us to automatically identify and categorize images based on their visual features
- Allows for scalability and adaptability
- Dataset of Olympic Games symbols grows over time, the pattern classification algorithm can be retrained with new data to improve its accuracy
- Same applications -
  - Medical imaging
  - Surveillance
  - Robotics

Recognition of Printed Characters

- Process to perform electronic conversion of the text on a physical paper.
- The text on the document can be either machine print or handwritten.
- Note: If the physical paper has typed text then chances are high for an exact match. But if it has handwritten text, it will use artificial intelligence algorithm to find a corresponding character.
- Machine printed character recognition
  - uses intelligent document recognition technology
  - To read the pattern of the text on the physical paper
  - Convert them accurately in digital format.
- Increases efficiency and work productivity like never before.
- Plays the major part in digitization of companies and making workplaces more efficient

Recognition of Printed Characters

- **Step 1 - Image acquisition**
  - To obtain a digital image of the printed document or text
  - Can be done using a scanner or a camera.
- **Step 2 – Preprocessing**
  - The acquired image may contain noise or other artifacts that can interfere with character recognition
  - Preprocessing techniques such as filtering, binarization, and noise reduction
  - Can help improve the quality of the image and make it more suitable for analysis.
- **Step 3 -Segmentation**
  - The image is divided into smaller regions, each containing a single character
  - To make it easier to recognize each character individually.
  - Techniques such as thresholding and contour detection can be used for segmentation.

Recognition of Printed Characters

- **Step 4 - Feature extraction**
    - Each segmented character is analyzed to identify its unique features, such as shape, size, and orientation.
    - Can be done using techniques such as edge detection, morphological operations, and texture analysis.
- **Step 5 - Classification**
    - To match the features to known characters and recognize them.
    - Popular classification algorithms used in OCR include Support Vector Machines (SVMs), Random Forests, and Convolutional Neural Networks (CNNs).
- **Step 6 – Post-processing**
    - The recognized characters may contain errors, such as misrecognitions or false positives.
    - Post-processing techniques -
        - spell-checking and dictionary lookup
        - Can be used to correct these errors and improve the accuracy of the recognition result

Neocognitron

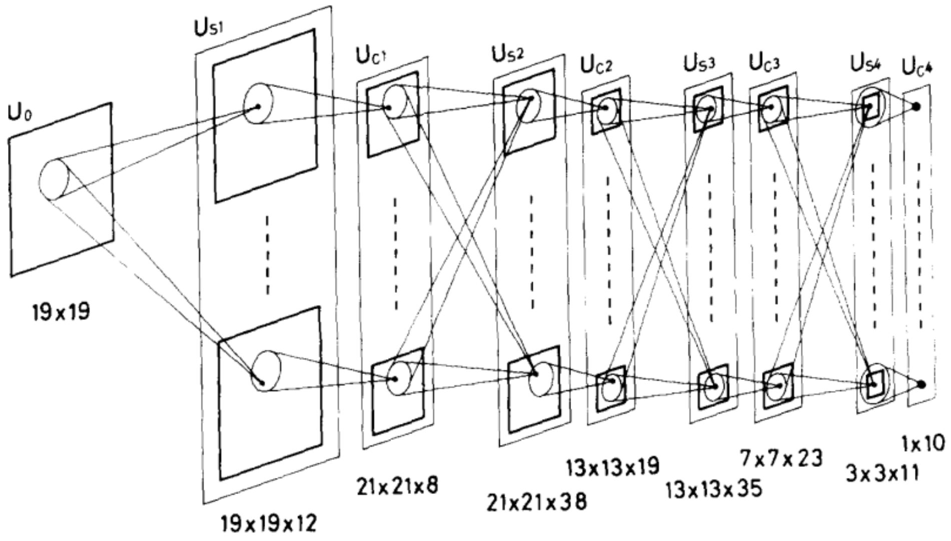
- Visual pattern recognition, such as reading characters or distinguishing shapes, can easily be done by human beings,
- But very difficult to design a machine which can do it as well as human beings do.
- Best strategy is to learn from the brain itself
- Study on - how to synthesize a neural network model which has the same ability as the human brain.
- As a result of this approach, Hierarchical, multilayered artificial neural network proposed by Kunihiko Fukushima in 1979

Neocognitron

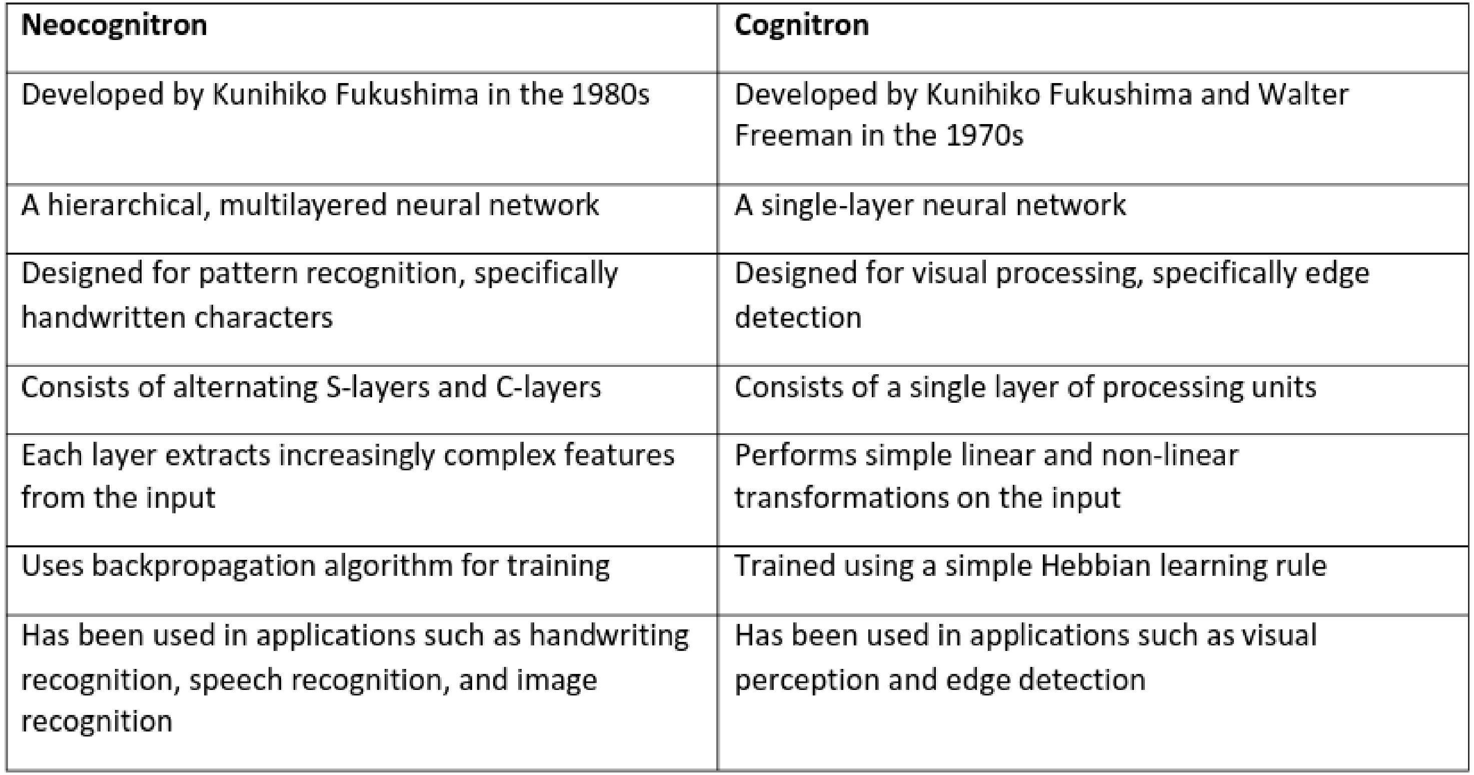
- Hierarchical network consisting of many layers of cells, and has variable connections between the cells in adjoining layers.
- Can acquire the ability to recognize patterns by learning, and can be trained to recognize any set of patterns.
- After finishing the process of learning, pattern recognition is performed on the basis of similarity in shape between patterns
- Not affected by deformation, nor by changes in size, nor by shifts in the position of the input patterns.
- In the hierarchical network of the neocognitron
    - Local features of the input pattern are extracted by the cells of a lower stage
    - Gradually integrated into more global features
    - Each cell of the highest stage integrates all the information of the input pattern
    - Shows the final result of the pattern-recognition of the network

Neocognitron

- Cascade of many layers of neuron-like cells
- Cells are of the analog type
- Forward connections between cells in adjoining layers.
- Initial stage of the network is the input layer, called U0,consists of a two-dimensional array of receptor cells uo.
- Succeeding stages has a layer of"S-cells" followed by a layer of "'C-cell
- S-celts and C-cells are arranged alternately
- S-celts are feature-extracting cells and C-cells are inserted in the network to allow for positional errors in the features of the stimulus.



Neocognitron



Neocognitron	Cognitron
Developed by Kunihiko Fukushima in the 1980s	Developed by Kunihiko Fukushima and Walter Freeman in the 1970s
A hierarchical, multilayered neural network	A single-layer neural network
Designed for pattern recognition, specifically handwritten characters	Designed for visual processing, specifically edge detection
Consists of alternating S-layers and C-layers	Consists of a single layer of processing units
Each layer extracts increasingly complex features from the input	Performs simple linear and non-linear transformations on the input
Uses backpropagation algorithm for training	Trained using a simple Hebbian learning rule
Has been used in applications such as handwriting recognition, speech recognition, and image recognition	Has been used in applications such as visual perception and edge detection

Neocognitron

- **S-cells (Simple cells)** work as feature-extracting cells.
    - Resemble simple cells of the primary visual cortex in their response.
    - Their input connections are variable and are modified through learning.
    - After having finished learning, each S-cell come to respond selectively to a particular feature presented in its receptive field.
    - The features extracted by S-cells are determined during the learning process.
    - Generally speaking, local features, such as edges or lines in particular orientations, are extracted in lower stages.
    - More global features, such as parts of learning patterns, are extracted in higher stage

Neocognitron

- **C-cells** (Complex cells),
  - Resembles complex cells in the visual cortex,
  - Inserted in the network to allow for positional errors in the features of the stimulus.
  - The input connections of C-cells, which come from S-cells of the preceding layer, are fixed and invariable.
  - Each C-cell receives excitatory input connections from a group of S-cells that extract the same feature, but from slightly different positions.
  - The C-cell responds if at least one of these S-cells yield an output.
  - Even if the stimulus feature shifts in position and another S-cell comes to respond instead of the first one, the same C-cell keeps responding.
  - Thus, the C-cell's response is less sensitive to shift in position of the input pattern.
  - We can also express that C-cells make a blurring operation, because the response of a layer of S-cells is spatially blurred in the response of the succeeding layer of C-cell

Recognition of Handwritten characters using Neocognitron

- **Preprocessing**:
  - Preprocess the image of the handwritten character.
  - Such as noise reduction, normalization, and smoothing
  - To improve the quality of the image and make it more suitable for analysis.
- **Feature extraction**:
  - Analyse Image to extract features that are relevant for character recognition.
  - For example, features such as stroke direction, curvature, and line thickness
  - Can be extracted using techniques such as edge detection, contour analysis, and Hough transforms. T
- **Training**:
  - Train Neocognitron network using a set of training examples.
  - During training, the network adjusts its weights and biases to minimize the difference between its output and the desired output.
- **Testing**:
  - Test network using a set of test examples that it has not seen before.
  - The performance of the network is evaluated based on its ability to correctly recognize the characters in the test set.
- **Post-processing**:
  - The recognized characters may contain errors, such as misrecognitions or false positives.
  - Post-processing techniques such as spell-checking and dictionary lookup

NET Talk: to convert English text to speech

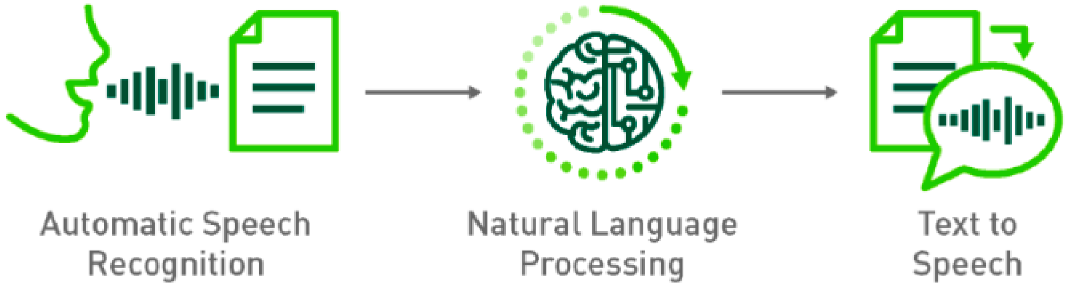
- Text to speech is also called as Neural Text to Speech.
- How general Text to Speech works? -
  - First, the speech engines take the audio input and recognize sound waves produced by a human voice.
  - This information is then translated into language data, which is called automatic speech recognition (ASR).
  - After acquiring this data, it must then analyze it to understand the meaning of the words it has collected, which is called natural-language generation (NLG).
  - AI has advanced to the point where it can understand human communication to the point where it can determine the appropriate response.
  - AI does this by analyzing a large volume of human speech. -
  - After understanding the context of the text response, it produces the necessary speech sounds or phonemes.

NET Talk: to convert English text to speech

- NET Talk: NET Talk is a neural network-based speech synthesizer
- Developed in the 1980s by Terry Sejnowski and Charles Rosenberg.
- Designed to convert English text into speech, and it is based on a type of neural network called a time-delay neural network (TDNN)
- Consists of two main components:
    - Training module
    - Synthesis module.

NET Talk: to convert English text to speech

- Training of system using a large corpus of English text and corresponding speech recordings.
- Use of training data to adjust the weights and biases of the TDNN (time-delay neural network),
- So that it can accurately predict the acoustic features of the speech given the input text.
- Once the TDNN has been trained, it can be used to synthesize speech from new input text.
- The synthesis module takes a text input and converts it into a sequence of acoustic features, which are then used to generate the corresponding speech waveform



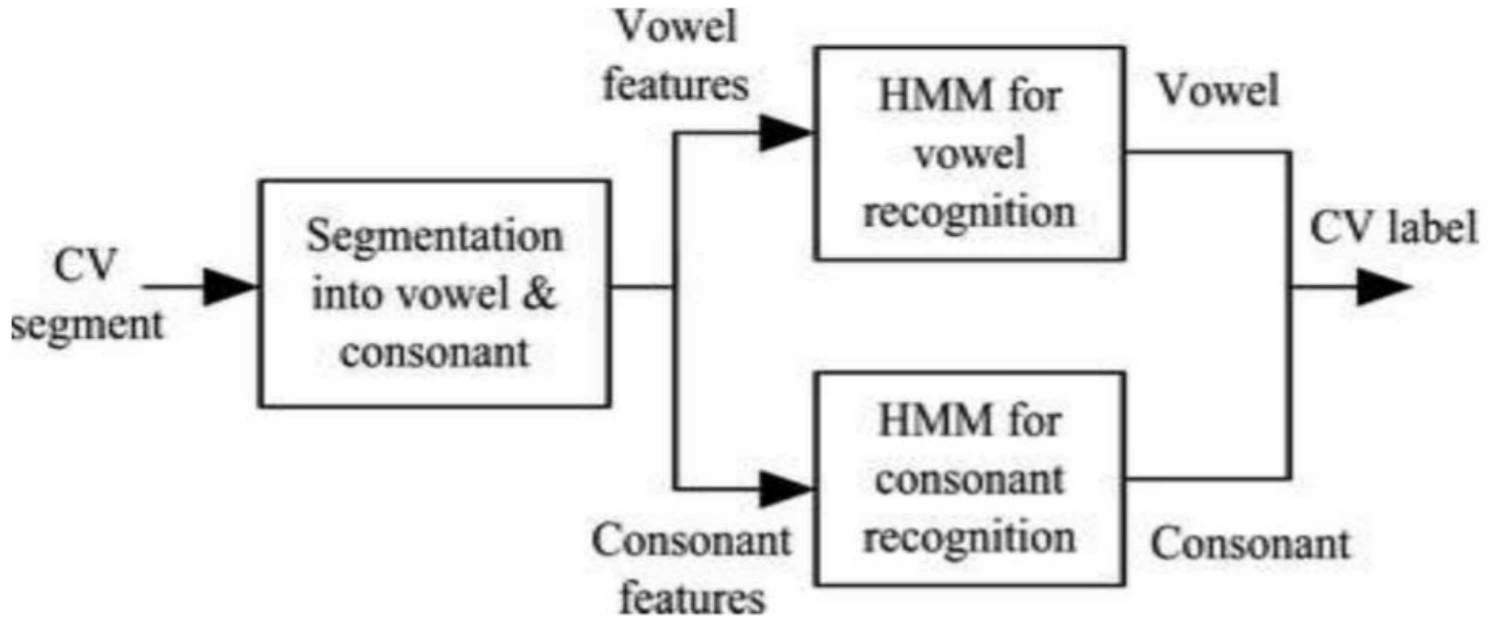
Recognition of consonant vowel (CV) segments

- Segment: section or part of something
- Combination of Consonant and Vowel creates letters in English (or any other language)
- Vowel: A, E, I, O, U
- **In text or speech recognition, why differently recognition of consonant and vowel is required?**
  - Text and speech recognition systems differ in the way they recognize consonants and vowels
  - Because consonants and vowels have different acoustic properties that affect their perceptual distinctiveness and recognition.
  - Due to differences in acoustic properties, recognizing consonants and vowels requires different signal processing techniques and machine learning models.
  - Different feature extraction techniques and acoustic models are used

Recognition of consonant vowel (CV) segments

- **Consonants** -
  - Produced by obstructing or constricting the airflow in the vocal tract, creating noise or turbulence in the sound signal.
  - Generally shorter in duration and have a faster changing spectrum compared to vowels.
  - Moreover, consonants are produced by different articulators in the vocal tract, such as lips, tongue, and teeth, which affects their acoustic properties.
  - Consonants: B, C, D, F, G, J, K, L, M, N, P, Q, S, T, V, X, Z and often H, R, W, Y
- **Vowels** -
  - Produced by maintaining a relatively stable and unobstructed airflow in the vocal tract, resulting in a more tonal or harmonic sound signal.
  - Generally longer in duration and have a slower changing spectrum compared to consonants.
  - Moreover, vowels are produced by the position of the tongue and the shape of the lips, which affects their acoustic properties.

Recognition of consonant vowel (CV) segments


Vowel features
HMM for vowel recognition
Vowel
CV segment
Segmentation into vowel & consonant
CV label
HMM for consonant recognition
Consonant features
Consonant

Recognition of consonant vowel (CV) segments

Aspect	**HMM for Vowel Recognition**	**HMM for Consonant Recognition**
Acoustic Model	Typically uses a single Gaussian distribution per state due to the simplicity of vowel sounds.	Uses multiple Gaussian distributions per state to model the more complex and diverse range of consonant sounds.
Training Data	Requires a large amount of training data for each vowel sound, as the acoustic features that distinguish vowels from each other are subtle.	Requires a smaller amount of training data for each consonant sound, as the acoustic features that distinguish consonants from each other are more pronounced.
State Topology	Typically uses a simple state topology with fewer states due to the relatively uniform spectral properties of vowels.	Uses a more complex state topology with more states to capture the rapid spectral changes in consonant sounds.
Recognition Accuracy	Achieves high recognition accuracy for vowel sounds due to the clear acoustic differences between vowels.	Achieves lower recognition accuracy for consonant sounds due to the more complex and diverse range of consonant sounds.
Application	Used in speech recognition tasks where the focus is on recognizing spoken words or phrases.	Used in tasks that require recognition of individual phonemes in spoken language, such as in phonetic transcription or speech therapy.

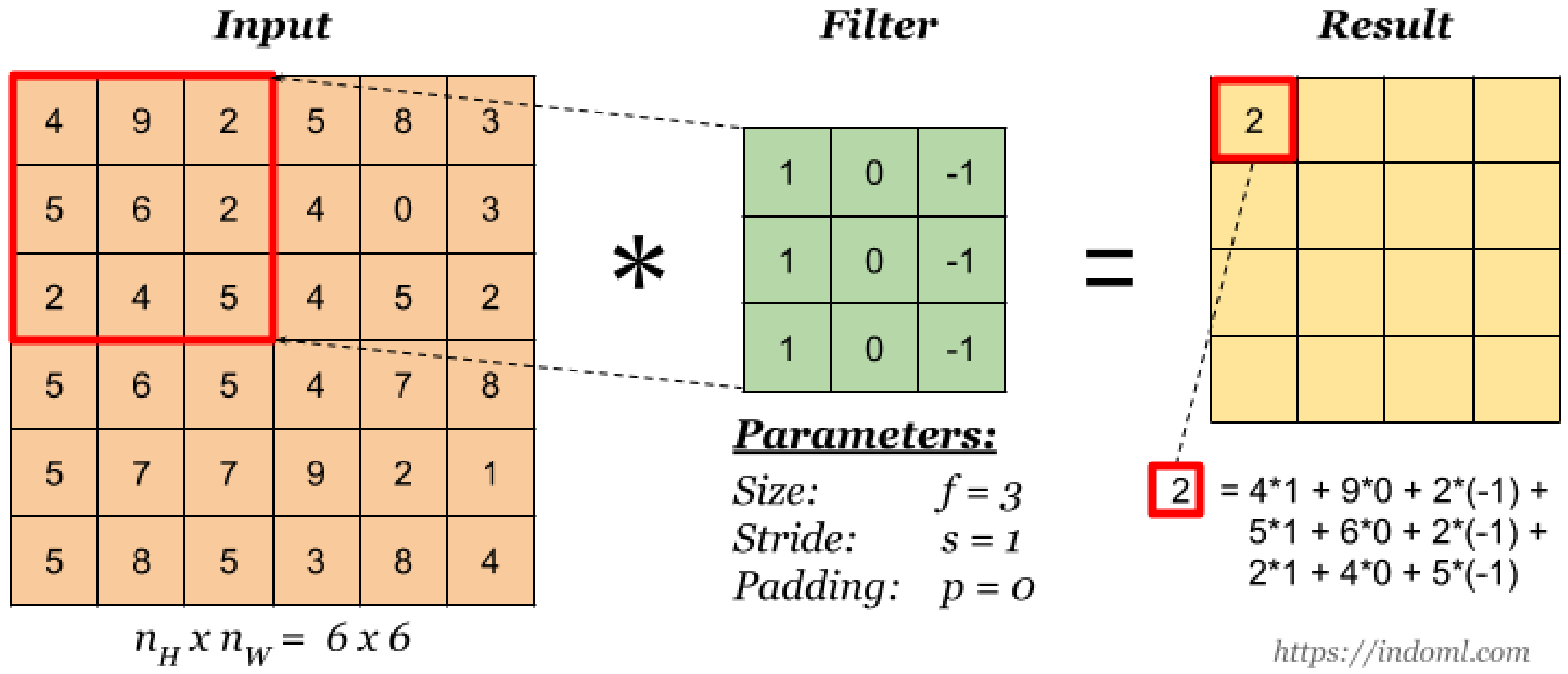
Recognition of consonant vowel (CV) segments

- Hidden Markov Models (HMMs) are commonly used for speech recognition tasks, including vowel recognition and consonant recognition.
- **<u>How HMMs work for each task</u>:**
- **1. HMM for Vowel Recognition**
    - Train the model with a large set of labeled training data, consisting of audio recordings of different vowel sounds.
    - HMM is trained to learn the statistical properties of each vowel sound, by estimating the probability distribution of the spectral features of each vowel sound
    - Once the HMM is trained, can be used to recognize vowel sounds in new speech signals.
    - This involves breaking the speech signal into small time frames and extracting the spectral features of each frame.
    - The HMM then calculates the probability that each frame of the speech signal belongs to each vowel sound.
    - By combining the probabilities of each frame, the HMM produces a final estimate of the most likely vowel soun

Recognition of consonant vowel (CV) segments

- Hidden Markov Models (HMMs) are commonly used for speech recognition tasks, including vowel recognition and consonant recognition.
- <u>**How HMMs work for each task**</u>**:**
- **2. HMM for Consonant Recognition**
    - Consonant sounds more complex and diverse than vowel sounds
    - Require a more complex model to capture the variability of the acoustic features that distinguish different consonants
    - Train the model with a large set of labeled training data, consisting of audio recordings of different consonant sounds.
    - Learn the statistical properties of each consonant sound, by estimating the probability distribution of the spectral features of each consonant sound
    - Once the HMM is trained, it can be used to recognize consonant sounds in new speech signals
    - Same as Vowel Recognition

Basic Convolution Operation – Step 1



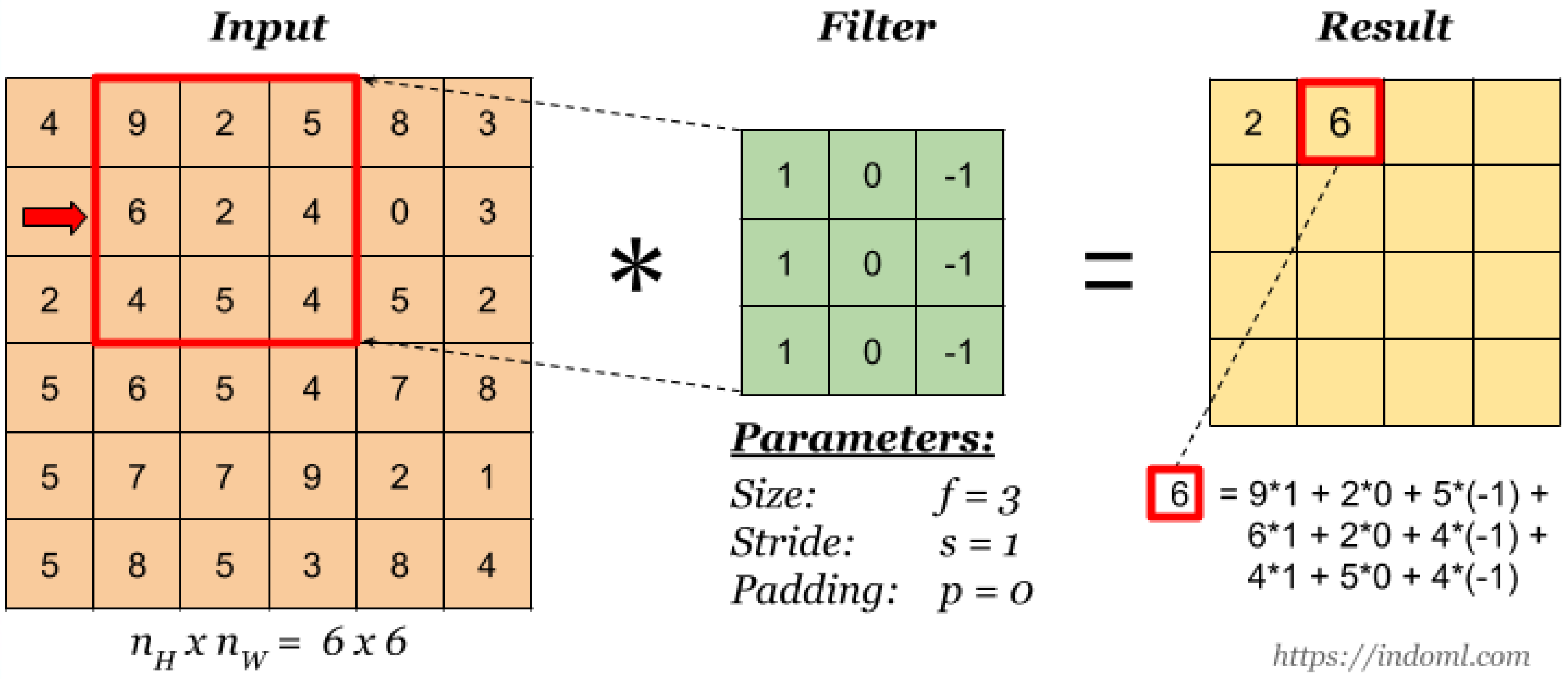
Parameters:

Size:	$f = 3$
Stride:	$s = 1$
Padding:	$p = 0$

$\boxed{2} = 4*1 + 9*0 + 2*(-1) + 5*1 + 6*0 + 2*(-1) + 2*1 + 4*0 + 5*(-1)$

https://indoml.com

Basic Convolution Operation – Step 2



$$
\begin{bmatrix} 4 & 9 & 2 & 5 & 8 & 3 \\ 6 & 6 & 2 & 4 & 0 & 3 \\ 2 & 4 & 5 & 4 & 5 & 2 \\ 5 & 6 & 5 & 4 & 7 & 8 \\ 5 & 7 & 7 & 9 & 2 & 1 \\ 5 & 8 & 5 & 3 & 8 & 4 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}
=
\begin{bmatrix} 2 & \boxed{6} & & \\ & & & \\ & & & \\ & & & \end{bmatrix}
$$

$n_H \, x \, n_W = 6 \, x \, 6$

Parameters:

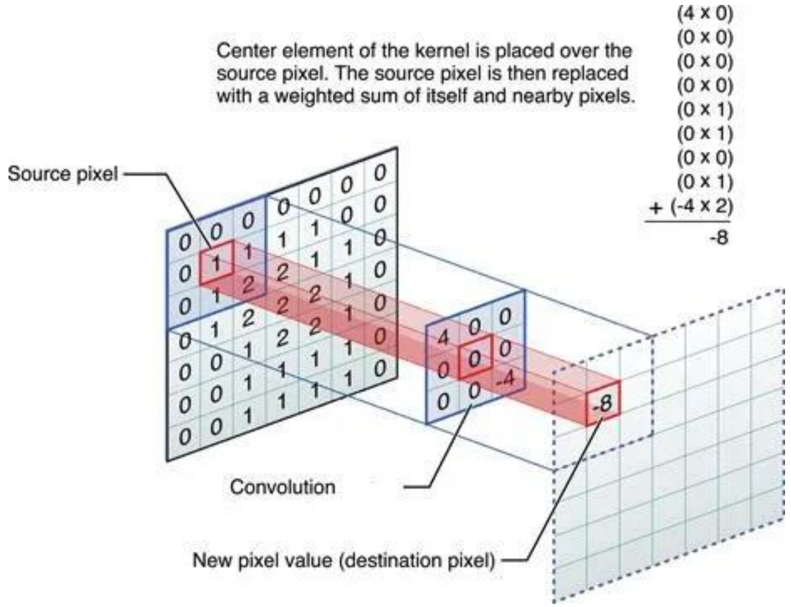
Size:	$f = 3$
Stride:	$s = 1$
Padding:	$p = 0$

$\boxed{6}$ = 9*1 + 2*0 + 5*(-1) +
6*1 + 2*0 + 4*(-1) +
4*1 + 5*0 + 4*(-1)

https://indoml.com

Convolution Operation

- Kernel overlaps few pixels at each position on the image
- Each pixel which is overlapped is multiplied and then added.
- And the sum is set as the value of the current position.
- This process is repeated across the entire image.
- Process in which each element of the image is added to its local neighbors, and then it is weighted by the kernel

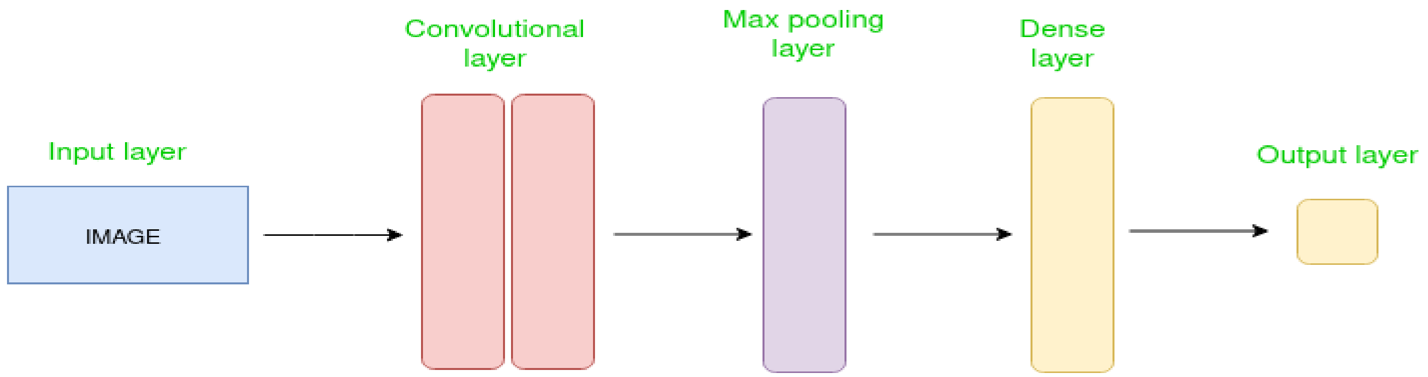


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution
New pixel value (destination pixel)

CNN comprises -:

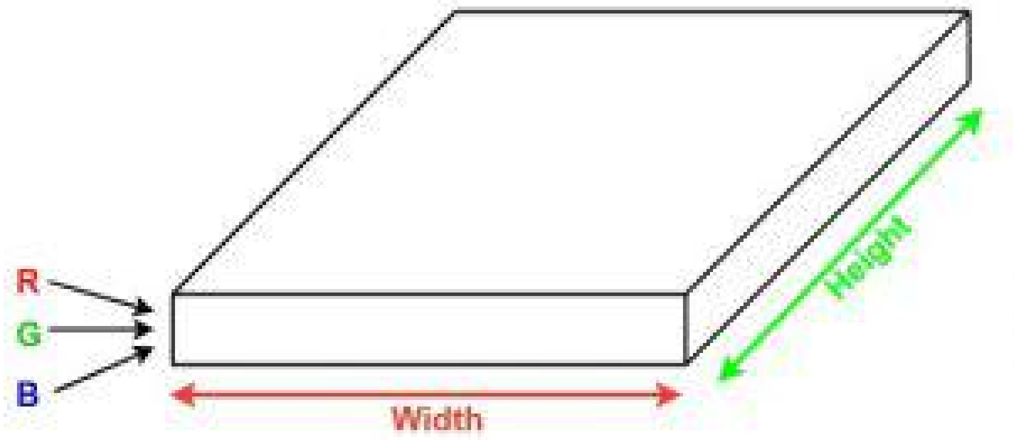
- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer.

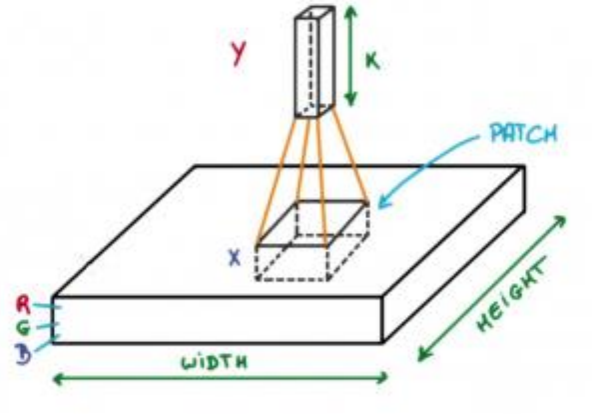
Simple CNN Architecture



- Layers-
  - **Input Layer**
  - **Convolution layer** - applies filters to the input image to extract features
  - **Pooling Layer** - downsamples the image to reduce computation
  - **Fully Connected Layer** - makes the final prediction.

How Convolution Llayers Work





- Convolution layers consist of a set of learnable filters (or kernels)
- having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).

  e. g. Run Convolution on image with dimension 34x34x3.

  -- > possible size of filters can be axax3,

  ‘a’ -> anything like 3, 5, or 7 but smaller as compared to the image dimension.
- Slide filter with stride

How Convolution Layers Work

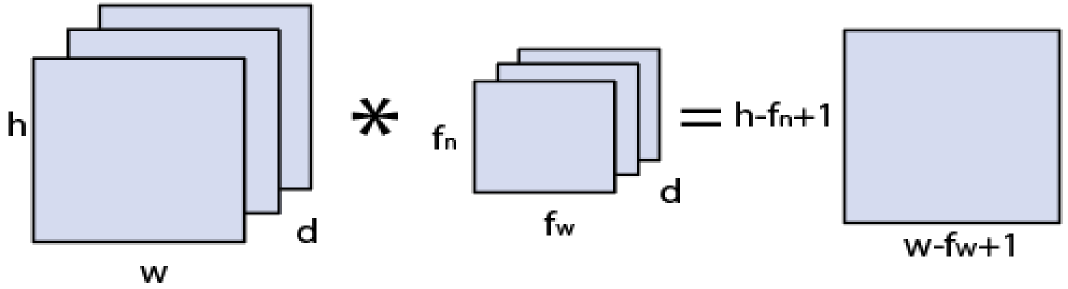


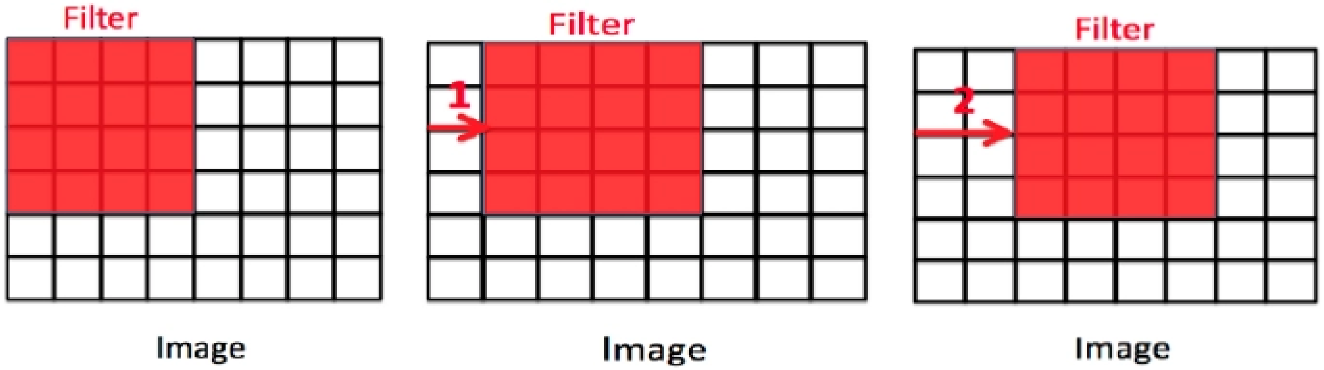
Image matrix multiplies kernl or filter matrix

The image matrix is h x w x d

The dimensions of the filter are fh x fw x d

The output is calculated as (h- fh +1)(w- fw+1) x 1

Stride

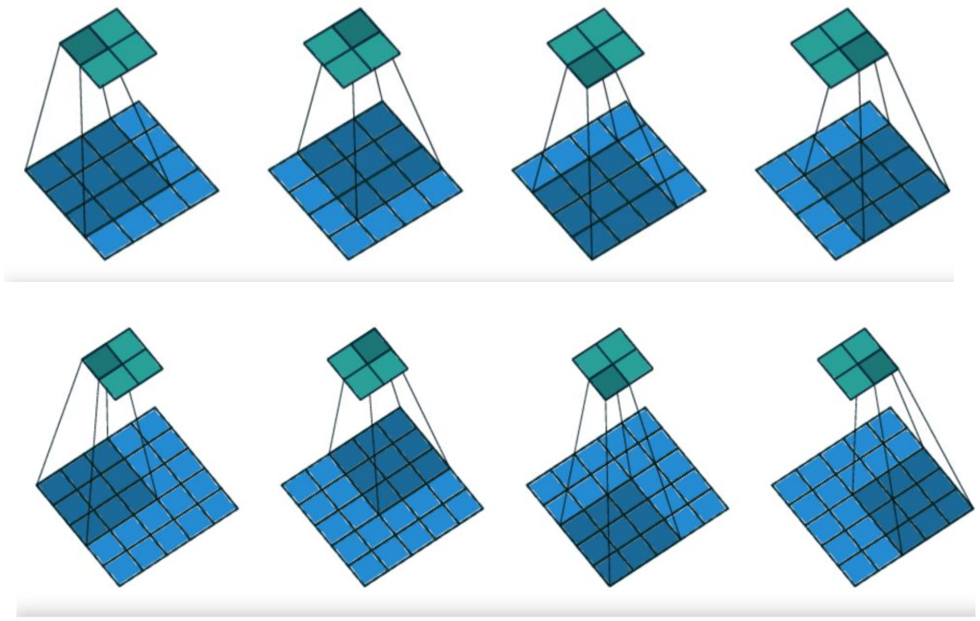


- When the array created, the pixels shifted over to the input matrix
- Definition- The number of pixels turning to the input matrix
- When the number of strides 1, we move the filters to 1 pixel at a time an so on
- Essential because they control the convolution of the filter against the input
- Strides are responsible for regulating the features that could be missed while flattening the image.
- Denote the number of steps we are moving in each convolution

  n the first matrix, the stride = 0, second image: stride=1, and the third image: stride=2.
- The size of the output image is calculated by:

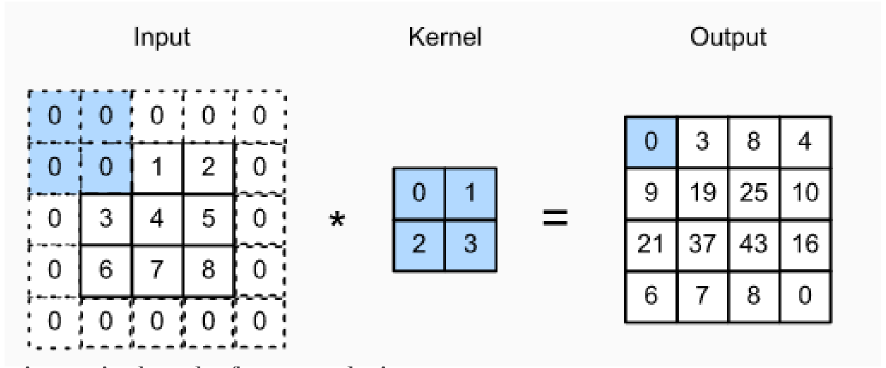
[{(n+2p-f+1)/s}+1][{(n+2p-f+1)/s}]

Stride Convolution



- Stride is a parameter that dictates the movement of the kernel, or filter, across the input data, such as an image
- The stride determines how many units the filter shifts at each step, while performing convolution operation
- Shift can be
  - Horizontal
  - Vertical
  - or both
  - depending on the stride's configuration.

Padding



- Original size of the image is shrunk after convolution
- Two Problems:-
  1. Also, in the image classification task, multiple convolution layers after which our original image is shrunk after every step, which we don't want.
  2. When the kernel moves over the original image, it passes through the middle layer more times than the edge layers, due to which there occurs an overlap
- To overcome this, "Padding" introduced
- Additional layer that can add to the borders of an image
- While preserving the size of the original picture
- if an n x n matrix is convolved with an ff matrix with a padding p, then the size of the output image will be:
  - (n+2p-f+1) x (n+2p-f+1)

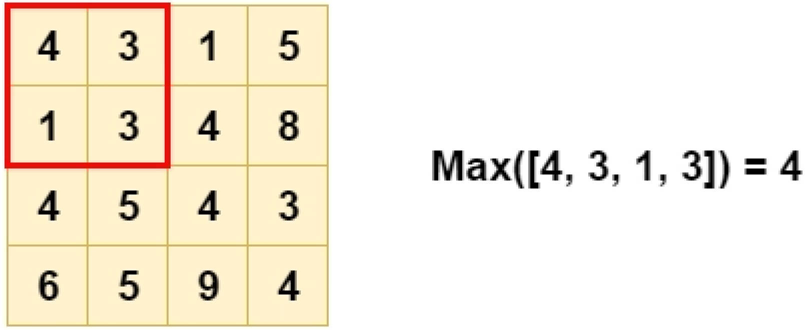
Pooling

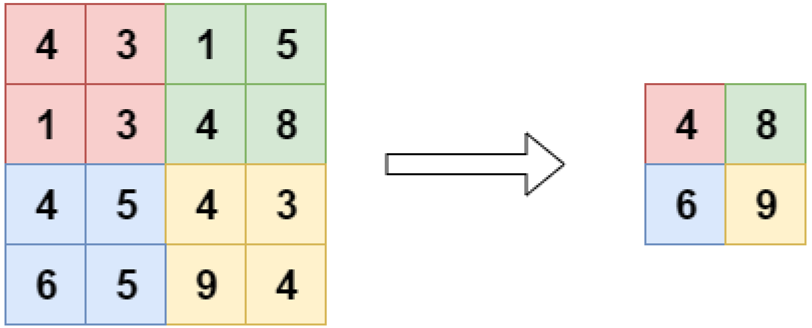
- In the pre-process, the image size shrinks by reducing the number of parameters if the image is too large
- Picture is shrunk, the pixel density is also reduced,
- the downscaled image is obtained from the previous layers.
- Progressively reduce the spatial size of the image to reduce the network complexity and computational cost
- Spatial pooling --> downsampling or subsampling that reduces the dimensionality of each map but retains the essential features
- ReLU, is applied to each value in the feature map
- Pooling is added after the nonlinearity is applied to the feature maps
- Three Types -:

1) Max Pooling
2) Average pooling
3) Sum Pooling

Max pooling

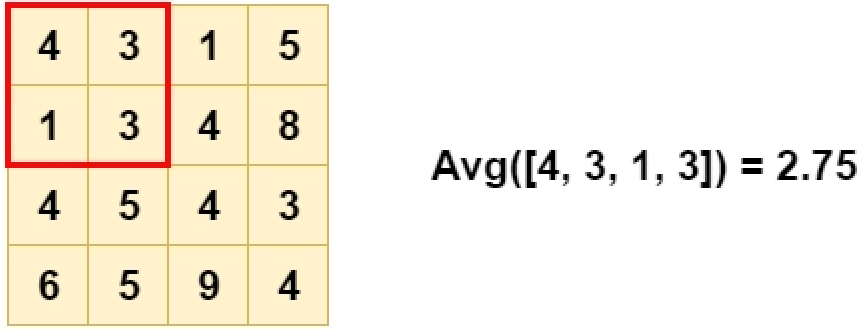
- Rule to take the maximum of a region
- Transfers continuous functions into discrete counterparts
- primary objective
  - ➔ To downscale an input by reducing its dimensionality
  - ➔ making assumptions about features contained in the sub-region that were rejected.

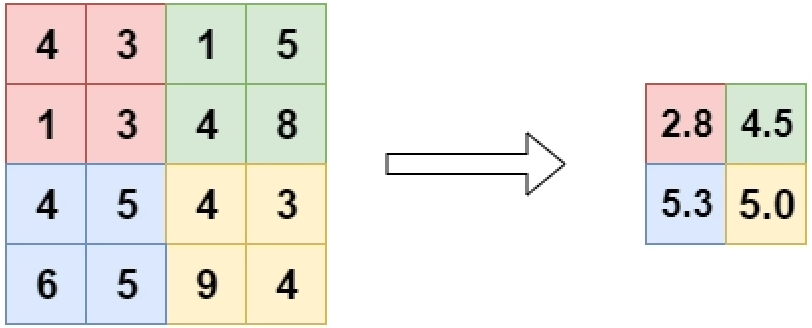




Average pooling

- Retains information about the lesser essential features
- Downscales
  - by dividing the input matrix into rectangular regions and
  - calculating the average values of each area.
- primary objective
  - To downscale an input by reducing its dimensionality
  - making assumptions about features contained in the sub-region that were rejected.



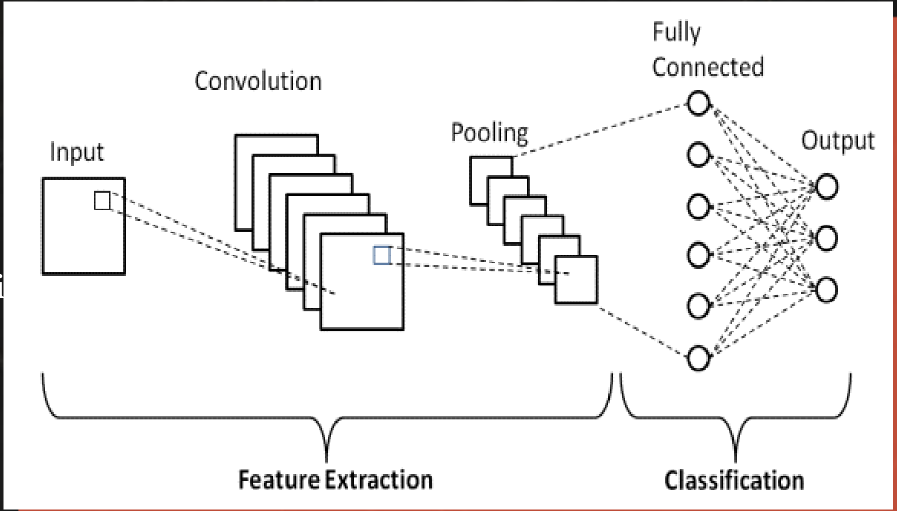


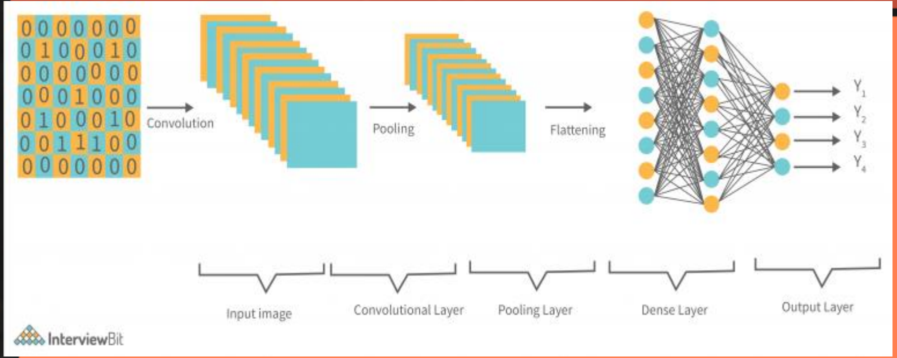
Pooling Layers

Pooling Type	Operation	Description	Advantages	Disadvantages
Max Pooling	Max operation	Takes the maximum value in each pooling region	Preserves strong features, provides invariance to small translations	May discard some information, not suitable for capturing average or global statistics
Average Pooling	Average operation	Computes the average value in each pooling region	Smoothes features, retains overall trends	May blur or lose details in the presence of extreme values
Sum Pooling	Summation operation	Computes the sum of values in each pooling region	Captures total activation, suitable for preserving intensity information	Sensitive to noise, less robust to outliers

Basic CNN Architecture

- Two main parts to a CNN architecture
  - **Feature Extraction -**
    1) A convolution tool that separates and identifies the various features of the image for analysis
    2) Aims to reduce the number of features present i a dataset
    3) Creates new features which summarises the existing features contained in an original set of features
  - **Classification -**
    1) Fully connected layer that utilizes the output from the convolution process
    2) Predicts the class of the image based on the features extracted in previous stages





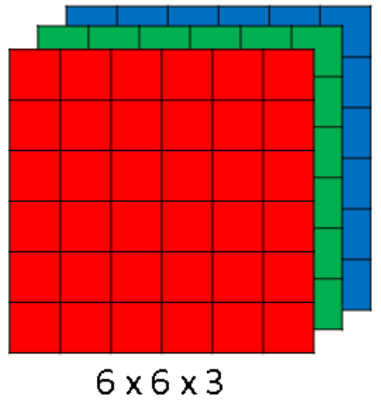
Convolution Layers

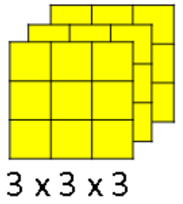
Layer Type	Description	Parameters	Advantages	Use Cases
Convolutional Layer	Applies convolution operation for feature extraction	Kernel size, Stride, Padding	Local feature learning, spatial hierarchies	Image classification, object detection, feature extraction
Pooling Layer	Reduces spatial dimensions, extracts dominant features	Pool size, Stride	Downsampling, translation invariance	Image classification, spatial hierarchy learning
Fully Connected Layer	Connects every neuron to every neuron in the previous layer	Number of neurons	Global patterns, parameter sharing	Image classification, final decision making
Batch Normalization	Normalizes and scales input within a mini-batch	Momentum, Epsilon	Faster training, reduced internal covariate shift	Improved convergence, regularization
Dropout Layer	Randomly drops neurons during training	Dropout rate	Regularization, prevents overfitting	Generalization, robustness
Activation Layer	Applies a non-linear activation function	Activation function (e.g., ReLU, Sigmoid)	Introduces non-linearity, enables learning complex patterns	Non-linearity introduction
Flatten Layer	Converts multi-dimensional data to a flat vector	-	Prepares input for fully connected layers	Transition from convolutional to fully connected layers

Advantages of CNN Architecture

- Computationally efficient.
- Performs parameter sharing and uses special convolution and pooling algorithms.
- CNN models may now run on any device, making them globally appealing.
- Finds the relevant features without the need for human intervention.
- Can be utilized in a variety of industries to execute key tasks such as facial recognition, document analysis, climate comprehension, image recognition, and item identification, among others.
- By feeding your data on each level and tuning the CNN a little for a specific purpose, you can extract valuable features from an already trained CNN with its taught weights.

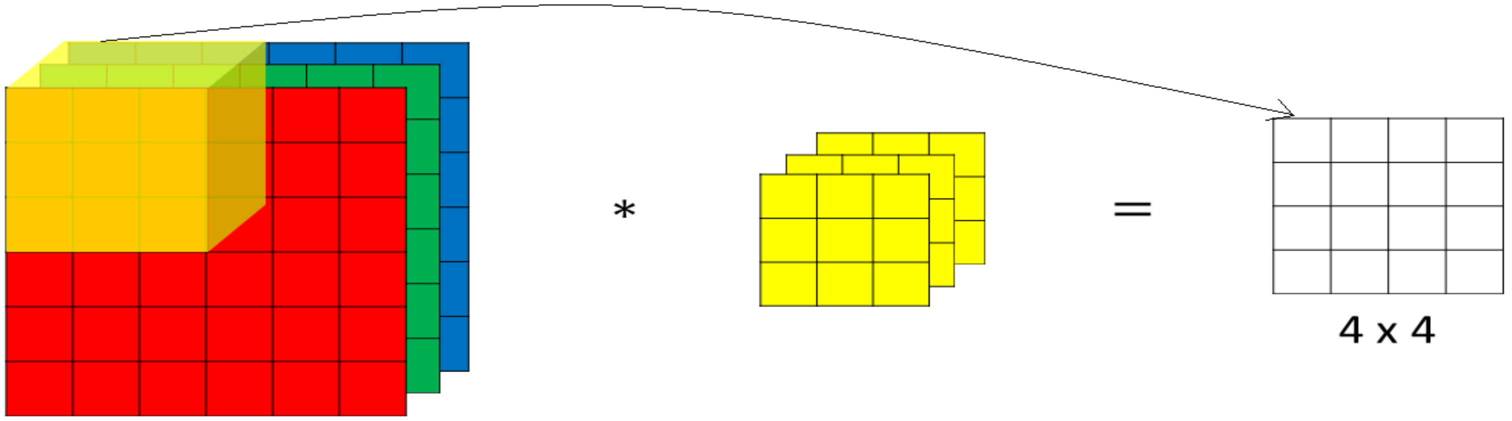
Convolution Over Volume



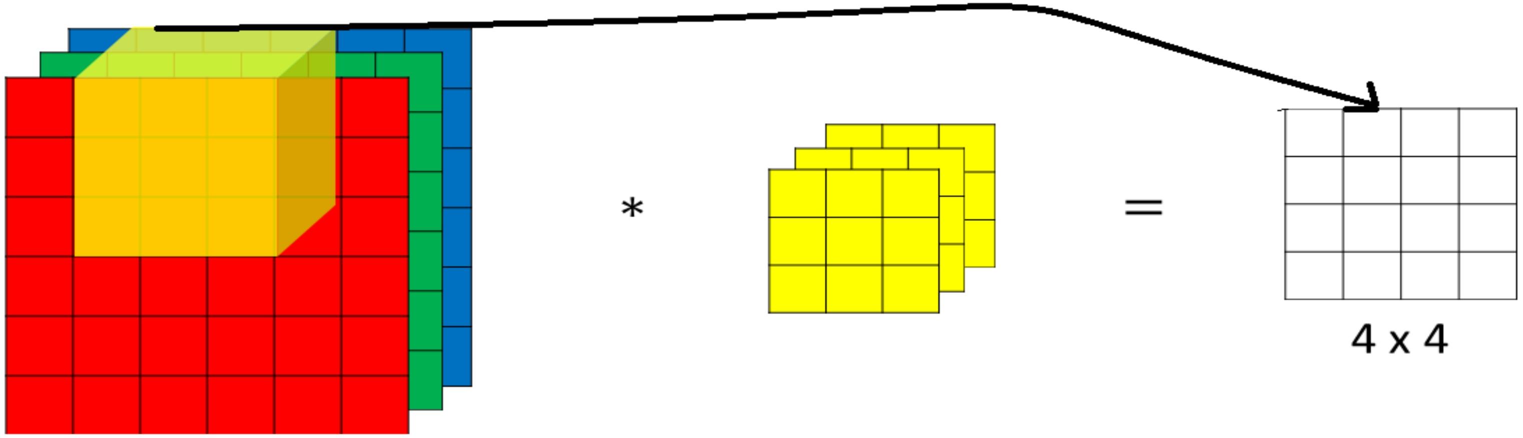


- Convolution on RGB Image
- Input data is no more 2D but in fact 3D
- Image dimension n x n x # channels
- Filter dimension f x f x # channels

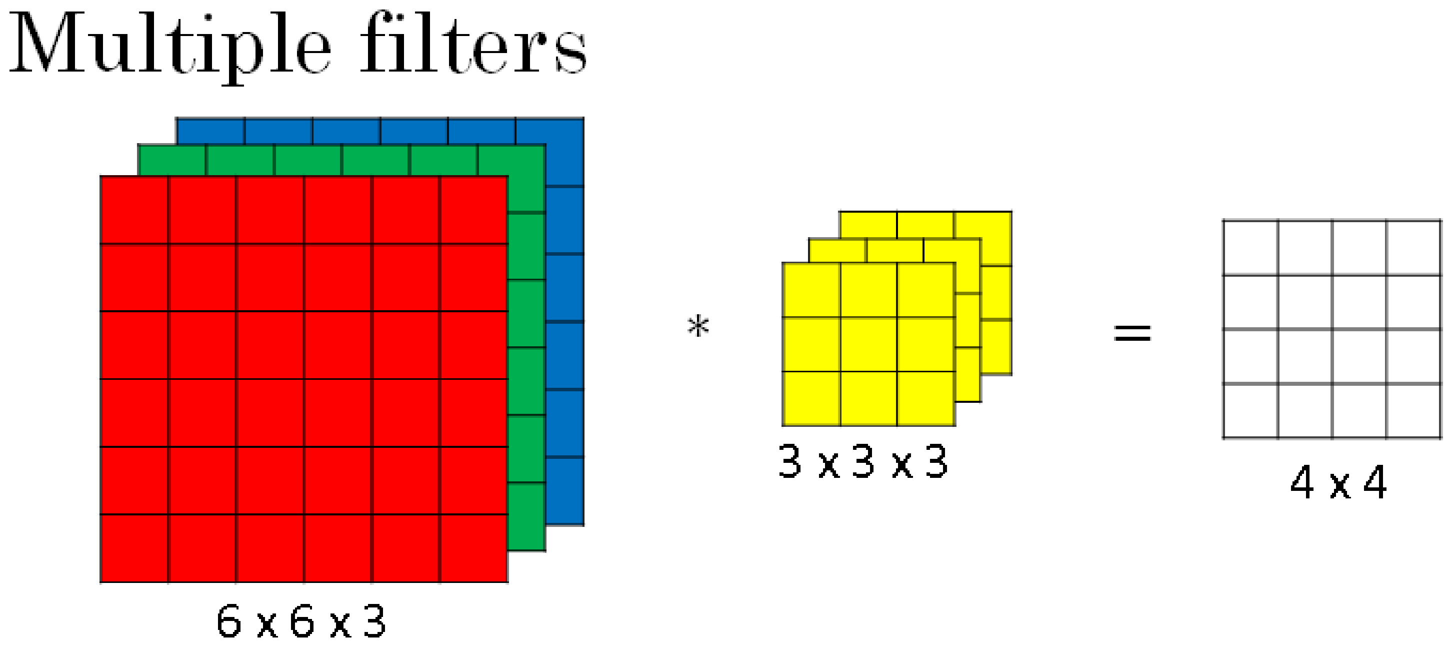
Convolutions on RGB image



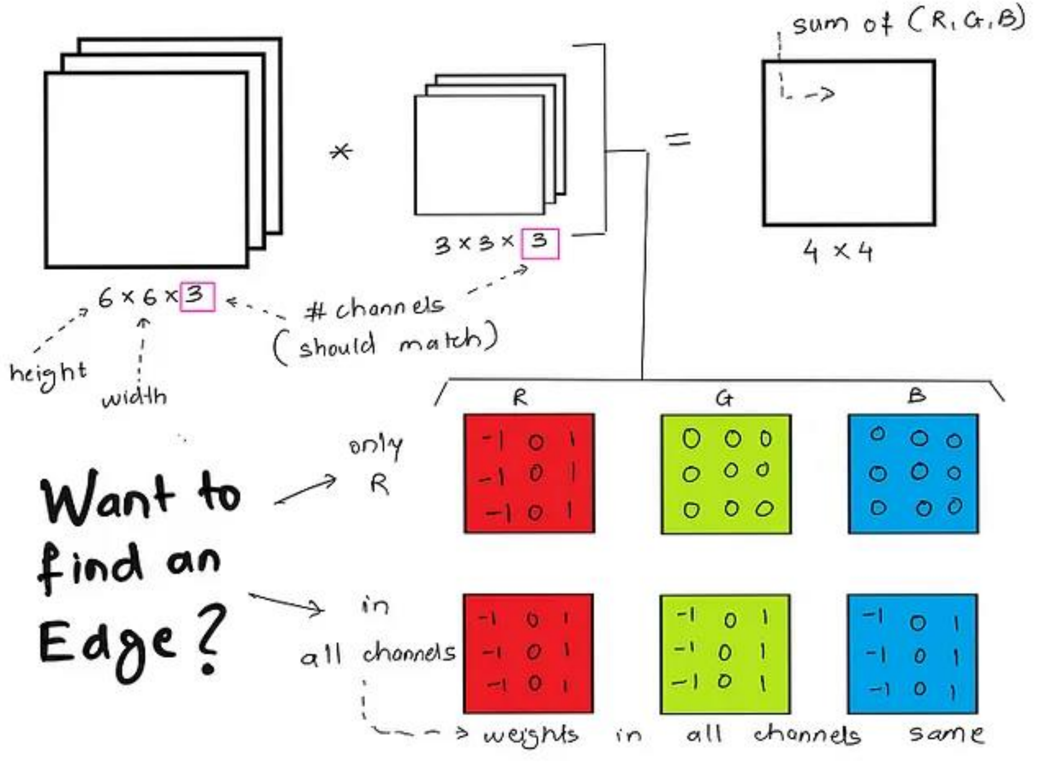
Convolutions on RGB image



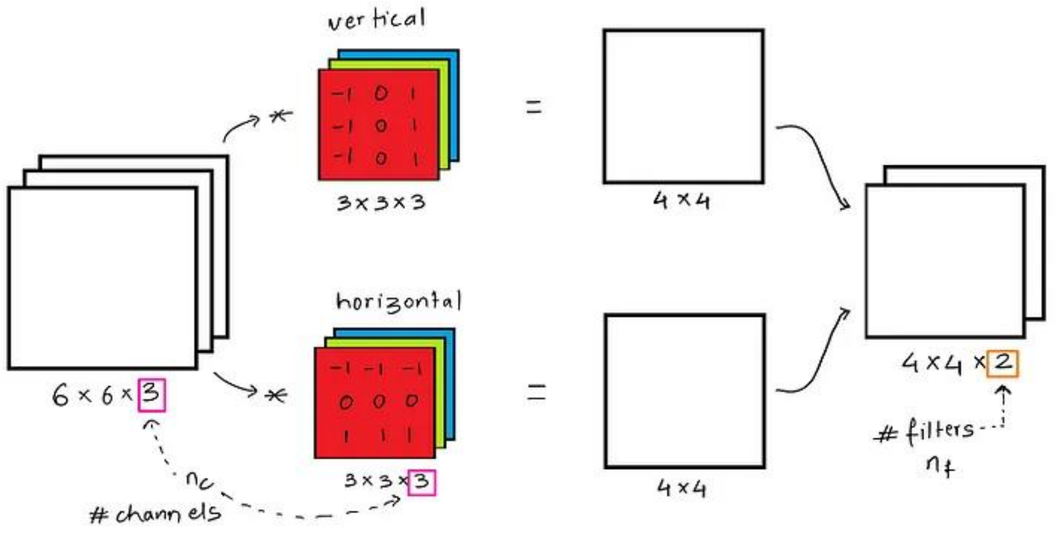
Convolution Over Volume



Convolution Over Volume



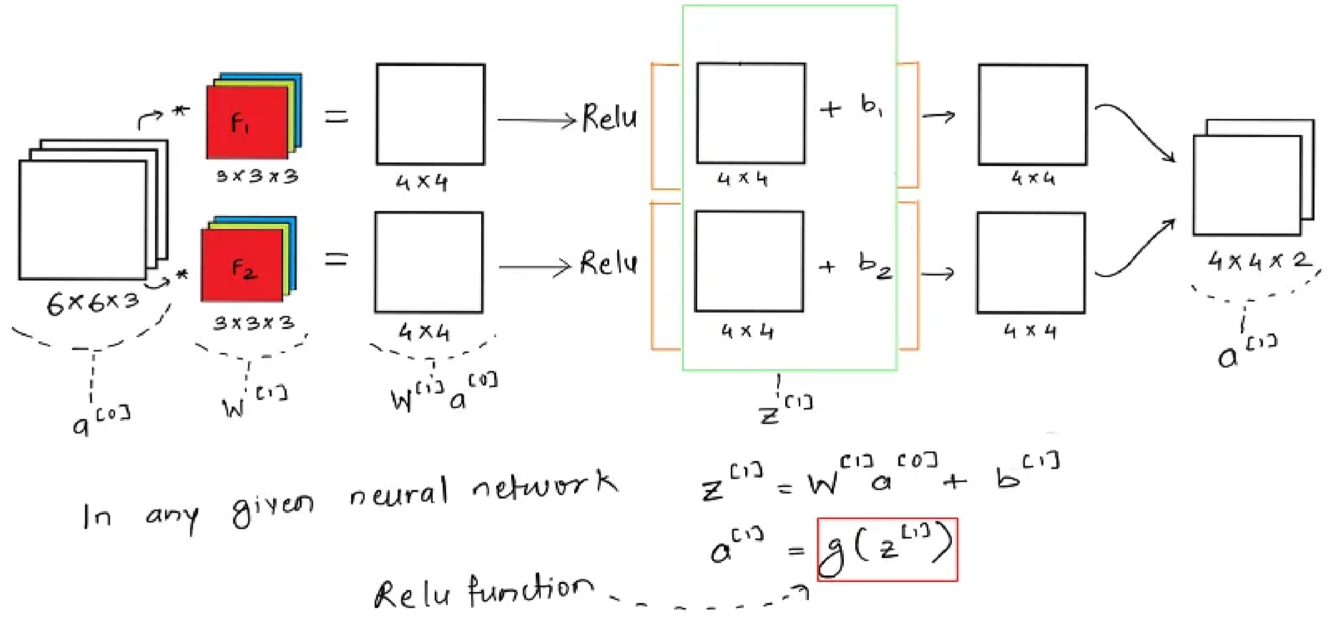
Multiple Filters at One Time



$$(n \times n \times n_c) * (f \times f \times n_c) = (n-f+1) \times (n-f+1) \times n_f$$

- Need to extract a lot of different features from an image
- If individual filters are convolved separately, increases computation time
- Its more convenient to use all required filters at a time directly.
- nc is the number of channels in the input image
- nf are the number of filters used.

One Layer of Convolution Network



$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[1]} = \boxed{g(z^{[1]})}$$

- consider one layer of a convolution network without the pooling layer and flattening, then it will appear close to this

Understanding the dimensionality Change

Layer *l* which is a convolution layer

filter size	$= f^{[l]}$	input	$= n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$
padding	$= p^{[l]}$	each filter	$= f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
stride	$= s^{[l]}$	weights	$= f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
		output	$= n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$
		bias	$= 1 \times 1 \times 1 \times n_c^{[l]}$

Final dimension n of layer l can be calculated using:

$$n^{[l]} = \left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Deep Learning Frameworks

1. Tensorflow
2. Keras
3. PyTorch
4. Theano
5. Deeplearning4j (DL4J)
6. MaxNet
7. Chainer
8. Caffe
9. CNTK
10. Torch

Deep Learning Frameworks

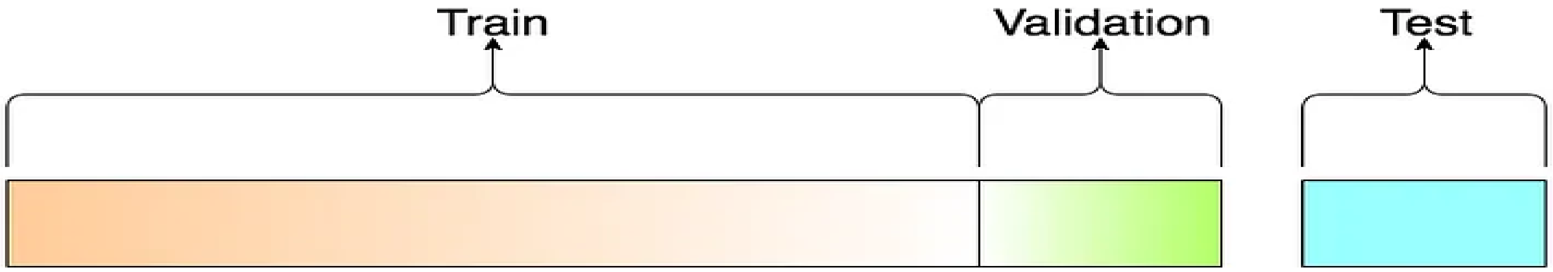
Content	Tensorflow	Keras	PyTorch	Theano	Deeplearning4j
Initial release:	November 9, 2015	March 27, 2015	September 2016	2007	2014
Stable release:	2.4.1 / January 21, 2021	2.4.0 / June 17, 2020	1.7.1 / December 10, 2020	1.0.5 / July 27, 2020	1.0.0-beta6 / September 10, 2019
Written in:	Python, C++, CUDA	Python	Python, C++, CUDA	Python	
Platform:	Linux, macOS, Windows, Android, JavaScript	Cross-platform	IA-32, x86-64	Linux, macOS, Windows	Cross-platform
Type:	Machine learning library	Neural networks	Library for machine learning and deep learning	Machine learning library	NLP, Deep Learning, Machine Vision, AI

Deep Learning Frameworks

Framework	Description	Main Features
TensorFlow	- Developed by Google Brain - One of the most popular deep learning frameworks - Provides a comprehensive ecosystem of tools, libraries, and community resources for developing machine learning models.	- Graph-based computation -Flexibility - Scalability - Production readiness - Support for distributed training - Serving for deploying models - TensorFlow Lite for mobile and edge devices - TensorFlow.js for web applications - TensorFlow Extended (TFX) for end-to-end ML pipelines.
PyTorch	- Developed by Facebook AI Research - known for its dynamic computation graph, making it more intuitive and easier to debug compared to TensorFlow. - Gained popularity for research and prototyping due to its simplicity and Pythonic nature.	- Dynamic computation graph - Ease of use - Flexibility - Native support for Python debugging tools - Strong community support - Seamless integration with NumPy - PyTorch Lightning for structured deep learning research - TorchServe for model deployment - TorchScript for model optimization - ONNX interoperability.

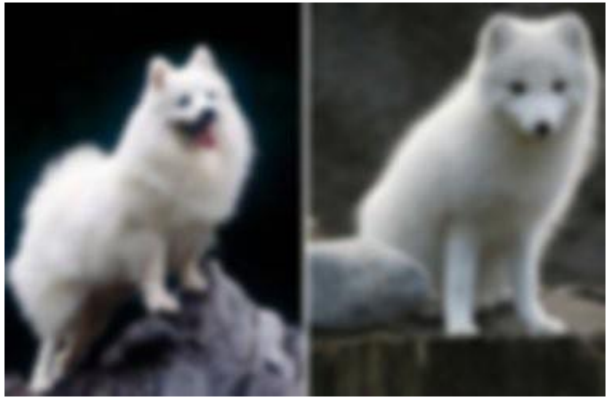
Training and Testing on Different Distrubutions

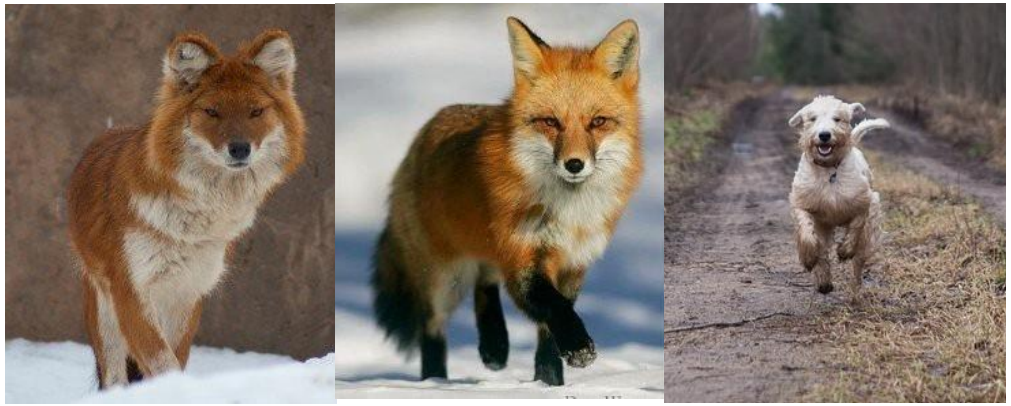
- To build a well-performing DL model
  - Train the model → data that come from
  - Test the model the same target distribution
- Fact :- Limited amount of data of target distribution
- Disadvantage : - may not be sufficient to build the needed train/dev/test sets.



Scenario- building a dog-image classifier

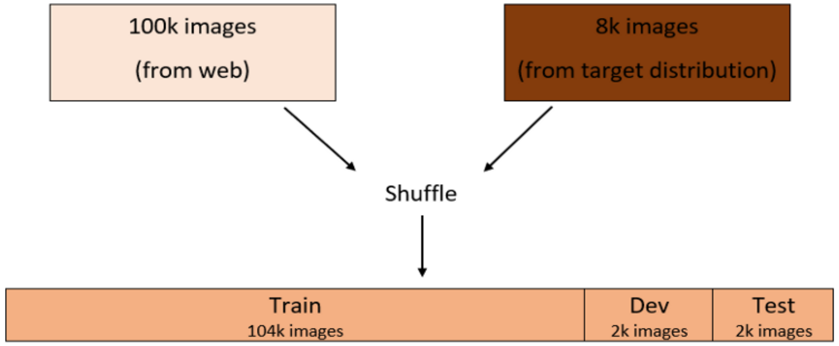
- Determines if n image is a dog or not
- Application for users in Rural areas
- Target data distrbution $\longrightarrow$ mostly blurry , low resolution images
- Captured Images = 8000 , not sufficient to build train/dev/test sets
- Solution :- crape the web to build a dataset of 100,000 images or more
- Problem :- comes from different ditribution having High resolution and clear images





Scenario- building a dog-image classifier

- How would you build the train/dev/test sets?
  - ➔ can't only use the original 8,000 images
  - ➔ need a lot of data
  - ➔ can't only use the web dataset
- **Possible Option** -
  - ➔ 96:2:2% split

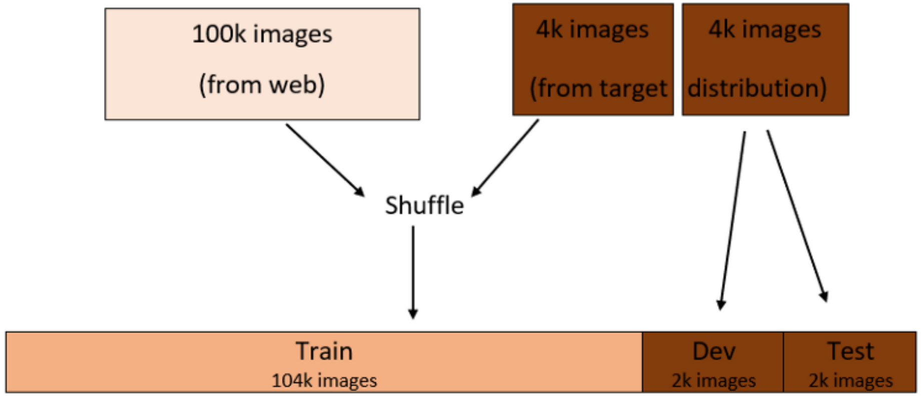


- Drawback:-
  - ➔ At the dev set, out of 2,000 images, on average only 148 images come from the target distribution.
  - ➔ Most part you are optimizing the classifier for the web images distribution (1,852 images out of 2,000 images) —
    - ➔ **which is not what you want!**

➔

Scenario- building a dog-image classifier

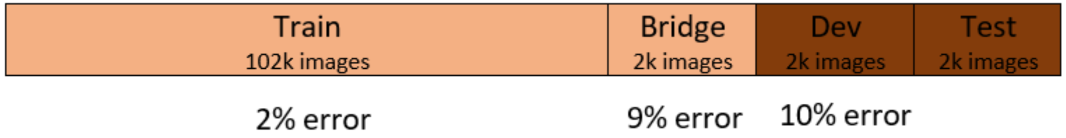
- **Possible Option** -
  - 96:2:2% split
  - dev/test sets will be 2,000 images each — coming from the target distribution
  - rest will go to the train set
  - optimizing the classifier to perform well on the target distribution



- Drawback:-
  - For the most part, you are training the classifier on web images
  - It will take longer and more effort to optimize the more

Scenario- building a dog-image classifier

- **Variance_error** -
  - ➔ Assume you found that the training error to be 2% and the dev error 10%
  - ➔ Weather overfitting or variance error we can't say
  - ➔ Take out a small portion of the train set and call it the "bridge" set. (Independent set- not used to train classifier)



  - ➔ 8% error between the train and dev set errors
  - ➔ 7% variance error and 1% data mismatch error
  - ➔ Reason : -the bridge set comes from the same distribution as the train set, and the error difference between them is 7%
  - ➔ The classifier is overfitted to the train set - **high variance problem**

Scenario- building a dog-image classifier

- **Data_mismatch_error** -
  - Assume you found that the training error to be 2% and the dev error 10%
  - Weather overfitting or variance error we can't say
  - Take out a small portion of the train set and call it the "bridge" set. (Independent set- not used to train classifier)

Train 102k images	Bridge 2k images	Dev 2k images	Test 2k images
2% error	3% error	10% error	

  - 8% error between the train and dev set errors
  - 1% variance error and 7% data mismatch error
  - Reason : it is because the classifier performs well on a dataset it hasn't seen before if it comes from the same distribution, such as the bridge set
  - It performs poorly if it comes from a different distribution, like the dev set – **data mismatch problem**

.

➔

Scenario- building a dog-image classifier

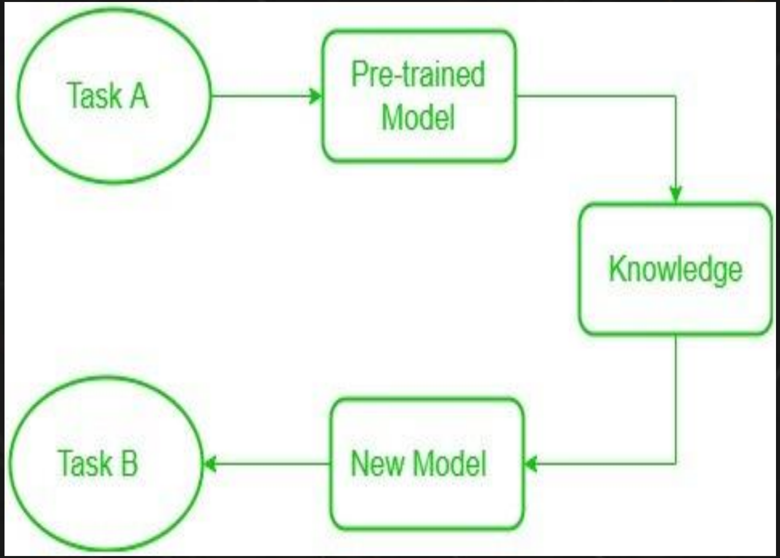
- **Mitigating data mismatch**
  - ➔ somehow incorporate the characteristics of the dev/test datasets — the target distribution — into the train set to reduce mismatch error
  - ➔ Best option - Collecting more data from the target distribution to add to the train set
  - ➔ If not possible, following approaches are used-

    1) **Error analysis** – Analyse errors on dev set and how they are different from train set
    2) **Artificial Data Synthesis** - synthesize data with similar characteristics

  - ➔

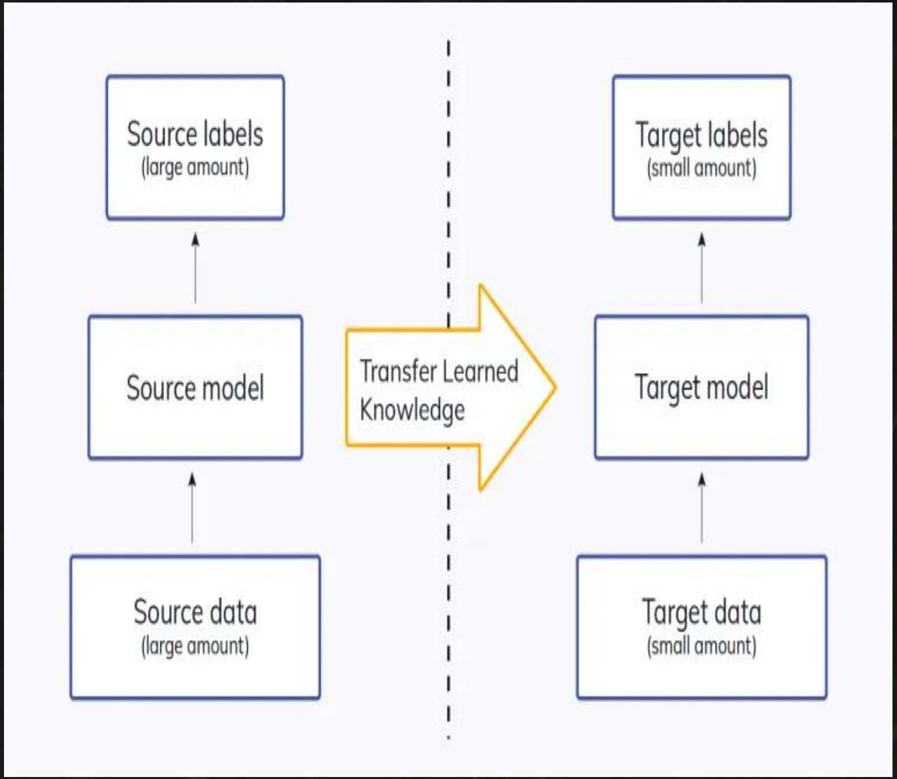
Transfer Learning

- Definition
  - Method that reuses a trained model designed for a particular task to accomplish a different but related task
  - knowledge acquired from task one is thereby transferred to the second model that focuses on the new task
- Captures the lessons learned in one task and applies them to fine-tune another task
- Idea-
  - Reprocess the information gained from task_1, which has labeled training data
  - Complete task_2, which has less data or labels than task_1
- Learning process can begin from patterns captured while addressing similar tasks rather than beginning from ground zero
- Used Mainly in:-
  - Computer Vision
  - NLP



Significance of Transfer Learning

- Speeds up the overall process of training a new model
- Improves its performance
- Primarily used when a model requires large amount of resources and time for training
- When the available training data is insufficient, transfer learning plays a vital role
- It uses the weights captured from the first model to initialize the weights of the second model.
- Can yield optimized results when the dataset used in the second training is similar to the one used in the first training


Source labels (large amount)
Target labels (small amount)
Source model
Transfer Learned Knowledge
Target model
Source data (large amount)
Target data (small amount)

Transfer Learning Methods

1) **Train ’similar domain’ models**

➔ need to complete task X but lack sufficient data

➔ task Y is similar to task X and has enough data to complete task Y

➔ train a model on task Y and then use the successful model to develop a new model to work on task X

2) **Extract Features**

➔ automatic feature extractors

3) **Use pre-trained models**

➔ re-used to train another model

➔ Popular Models - AlexNet, Oxford’s VGG Model, and Microsoft’s ResNet

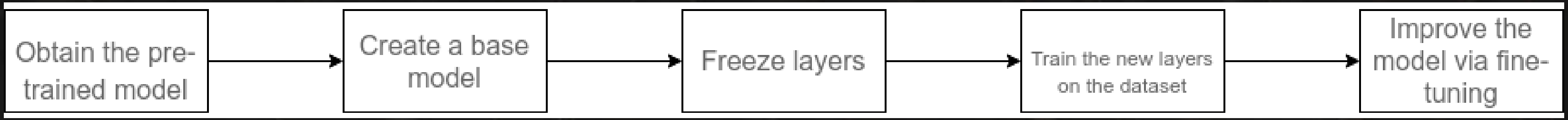
Transfer Learning Methods

1) **Train 'similar domain' models**

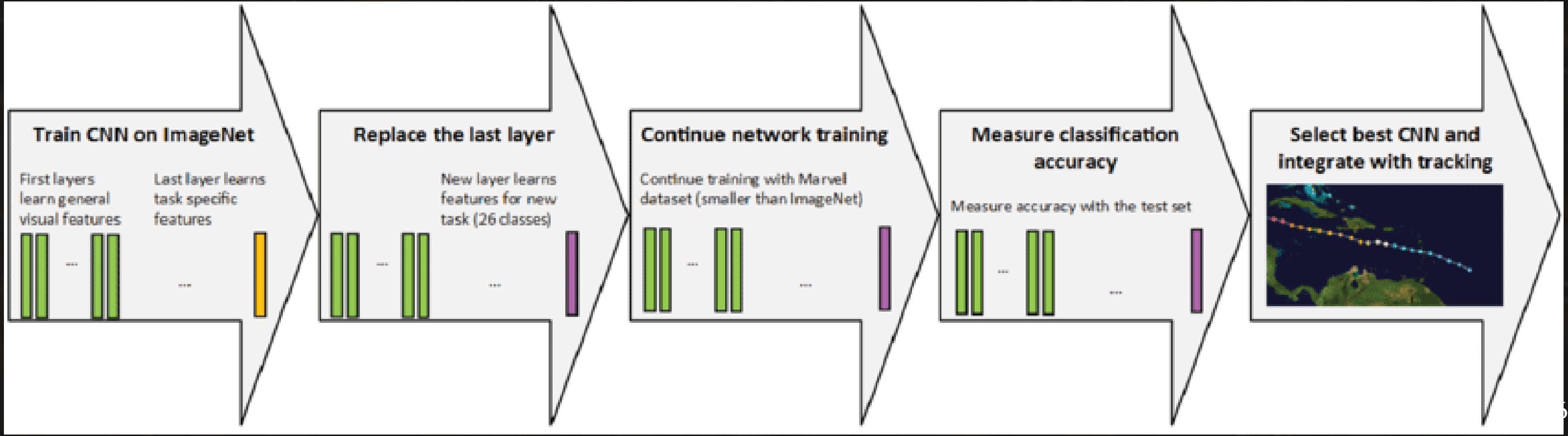
➔ need to complete task X but lack sufficient data

➔ task Y is similar to task X and has enough data to complete task Y

2)



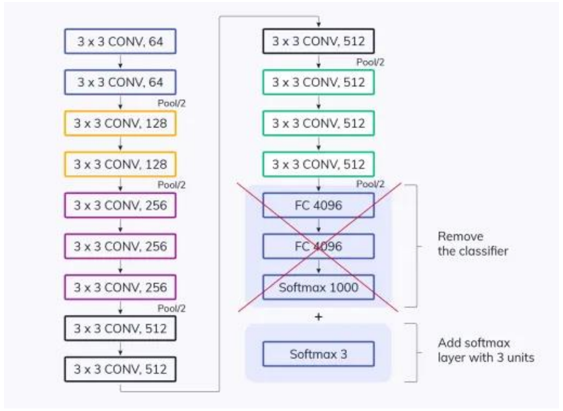
Obtain the pre-trained model
Create a base model
Freeze layers
Train the new layers on the dataset
Improve the model via fine-tuning

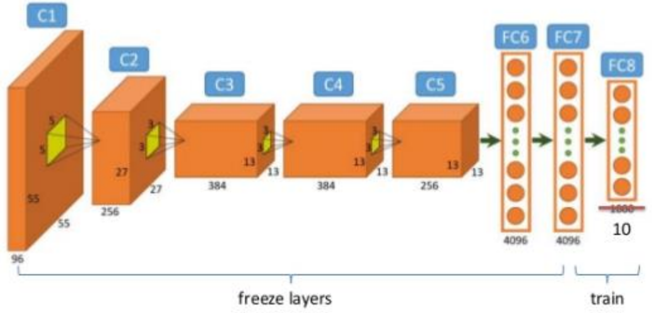


Train CNN on ImageNet
First layers learn general visual features
Last layer learns task specific features
Replace the last layer
New layer learns features for new task (26 classes)
Continue network training
Continue training with Marvel dataset (smaller than ImageNet)
Measure classification accuracy
Measure accuracy with the test set
Select best CNN and integrate with tracking

Transfer Learning Process

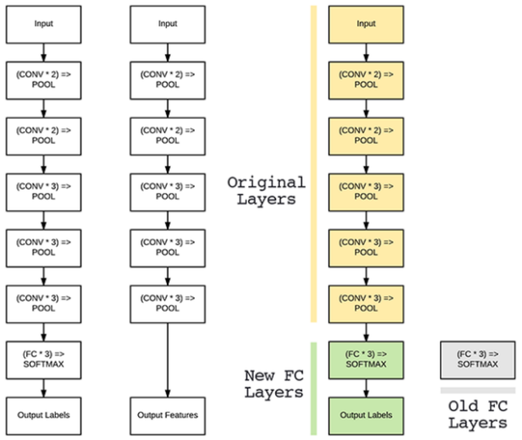
- **Access pre-trained models**
  - Can obtain pre-trained models from
    - Their own collection of model libraries
    - Other open-source repositories
  - Example - PyTorch Hub - open-source pre-trained model repository
    - Designed to speed up the research path, from prototyping to product deployment
  - TensorFlow Hub - open repository and reusable ML library
    - Can be used for tasks such as text embeddings, image classification
  - Download pre-trained weights
  - Base model will usually have more units in the final output layer than you require
  - When creating the base model, therefore, have to remove the final output layer.
  - Later on, add a final output layer that is compatible with your problem
- **Freeze Layers**
  - Typical neural network three layers: inner (early), middle, and latter layers
  - Retained inner and middle layers as they are in transfer learning
  - Only latter layers are retrained – so method can use labeled data o the task it was previously trained on

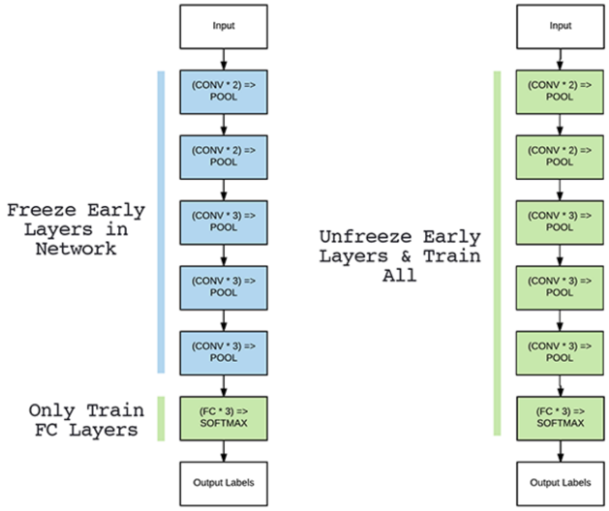




Transfer Learning Process

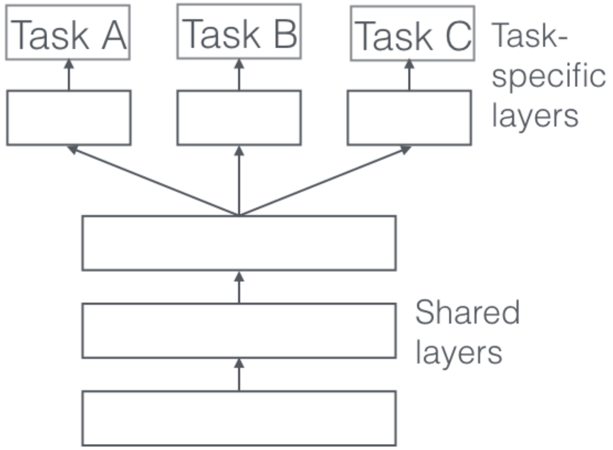
- **Add New Trainable Layers**
  - turn old features into predictions on the new dataset
  - Important because the pre-trained model is loaded without the final output layer.
- **Train the new layers on the dataset**
  - pre-trained model's final output will most likely be different from the output that you want for your model
  - So, add some new dense layers as you please
  - A final dense layer with units corresponding to the number of outputs expected by your model.
- **Improve the model via fine-tuning**
  - improve its performance through fine-tuning
  - Fine-tuning is done by unfreezing the base model or part of it and training the entire model again on the whole dataset at a very low learning rate
  - The low learning rate will increase the performance of the model on the new dataset while preventing overfitting.
  - The learning rate has to be low because the model is quite large while the dataset is small
  - Compile function again whenever you want to change the model's behavior
  -

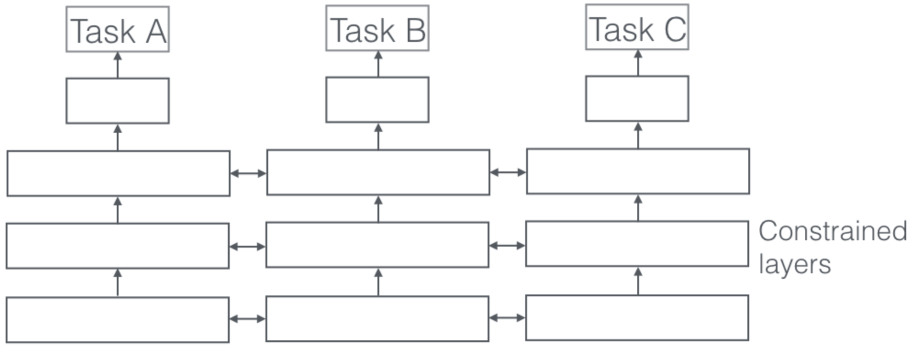




Multi-task Learning

- Model training technique where you train a single deep neural network on multiple tasks at the same time
- Applications in
  - Machine Learning
  - NLP
  - Speech Precessing
  - Computer Vision
- **Hard parameter sharing**
- **Soft parameter sharing**

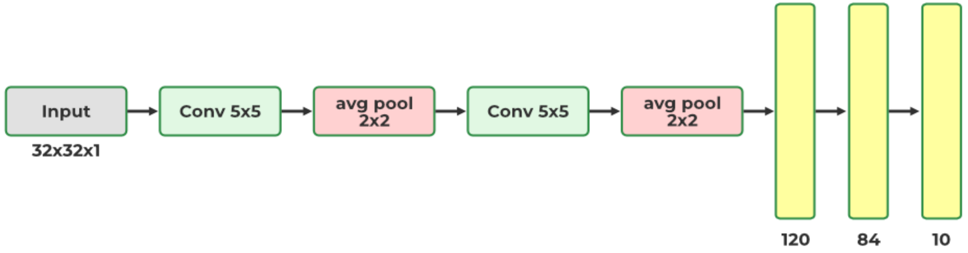




CNN Architectures – LeNet-5

- Most widely known CNN architecture.
- Introduced in 1998 and is widely used for handwritten method digit recognition. (Yann LeCun)
- Has 2 convolutional and 3 full layers
- Has 60,000 parameters.
- Has the ability to process higher one-resolution images that require larger and more CNN convolutional layers.
- Measured by the availability of all

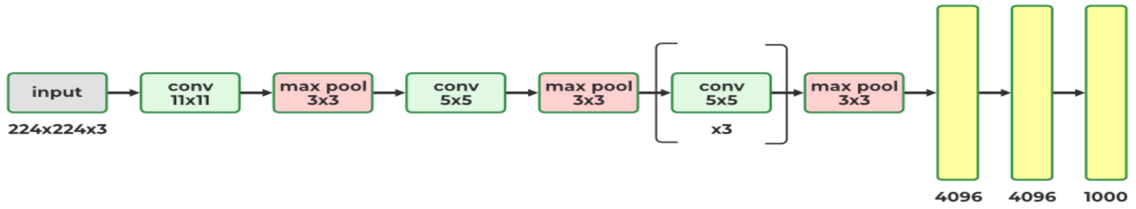
➢



CNN Architectures – AlexNNet

- Introduced by Alex Krizhevsky (name of founder)
- Quite similar to LeNet-5, only much bigger and deeper
- It was introduced first to stack convolutional layers directly on top of each other models, instead of stacking a pooling layer top of each on CN network convolutional layer.
- Has 60 million parameters as AlexNet has total 8 layers, 5 convolutional and 3 fully connected layers.
- First to execute (ReLUs) Rectified Linear Units as activation functions
- First CNN architecture that uses GPU to improve the performance.

➢



CNN Architectures – Comparison

Architecture	Year	Layers	Notable Features	Performance
LeNet-5	1998	7	Early CNN, simple structure	Handwritten digit recognition
AlexNet	2012	8	ReLU activation, dropout, data augmentation	Winner of ILSVRC 2012
VGGNet	2014	16/19	Uniform architecture, small 3x3 filters	Strong performance on ImageNet
GoogLeNet	2014	Varies	Inception modules, computational efficiency	State-of-the-art on ImageNet
ResNet	2015	Varies	Skip connections (residual connections)	Winner of ILSVRC 2015
DenseNet	2016	Varies	Dense connectivity, feature reuse	Effective with limited data