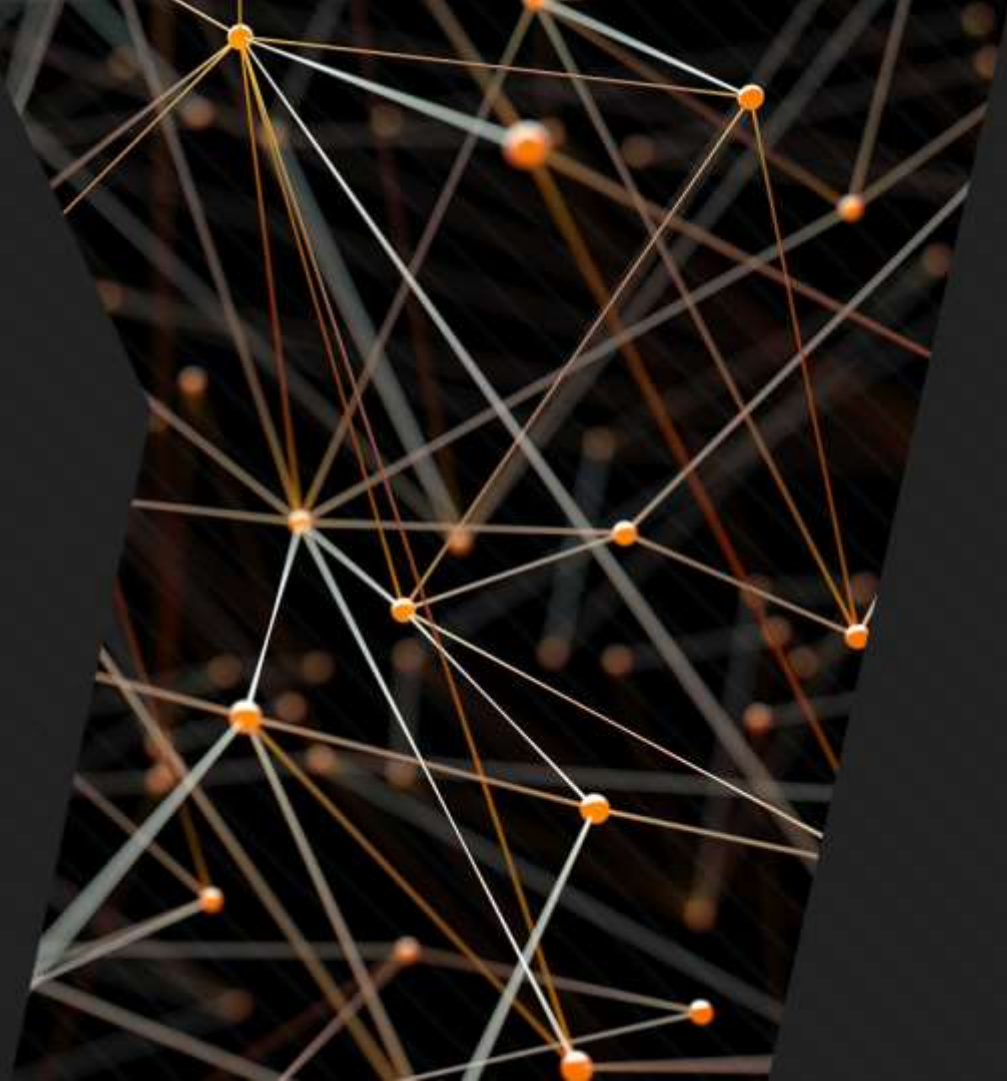
Convolutional Neural Network (CNNs)

Unit-III



Introduction to CNN

- Extension of traditional Perceptron, Special kind of multi-layer Neural Network
- Also known as **“Convets”**
- Type of Deep Learning neural network architecture commonly used in Computer Vision
  - Computer vision --> field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.
- one of the most deployed deep learning neural networks.
- It is Feed-Forward Network that can extract topological properties from an image
- Designed to recognize visual pattrens directly from pixel images with minimal preprocessing
- Can recognize patterns with extreme variability (such as handwritten characters)

Background and Common uses for CNNs

- Developed and deployed for the first time, Around the 1980s
- Could only detect handwritten digits at the time
- Primarily used in various areas to read zip and pin codes etc.
- common aspect of any A.I. model --> It requires a massive amount of data to train
  - biggest problems that CNN faced at the time, thus only used in postal industry
- Yann LeCun - first to introduce convolutional neural networks.
- Kunihiko Fukushima, a renowned Japanese scientist, who even invented recognition
  - very simple Neural Network used for image identification
- Use of CNN-
  - Image classification
  - Image Segmentation
  - NLP , Speech Processing

Classical Computer Vision Pipeline.

CV experts

1. Select / develop features: SURF, HoG, SIFT, RIFT, …
2. Add on top of this Machine Learning for multi-class recognition and train classifier

Classical CV feature definition is domain-specific and time-consuming

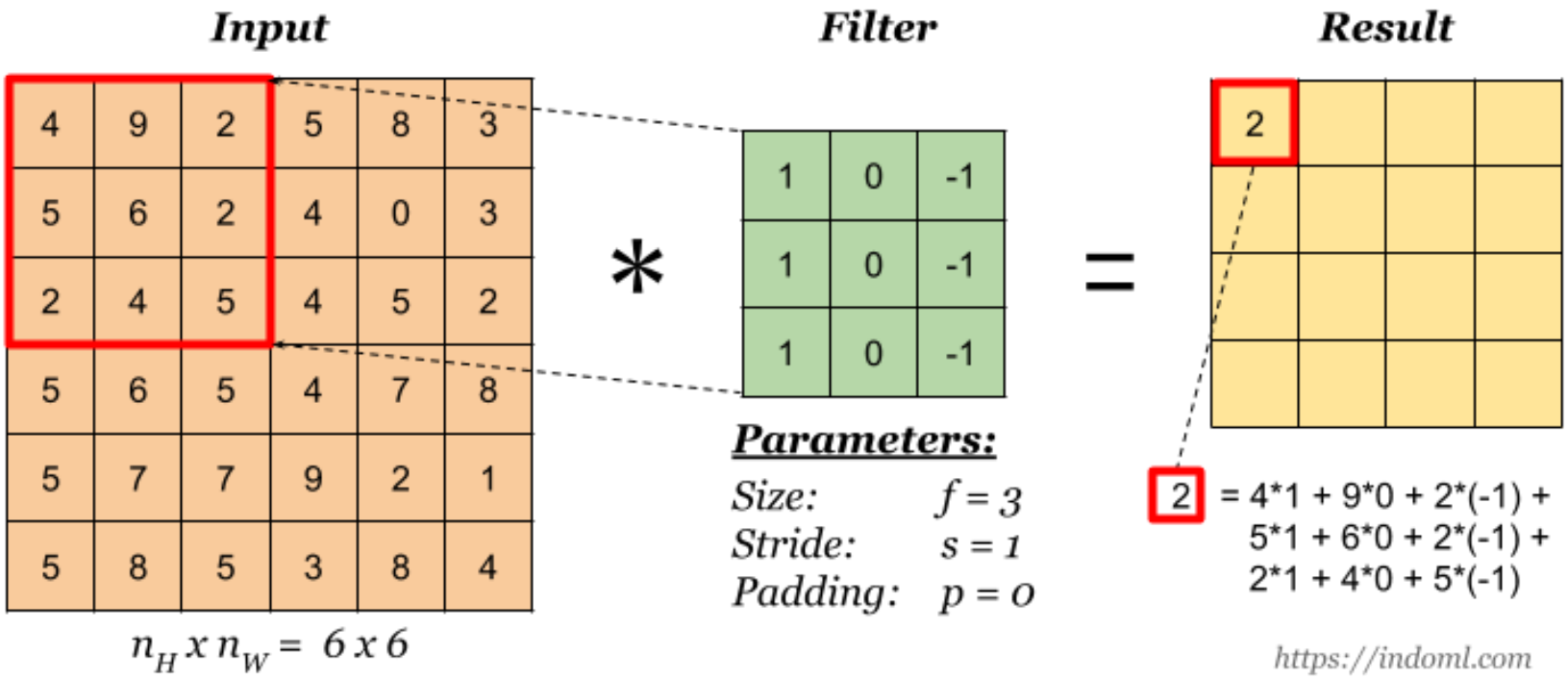
Disadvantage of classical Method

- Time consuming As Feature selection is manual
- Feature extraction is domain specific
- Overcome using Deep learning methods

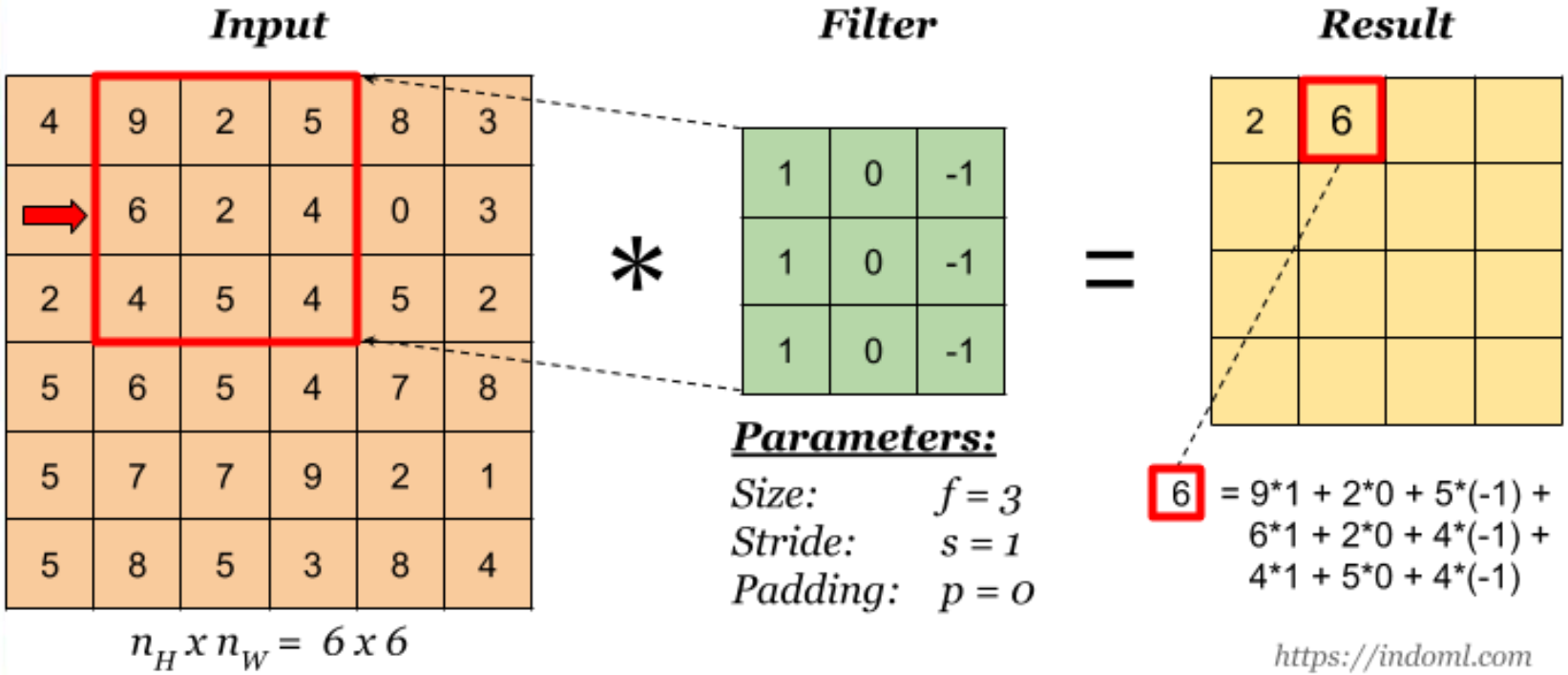
What is Convolution?

- Mathematical operation that combines two sets of information to produce a third set
- representing how one set modifies the other
- In image processing,
  - Convolution is the process of transforming an image
  - By applying a kernel over each pixel and its local neighbors across the entire image.
- **The kernel**
  - Matrix of values whose size and values determine the transformation effect of the convolution process.
  - Very small matrix and in this matrix, each cell has a number and also an anchor point.
  - Anchor point is used to know the position of the kernel with respect to the image
  - Starts at the top left corner of the image and moves on each pixel sequentially

Basic Convolution Operation – Step 1

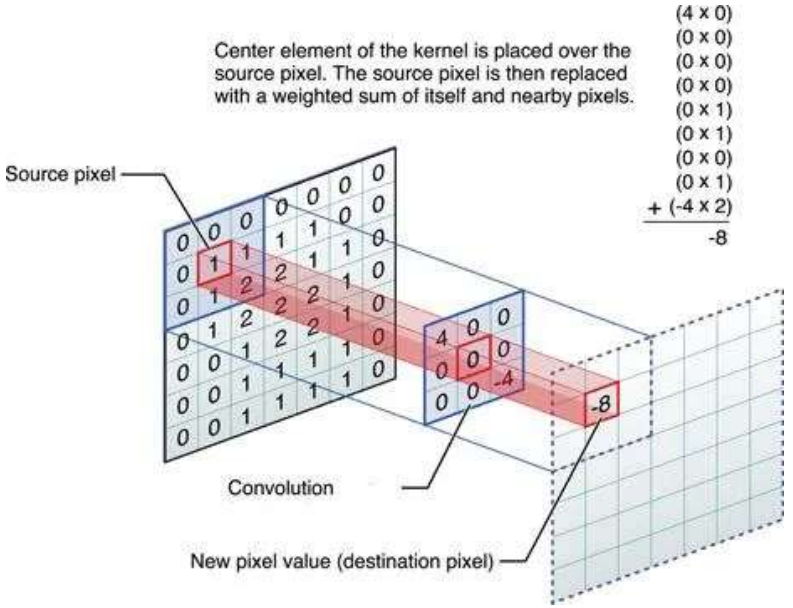


Basic Convolution Operation – Step 2



Convolution Operation

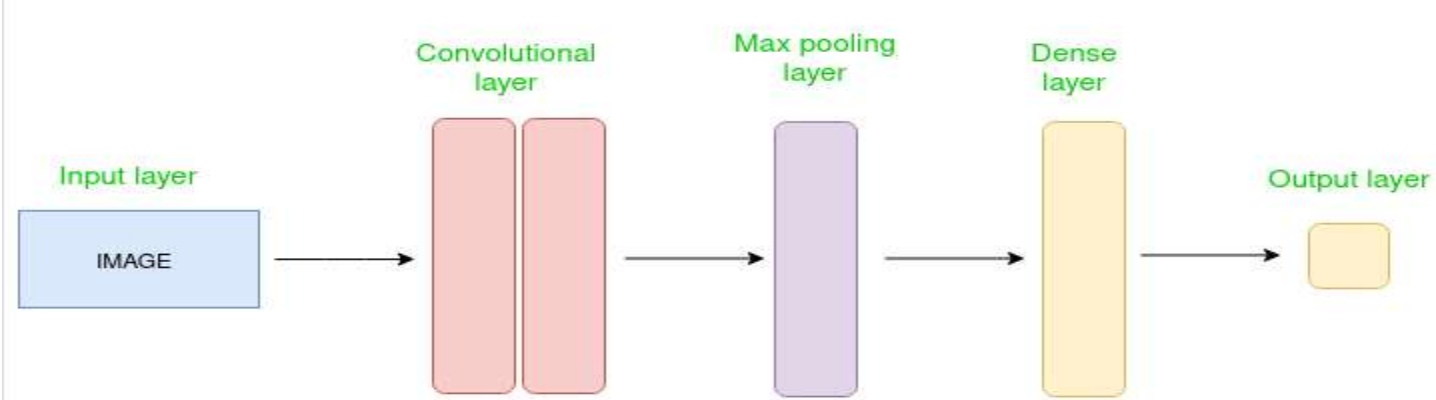
- Kernel overlaps few pixels at each position on the image
- Each pixel which is overlapped is multiplied and then added.
- And the sum is set as the value of the current position.
- This process is repeated across the entire image.
- Process in which each element of the image is added to its local neighbors,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



CNN comprises -:

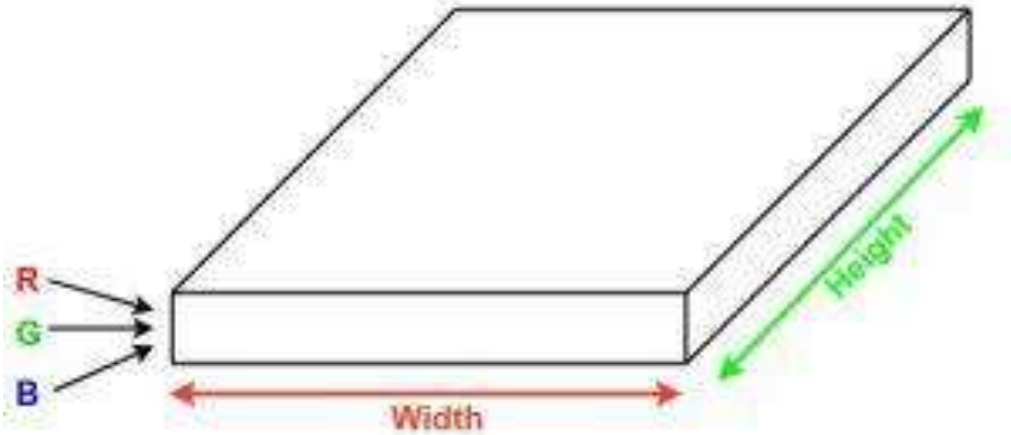
- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer.

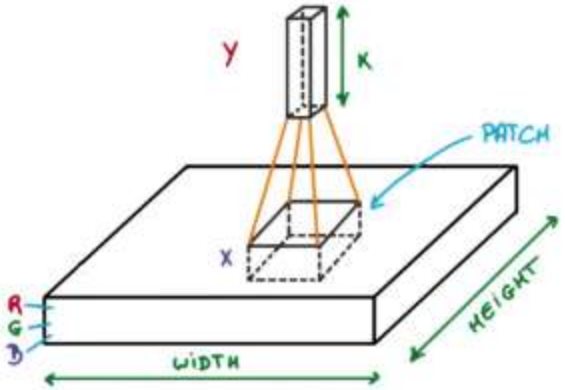
Simple CNN Architecture



- 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

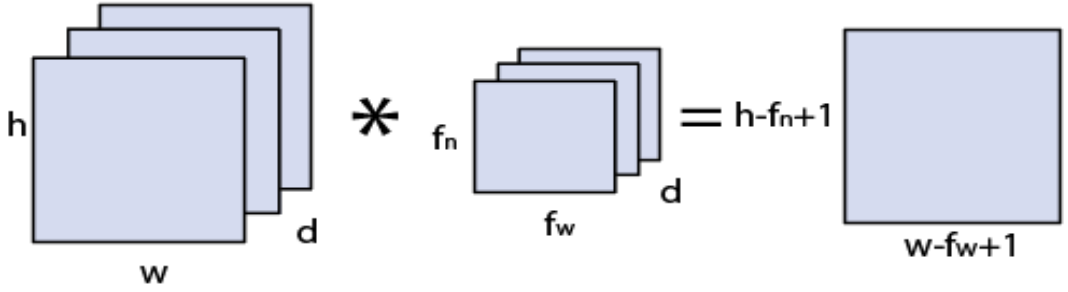


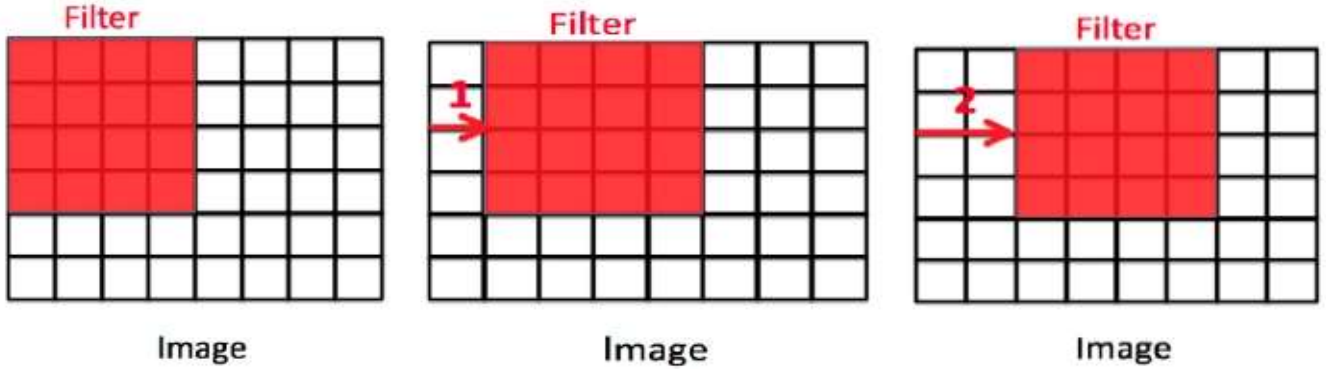
Image matrix multiplies kernl or filter matrix

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

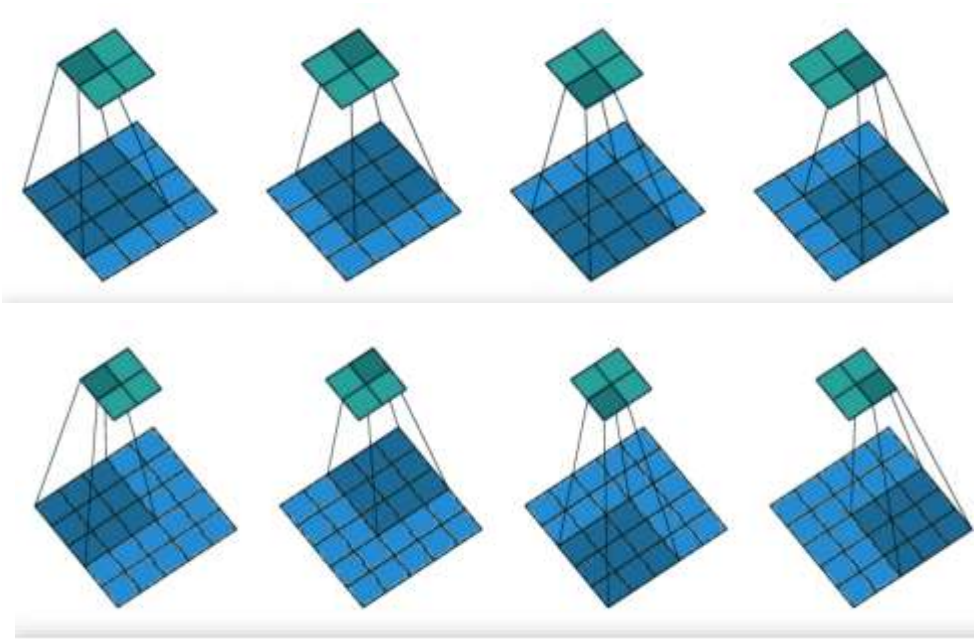


- When the array created, the pixels shifted over to the input matrix
- Definition- The number of pixels turning to the input matrix
- When the number of strides 1, we move the filters to 1 pixel at a time an so on
- Essential because they control the convolution of the filter against the input
- Strides 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:

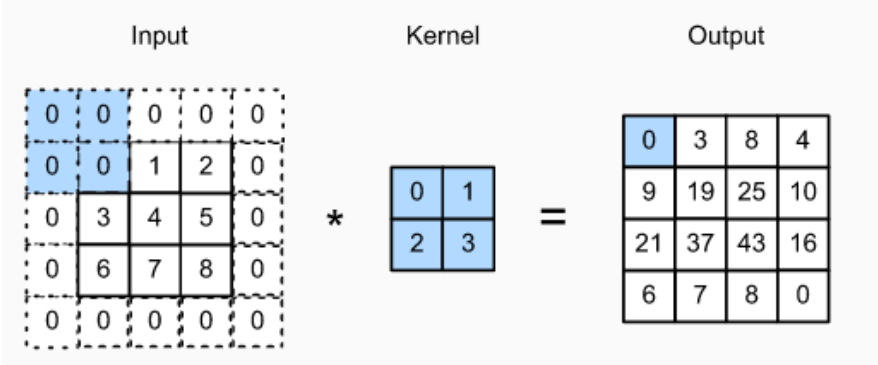
$$[\{(n+2p-f+1)/s\}+1][\{(n+2p-f+1)/s\}]$$

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)

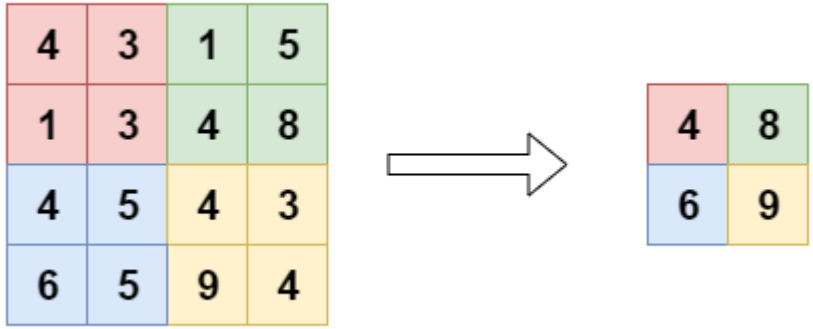
Pooling

- 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

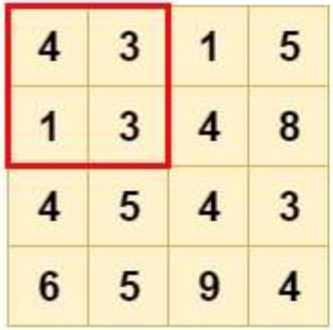
- Rule to take the maximum of a region
- Transfers continuous functions into discrete counterparts
- primary objective
  - To downscale an input by reducing its dimensionality
  - making assumptions about features contained in the sub-region that were rejected.

Max([4, 3, 1, 3]) = 4



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.



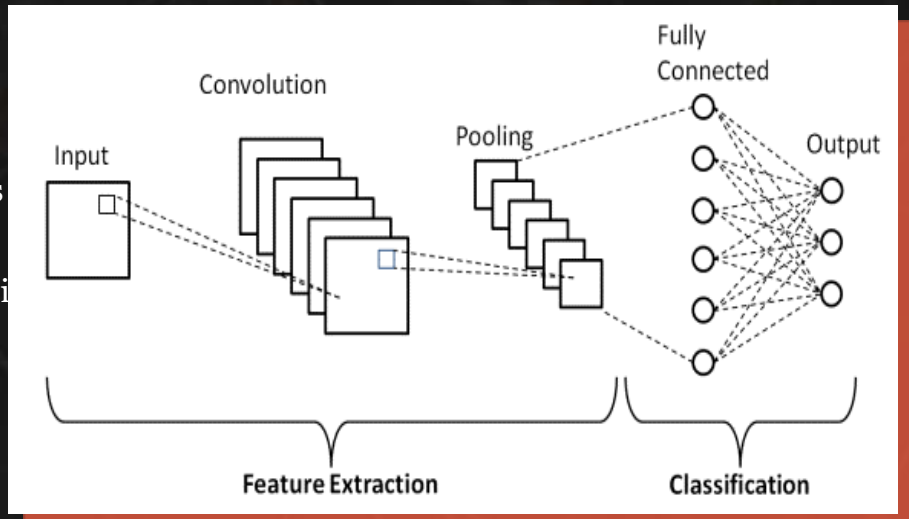
Avg([4, 3, 1, 3]) = 2.75

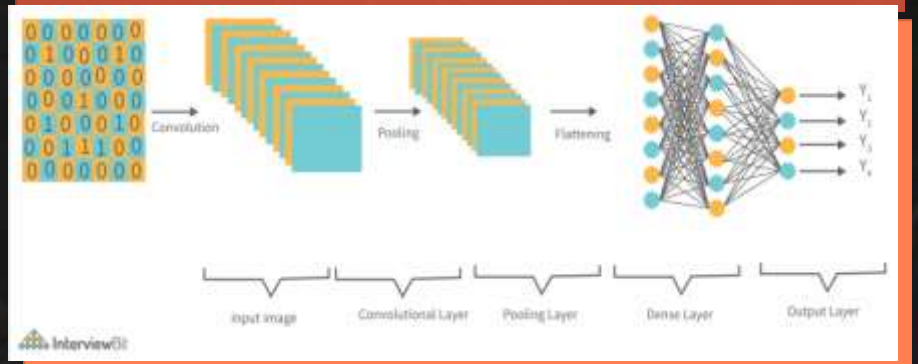
Pooling Layers

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





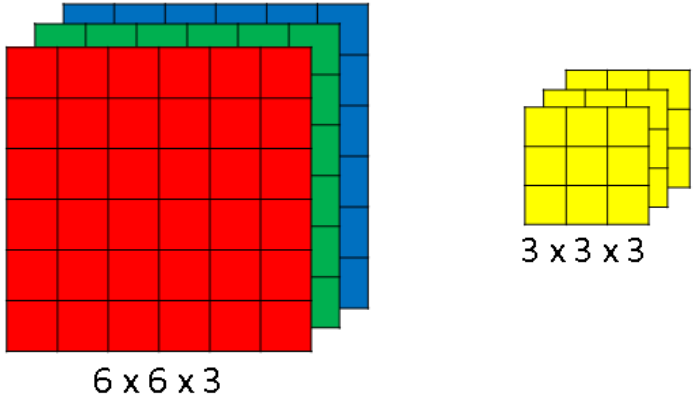
Convolution Layers

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

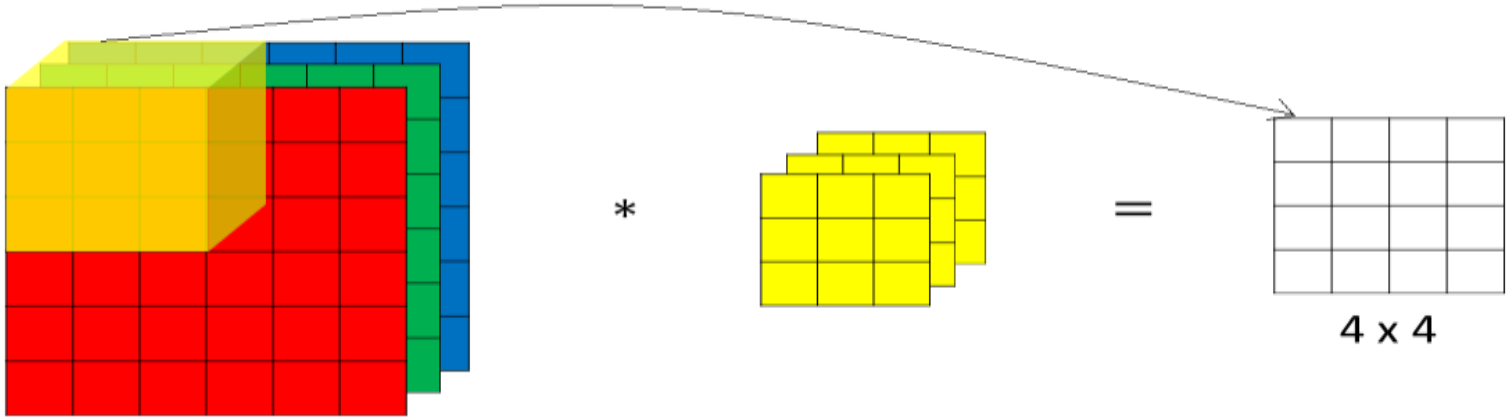
- Computationally efficient.
- Performs parameter sharing and uses special convolution and pooling algorithms.
- CNN models may now run on any device, making them globally appealing.
- Finds the relevant features without the need for human intervention.
- Can be utilized in a variety of industries to 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

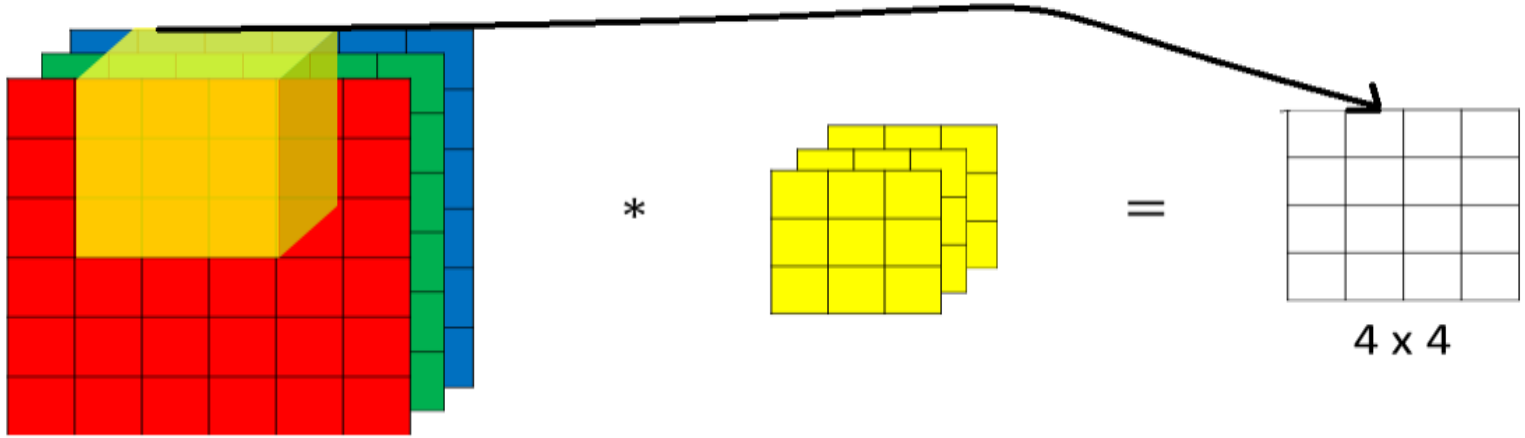


- Convolution on RGB Image
- Input data is no more 2D but in fact 3D
- Image dimension n x n x # channels
- Filter dimension f x f x # channels

Convolutions on RGB image

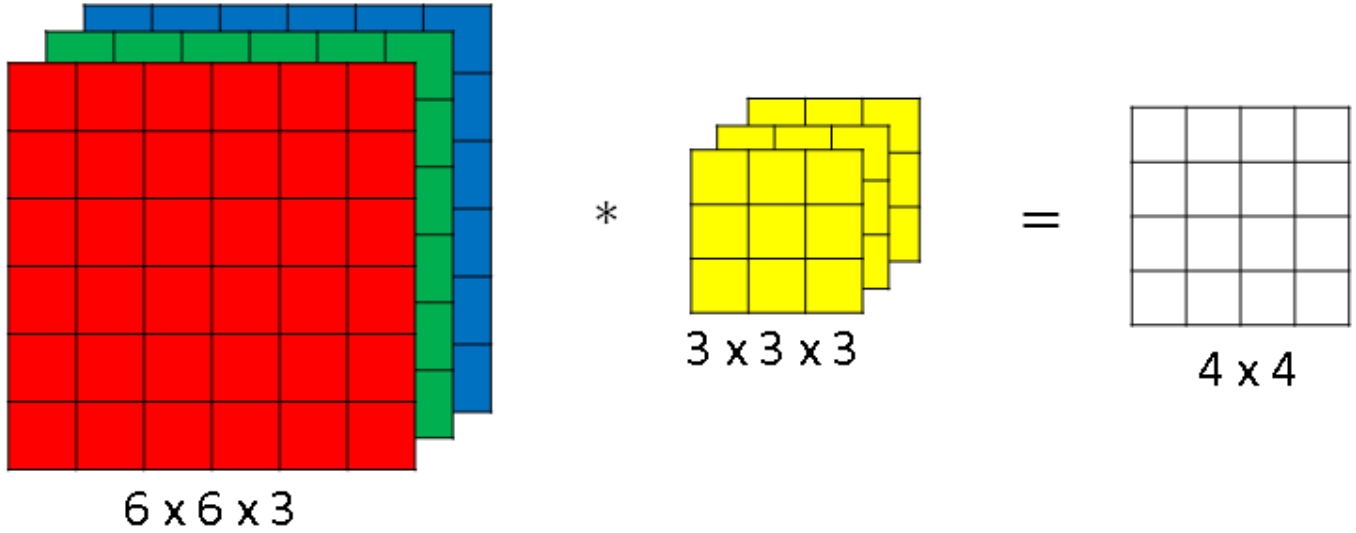


Convolutions on RGB image

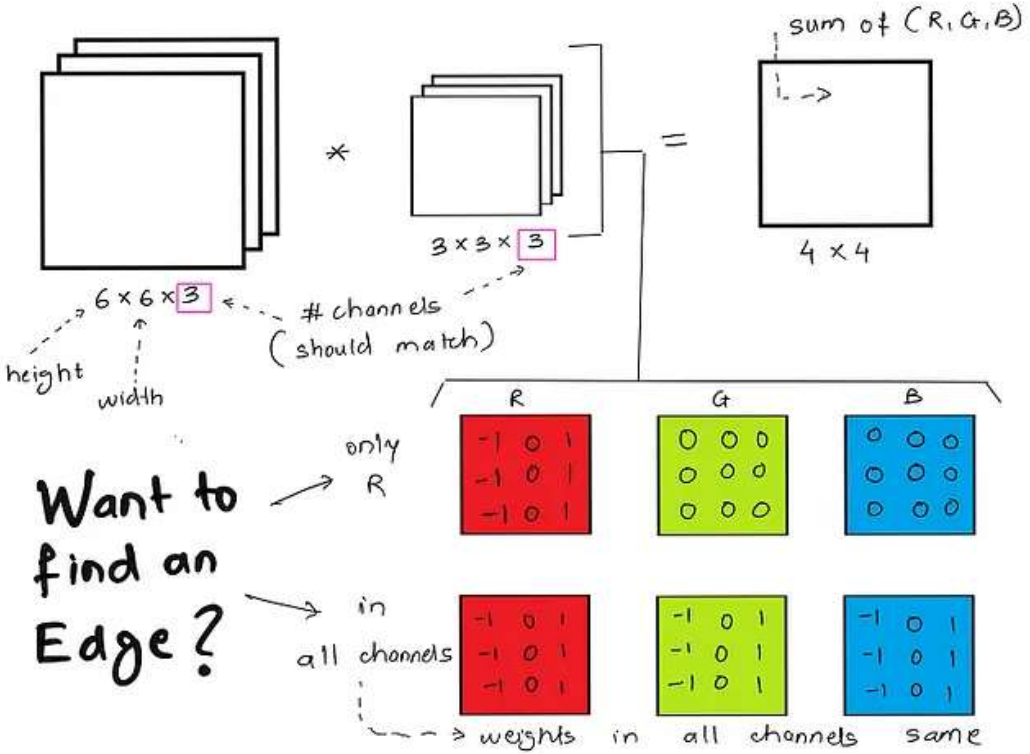


Convolution Over Volume

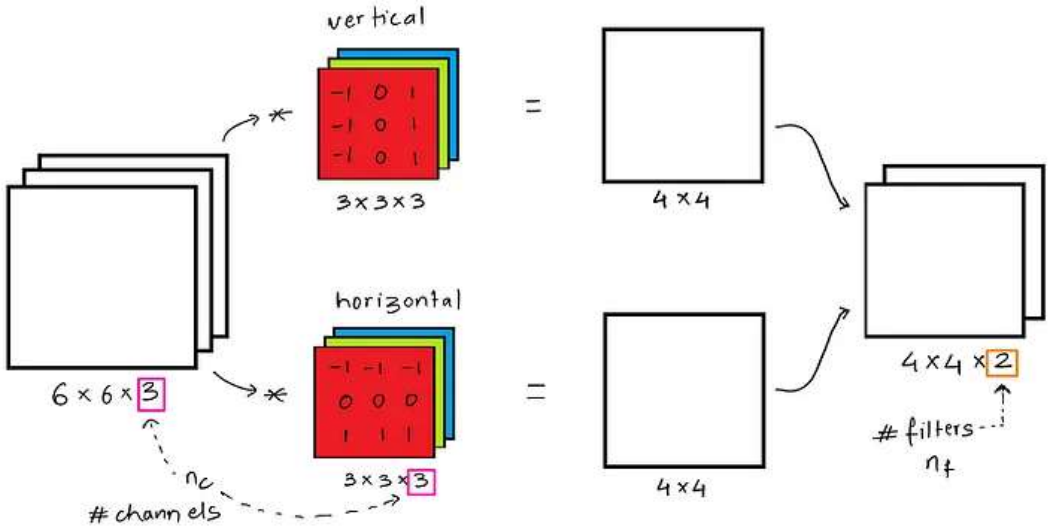
Multiple filters



Convolution Over Volume



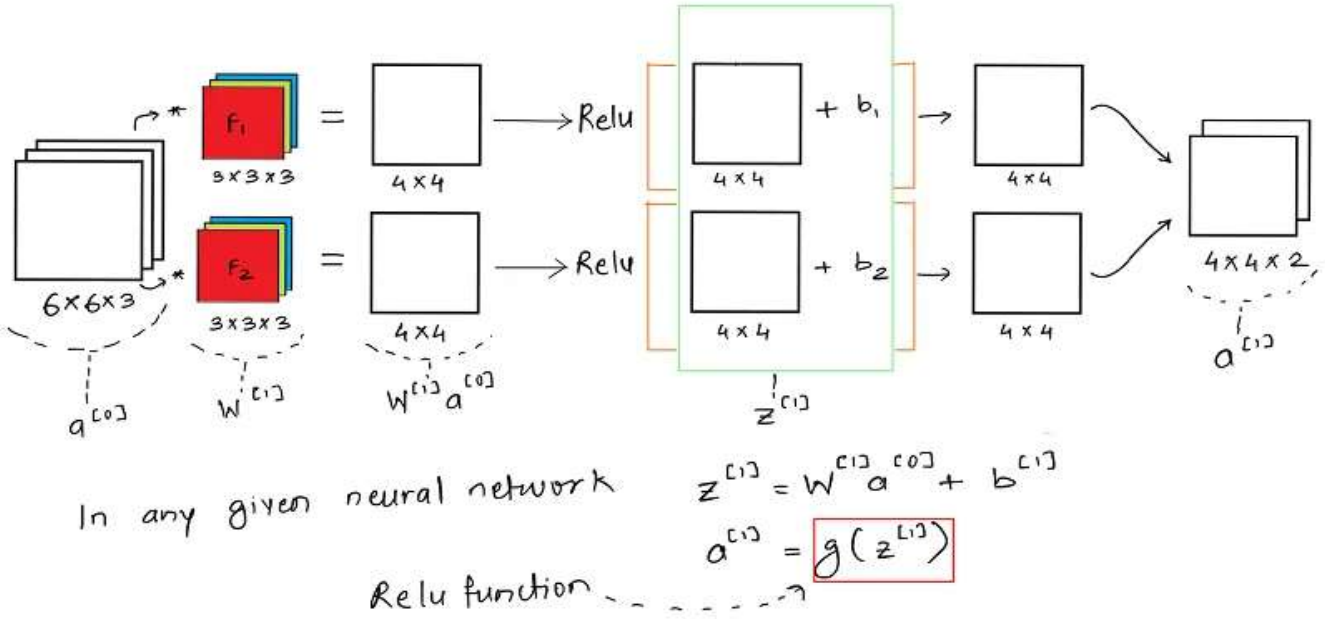
Multiple Filters at One Time



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

- 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



- 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 $= f^{[l]}$

padding $= p^{[l]}$

stride $= s^{[l]}$

input $= n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

each filter $= f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

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

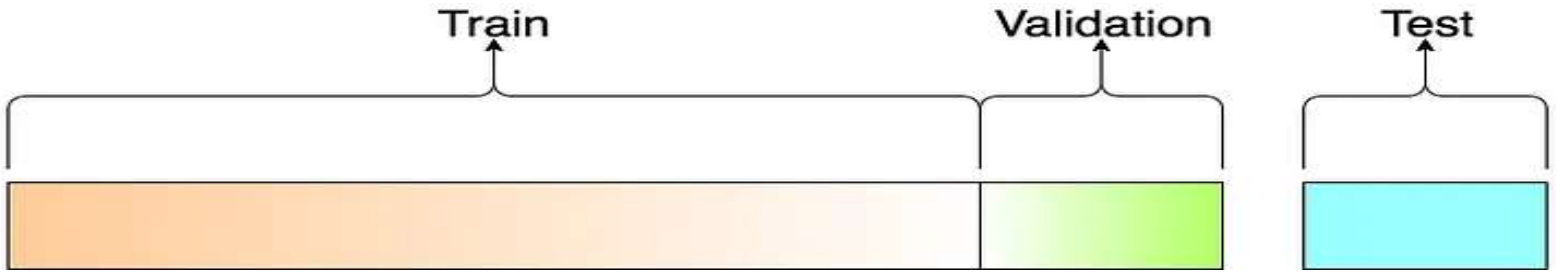
Content	Tensorflow	Keras	PyTorch	Theano	Deeplearning4j
Initial release:	November 9, 2015	March 27, 2015	September 2016	2007	2014
Stable release:	2.4.1 / January 21, 2021	2.4.0 / June 17, 2020	1.7.1 / December 10, 2020	1.0.5 / July 27, 2020	1.0.0-beta6 / September 10, 2019
Written in:	Python, C++, CUDA	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.

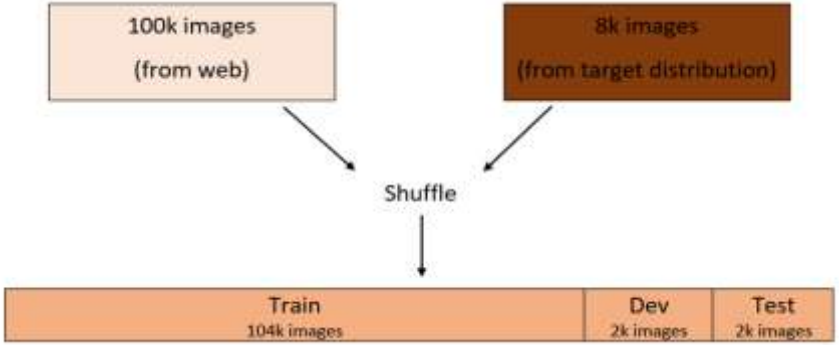


Scenario- building a dog-image classifier

- Determines if n image is a dog or not
- Application for users in Rural areas
- Target data distrbution → 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

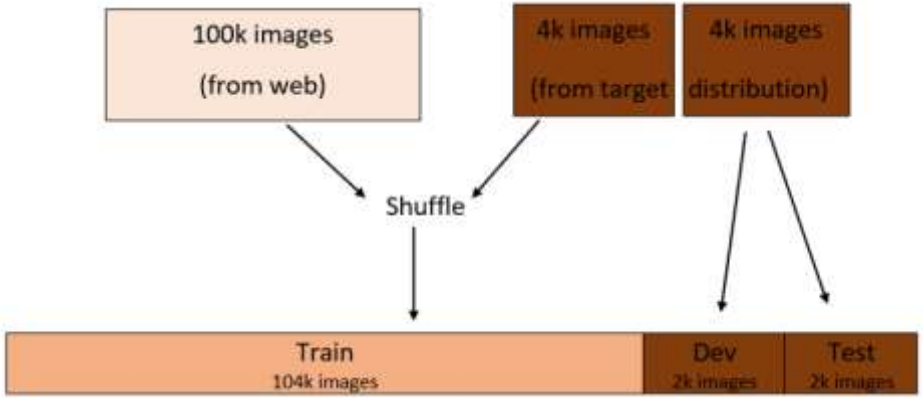
- How would you build the train/dev/test sets?
  - can't only use the original 8,000 images
  - need a lot of data
  - can't only use the web dataset
- **Possible Option** -
  - 96:2:2% split



- 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)

  - 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

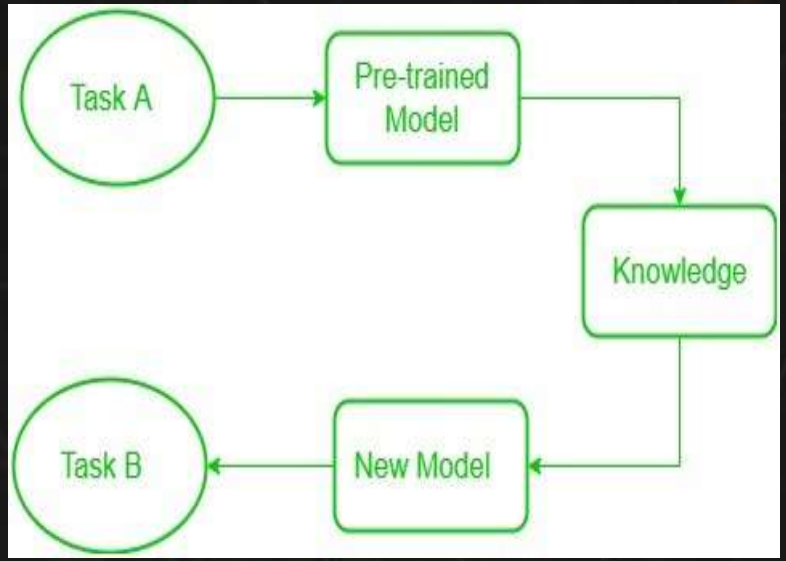
- **Mitigating data mismatch**
  - somehow incorporate the characteristics of the dev/test datasets — the target distribution — into the train set to reduce mismatch error
  - Best option - Collecting more data from the target distribution to add to the train set
  - If not possible, following approaches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

  -

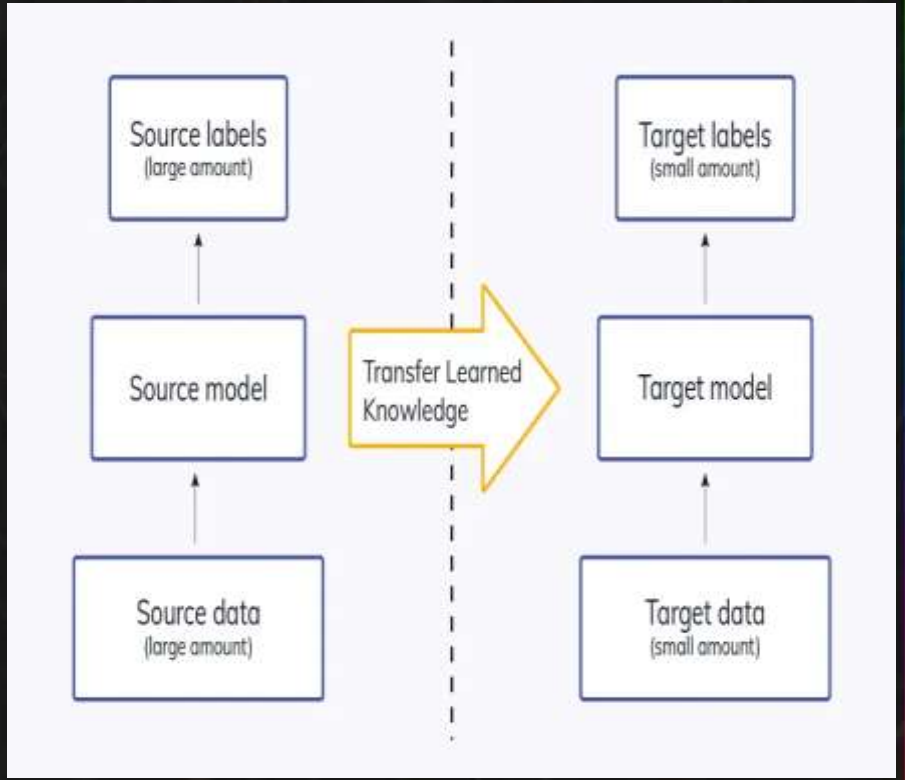
Transfer Learning

- 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



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

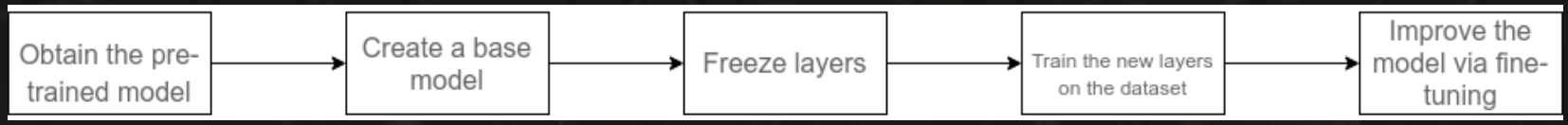
Transfer Learning Methods

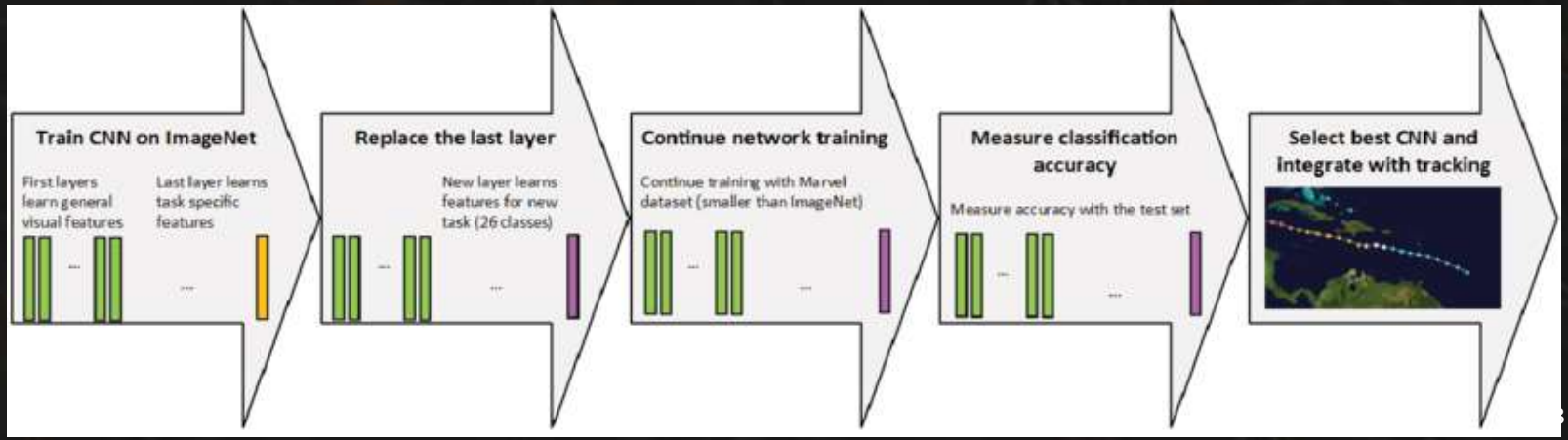
1) **Train 'similar domain' models**

   ➔ need to complete task X but lack sufficient data

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

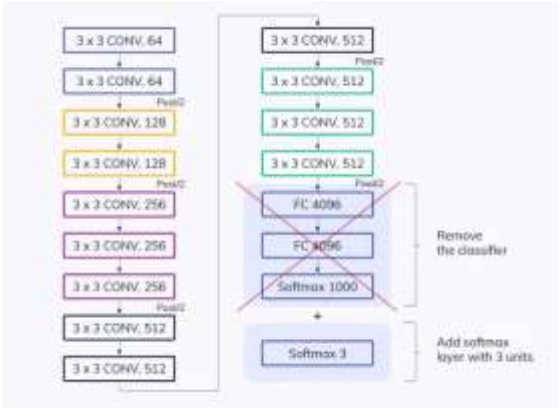
2)

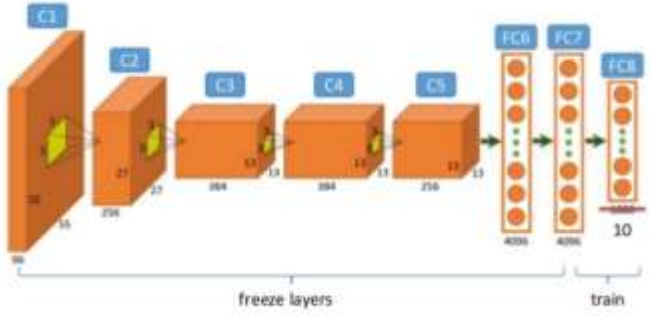




Transfer Learning Process

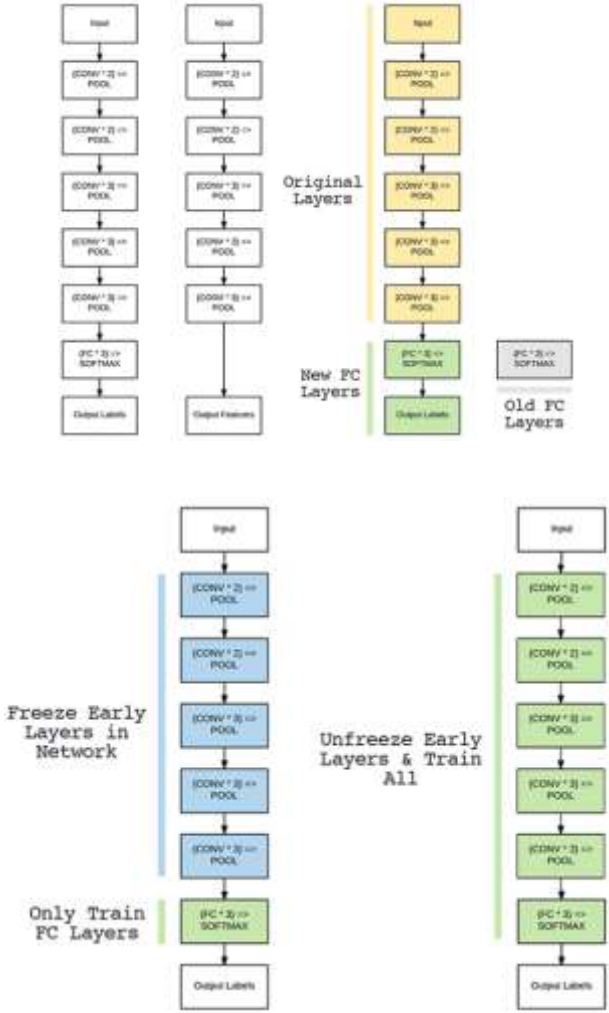
- **Access pre-trained models**
  - Can obtain pre-trained models from
    - Their own collection of model libraries
    - Other open-source repositories
  - Example - PyTorch Hub - open-source pre-trained model repository
    - Designed to speed up the research path, from prototyping to product deployment
  - 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





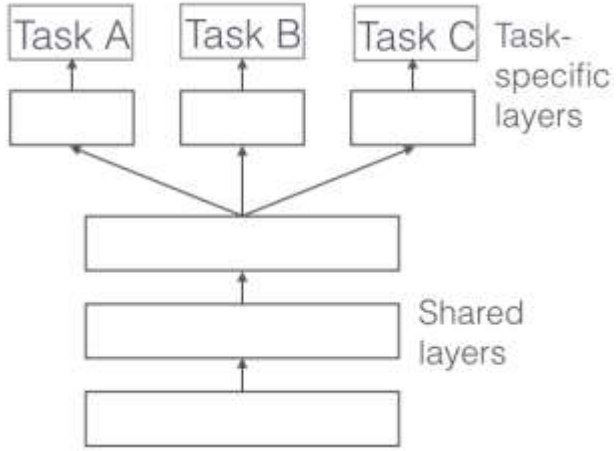
Transfer Learning Process

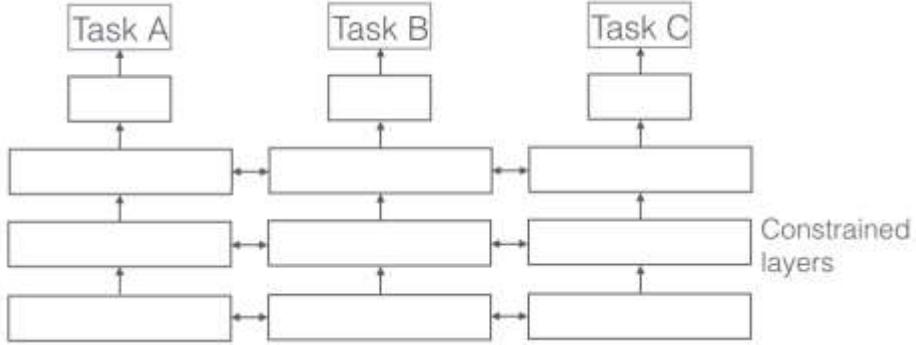
- **Add New Trainable Layers**
  - turn old features into predictions on the new dataset
  - Important because the pre-trained model is loaded without the final output layer.
- **Train the new layers on the dataset**
  - pre-trained model's final output will most likely be different from the output that 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

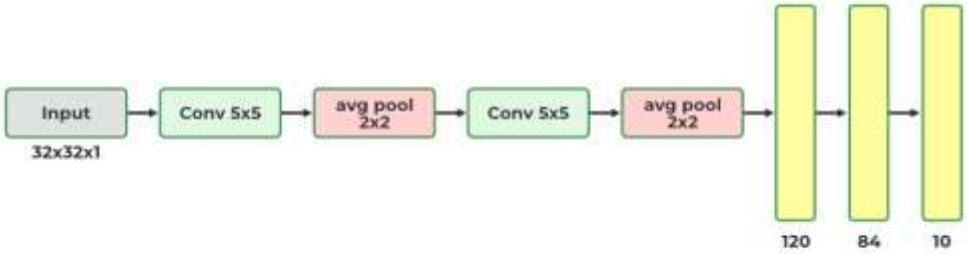
- Model training technique where you train a single deep neural network on multiple tasks at the same time
- Applications in
  - Machine Learning
  - NLP
  - Speech Precessing
  - Computer Vision
- **Hard parameter sharing**
- **Soft parameter sharing**





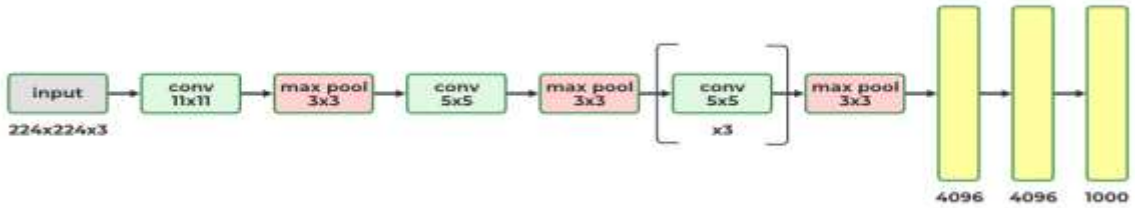
CNN Architectures – LeNet-5

- 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