Unit-3

CONVOLUTION NEURAL NETWORKS & TRANSFER LEARNING TECHNIQUES

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.

Architectural Overview:

Deep learning, there are several types of models such as the Artificial Neural Networks (ANN), Autoencoders, Recurrent Neural Networks (RNN) and Reinforcement Learning. But there has been one particular model that has contributed a lot in the field of computer vision and image analysis which is the Convolutional Neural Networks (CNN) or the ConvNets.

CNN is very useful as it minimises human effort by automatically detecting the features. For example, for apples and mangoes, it would automatically detect the distinct features of each class on its own.

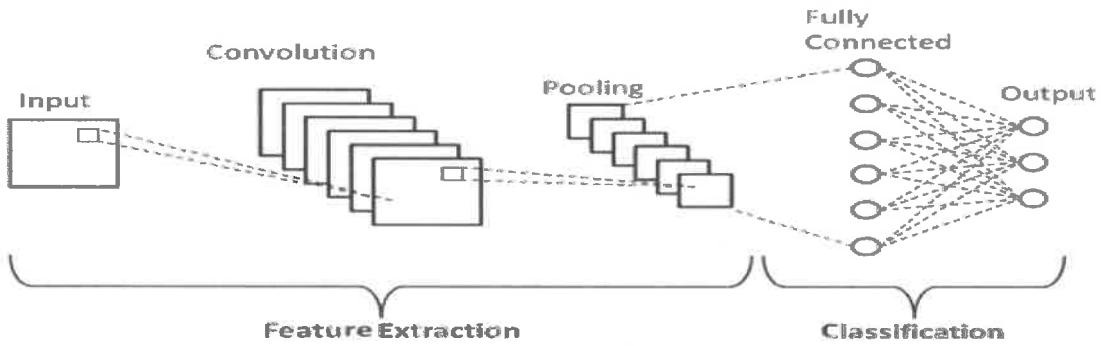
CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

Basic Architecture

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarises the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.



Convolution Layers

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolution layer in CNN passes the result to the next layer once applying the convolution operation in the input. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size n x n x n_c and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

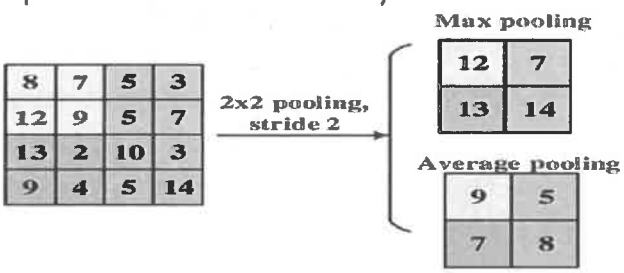
2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer.

In **Max Pooling**, the largest element is taken from feature map. **Average Pooling** calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in

Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

This CNN model generalises the features extracted by the convolution layer, and helps the networks to recognise the features independently. With the help of this, the computations are also reduced in a network.. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

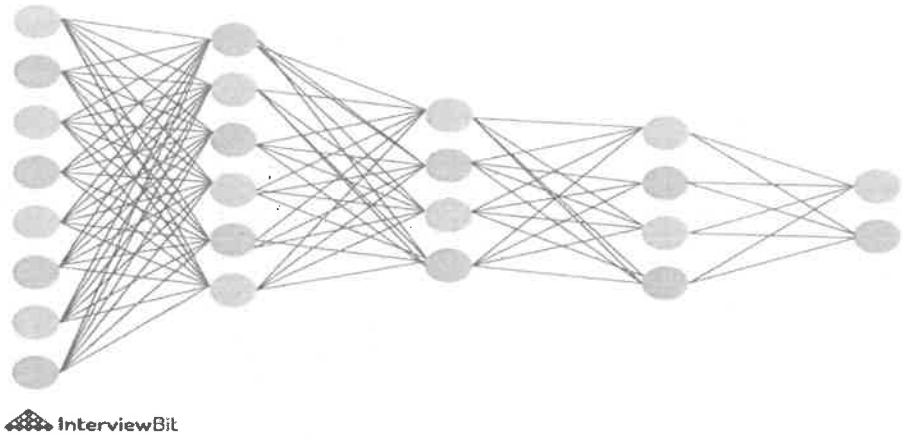


3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision.

The FC layer helps to map the representation between the input and the output.

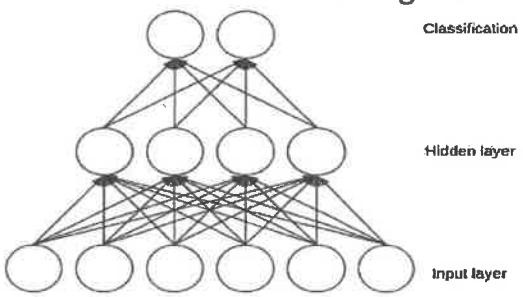


4. Dropout

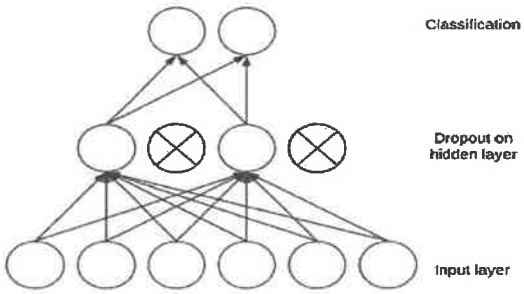
Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Dropout results in improving the performance of a deep learning model as it prevents overfitting by making the network simpler. It drops neurons from the neural networks during training.



Without Dropout



With Dropout

5. Activation Layer

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.

It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class

classification, generally softmax us used. In simple terms, activation functions in a CNN model determine whether a neuron should be activated or not. It decides whether the input to the work is important or not to predict using mathematical operations.

Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

1. Sigmoid

The sigmoid non-linearity has the mathematical form $\sigma(\kappa) = 1/(1+e^{-}\kappa)$. It takes a real-valued number and "squashes" it into a range between 0 and 1.

However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in backpropagation it will effectively "kill" the gradient. Also, if the data coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

2. Tanh

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

3. ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(\kappa)=\max(0,\kappa)$. In other words, the activation is simply threshold at zero.

In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.

Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

The major components of convolutional layers:

- Filters
- Activation maps
- Parameter sharing
- Layer-specific hyperparameters

1) Filters:

The parameters for a convolutional layer configure the layer's set of filters. Filters are a function that has a width and height smaller than the width and height of the input volume. Filter Sizes in Natural Language Processing Applications We can have a filter size equal to the input volume, but typically only in one dimension, not both. Filters (e.g., convolutions) are applied across the width and height of the input volume in a sliding window manner, as demonstrated in Figure below. Filters are also applied for every depth of the input volume. We compute the output of the filter by producing the dot product of the filter and the input region. The output of applying a filter to the input volume is known as the activation map (sometimes referred to as a feature map) of that filter. In many CNN diagrams, we often see lots of small activation maps; how these are produced can sometimes be confusing. The filter count is a hyperparameter value for each convolutional layer. This hyperparameter also controls how many activation maps are produced from the convolutional layer as input into the next layer and is considered the third dimension (number of activation maps) in the 3D layer output activation volume. The filter count hyperparameter can be chosen freely yet some values will work better than others.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 x 5 – Image Matrix

*

1	0	1
0	1	0
1	0	1

3 x 3 – Filter Matrix

Filtering Out Edges

- Edge detection filters based on the probability distribution of pixels in a certain dataset and the network's specific objective.
- They are the Prewitt, Sobel, Laplacian, Robinson Compass and Krisch Compass filters.

Prewitt Filters

The Prewitt operator is comprised of two filters which help to detect vertical and horizontal edges. The horizontal (x-direction) filter helps to detect edges in the image which cut perpendicularly through the horizontal axis and vise versa for the vertical (y-direction) filter.

Prewitt Filters

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

horizontal vertical

Sobel Filters

Detected edges are quite similar to results obtained using Prewitt filters but with a distinction of higher edge pixel intensity. In other words, edges detected using the Sobel filters are sharper in comparison to Prewitt filters.

Sobel Filters

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

horizontal vertical

Laplacian Filter

Unlike the Prewitt and Sobel filters, the Laplacian filter is a single filter which detects edges of different orientation. From a mathematical standpoint, it computes second order derivatives of pixel values unlike the Prewitt and Sobel filters which compute first order derivatives.

Laplacian Filter

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

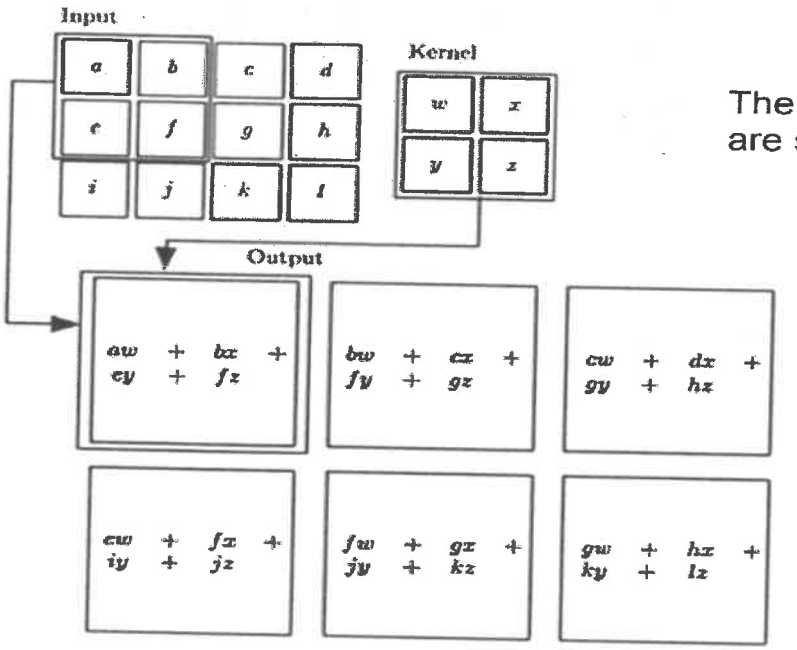
2)**Parameter sharing**

CNNs use a parameter-sharing scheme to control the total parameter count. This helps training time because we'll use fewer resources to learn the training dataset. To implement parameter sharing in CNNs, we first denote a single two-dimensional slice of depth as a "depth slice." We then constrain the neurons in each depth slice to use the same weights and bias. This gives us significantly fewer parameters (or weights) for a given convolutional layer.

- Parameter sharing is the method of sharing weights by all neurons in a particular feature map. Therefore helps to reduce the number of parameters in the whole system, making it computationally cheap.

Parameter sharing reduces the training time, which directly reduces the number of weight updates during backpropagation.

- Parameter sharing forces sets of parameters to be similar as we interpret various models or model components as sharing a unique set of parameters. We only need to store only a subset of memory.



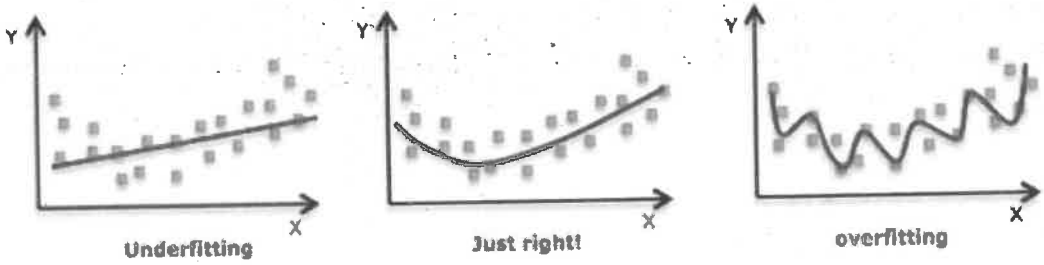
The kernel parameters are shared

- Suppose two models A and B, perform a classification task on similar input and output distributions. In such a case, we'd expect the parameters for both models to be identical to each other as well. We could impose a norm penalty on the distance between the weights, but a more popular method is to force the parameters to be equal. The idea behind Parameter Sharing is the essence of forcing the parameters to be similar. A significant benefit here is that we need to store only a subset of the parameters (e.g., storing only the parameters for model A instead of storing for both A and B), which leads to significant memory savings.

Regularization:

Regularization is a technique used in machine learning and deep learning to prevent overfitting and improve the generalization performance of a model. It involves adding a penalty term to the loss function during training. This penalty discourages the model from becoming too complex or having large parameter values, which helps in controlling the model's ability to fit noise in the training data. Regularization methods include L1 and L2 regularization, dropout, early stopping, and more. By applying regularization, models become more robust and better at making accurate predictions on unseen data

Before we deep dive into the topic, take a look at this image:



Have you seen this image before? As we move towards the right in this image, our model tries to learn too well the details and the noise from the training data, which ultimately results in poor performance on the unseen data.

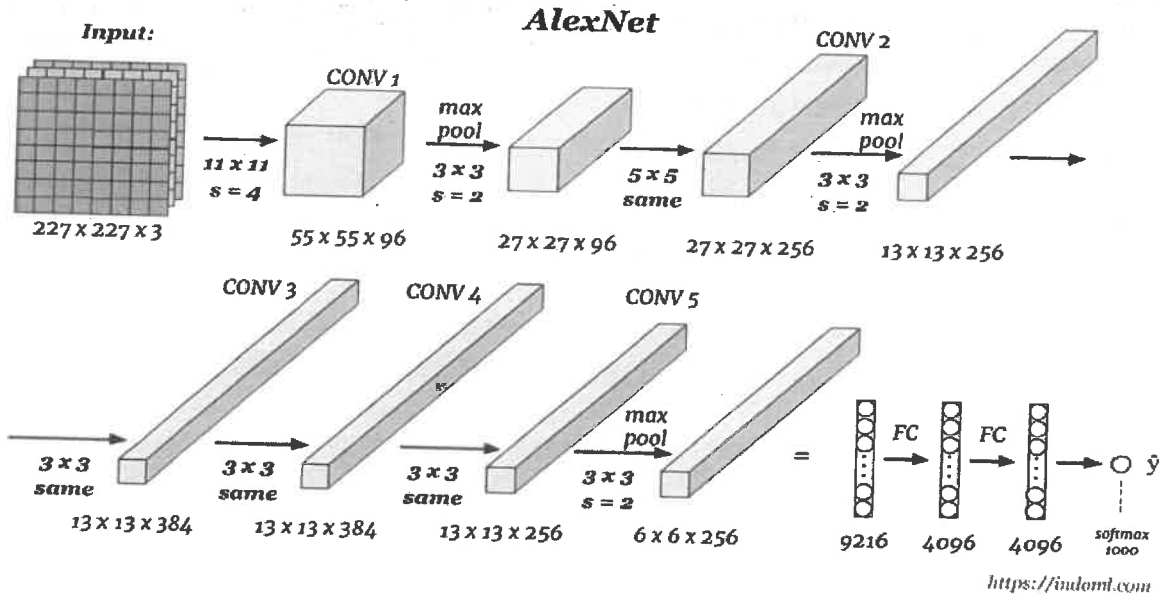
In other words, while going towards the right, the complexity of the model increases such that the training error reduces but the testing error doesn't. This is shown in the image below.

Regularization can take numerous forms, including:

- **L1 Regularization** imposes a penalty on the objective function that is proportional to the absolute value of the model's parameters. As a result, the model is sparse, with some of its parameters being exactly zero.
- **L2 Regularization** involves applying a penalty to the objective function that is proportional to the square of the model's parameters. As a result, the parameters in the model are small, but not necessarily zero.
- **Early Stopping**: Early stopping is a type of regularization that involves training the model until the performance on the validation set starts to degrade, and then stopping the training process. This helps to prevent the model from overfitting to the training data.
- **Dropout**: Dropout is a regularization technique that involves randomly setting a fraction of the model's neurons to zero during training. This helps to prevent the model from relying too heavily on any one neuron, which can help to reduce overfitting.

Popular CNN Architectures: ResNet, AlexNet–Application

AlexNet : introduces the ReLU activation function and LRN into the mix. ReLU becomes so popular that almost all CNN architectures developed after AlexNet used ReLU in their hidden layers, abandoning the use of tanh activation function in LeNet-5.



- Number of parameters: ~ 60 millions.

AlexNet architecture | Image by author

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

The network consists of 8 layers:

- 5 convolution layers with non-increasing kernel sizes, followed by
- 3 fully connected layers.

The last layer uses the softmax activation function, and all others use ReLU. LRN is applied on the first and second convolution layers after applying ReLU. The first, second, and fifth convolution layers are followed by a 3×3 max pooling.

➢ Similar to LeNet , but much bigger

- RELU is used as activation function
- Multiple GPU's
- Local Response Normalization (LRN)
- AlexNet speeds up the training by 10 times just by the use of GPU.

With the advancement of modern hardware, AlexNet can be trained with a whopping **60 million** parameters. AlexNet's architecture was extremely similar to LeNet's. It was the first convolutional network to employ the graphics processing unit (GPU) to improve performance. Convolutional filters and a non-linear activation function termed ReLU are used in each convolutional layer (Rectified Linear Unit). Max pooling is done using the pooling layers. Due to the presence of fully connected layers, the input size is fixed. The AlexNet architecture was created with large-scale image datasets in mind, and it produced state-of-the-art results when it was first released. It has 60 million characteristics in all.

Fully Connected and Dropout Layers

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
-	-	-	-	-	-	-
-	-	-	-	-	-	-
-	-	-	-	-	-	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

After this, we have our first dropout layer. The drop-out rate is set to be 0.5.

Finally, we have the last fully connected layer or output layer with 1000 neurons as we have 10000 classes in the data set. The activation function used at this layer is Softmax.

ResNet:

- ResNets or Residual networks are a type of deep convolutional neural network architecture.
- This refers to the residual blocks that make up the architecture of the network.

When deeper networks can start converging, a ***degradation problem*** has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.
Unexpectedly, such degradation is not caused by overfitting (usually indicated by lower training error and higher testing error) since adding more layers to a suitably deep network leads to higher *training error*.

What Is a Residual Block?
Residual blocks are an important part of the ResNet architecture. In older architectures such as VGG16, convolutional layers are stacked with batch normalization and nonlinear activation layers such as ReLu between them. This method works with a small number of convolutional layers—the maximum for VGG models is around 19 layers. However, subsequent research discovered that increasing the number of layers could significantly improve CNN performance.

The ResNet architecture introduces the simple concept of adding an intermediate input to the output of a series of convolution blocks. This is illustrated below.

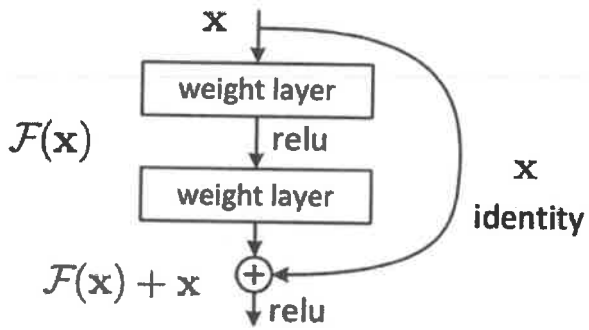


Figure 2. Residual learning: a building block.

The image above shows a typical residual block. This can be expressed in Python code using the expression *output = F(x) + x* where x is an input to the residual block and output from the previous layer, and F(x) is part of a CNN consisting of several convolutional blocks.

This technique of adding the input of the previous layer to the output of a subsequent layer is now very popular, and has been applied to many other neural network architectures including UNet and Recurrent Neural Networks (RNN).

ResNet 50 Architecture:

ResNet50 consists of 16 residual blocks. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks. ResNet-50 is based on a deep residual learning framework that allows for the training of very deep networks with hundreds of layers.

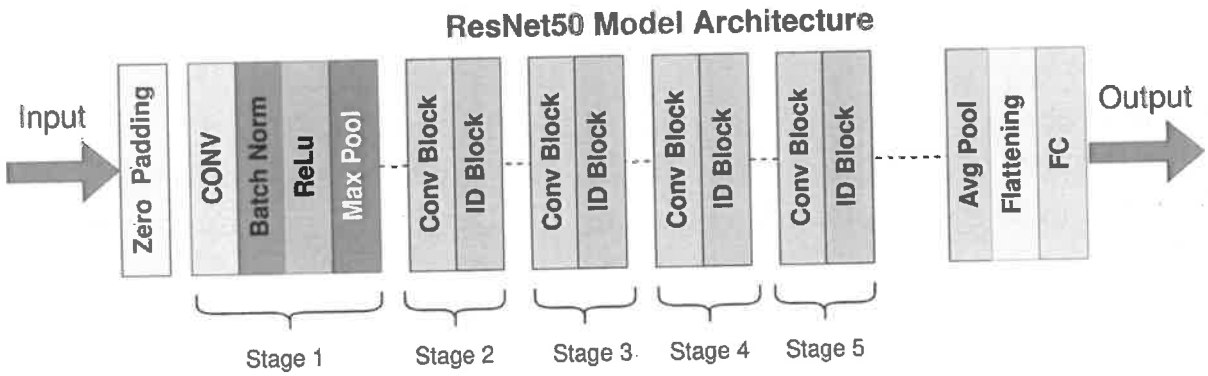
ResNet-50 consists of 50 layers that are divided into 5 blocks, each containing a set of residual blocks. The residual blocks allow for the preservation of information from earlier layers, which helps the network to learn better representations of the input data.

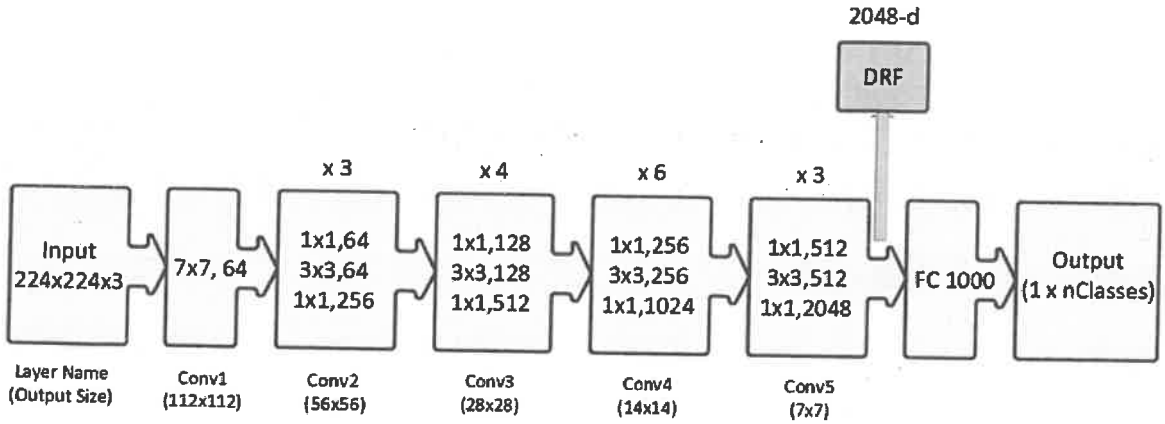
The degradation problem is addressed by introducing ***bottleneck residual blocks***. There are 2 kinds of residual blocks:

1. **Identity block:** consists of 3 convolution layers with 1×1, 3×3, and 1×1 kernel sizes, all of which are equipped with BN. The ReLU activation function is applied to the first two layers, while the input of the identity block is added to the last layer before applying ReLU.

2. **Convolution block:** same as identity block, but the input of the convolution block is first passed through a convolution layer with 1×1 kernel size and BN before being added to the last convolution layer of the main series.

3. **Batch Normalization**: Batch normalization is typically applied after the convolutional or fully connected layers in a neural network but before the activation function. It is a widely used technique in deep learning and has been shown to improve the performance of many types of neural networks. Faster convergence, Improved generalization, and Regularization are some benefits of batch normalization.





Notice that both residual blocks have 3 layers. In total, ResNet-50 has **26 million** parameters and 50 layers.

Special characteristics of ResNet-50

ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks

ResNet-50 has an architecture based on the model depicted above, but with one important difference. The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a "bottleneck", which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer. It uses a stack of three layers rather than two layers.

The 50-layer ResNet architecture includes the following elements, as shown in the table below:

- **One 7×7 kernel convolution** alongside 64 other kernels with a 2-sized stride followed by Batch Normalization, ReLU, Pooling.
- **A max pooling layer** with a 2-sized stride.
- **9 more layers**—3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times (i.e: 1 Convolution and 2 Identity layers= 3 layers)
- **12 more layers** with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times (i.e: 1 Convolution and 3 Identity layers= 4 layers)
- **18 more layers** with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times (i.e: 1 Convolution and 5 Identity layers= 6 layers)
- **9 more layers** with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times (i.e: 1 Convolution and 2 Identity layers= 3 layers).

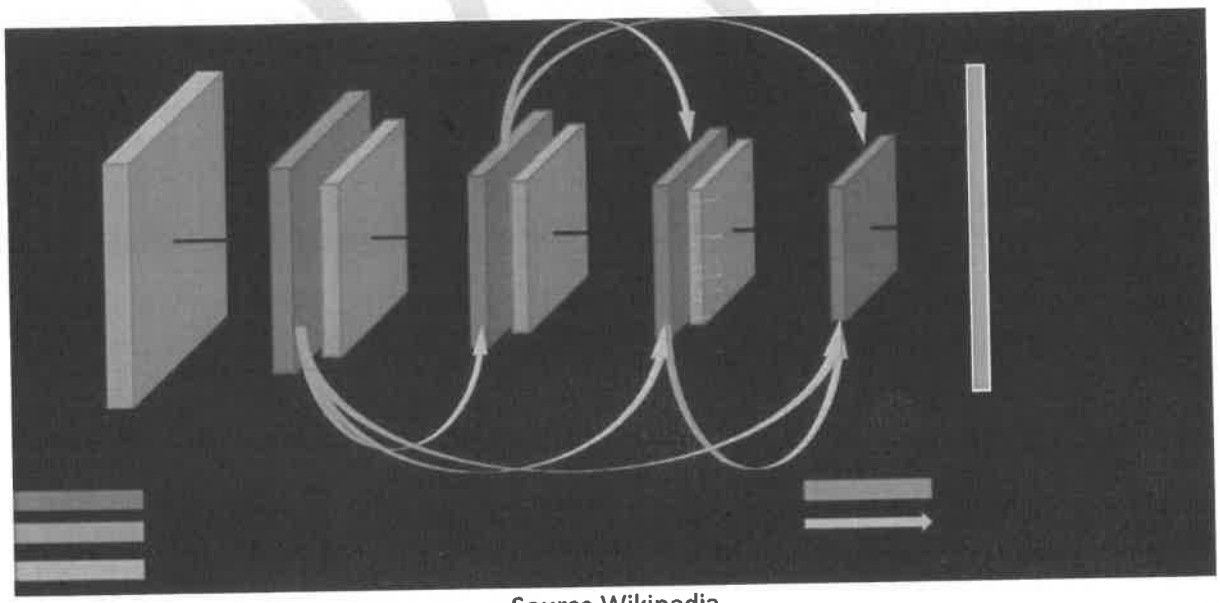
(up to this point the network has 50 layers)

- **Average pooling**, followed by a **one** fully connected layer with 1000 nodes, using the softmax activation function.

Variants of Dense Net:

Dense net is densely connected-convolutional networks. It is very similar to a ResNet with some-fundamental differences. ResNet is using an additive method that means they take a previous output as an input for a future layer, & in DenseNet takes all previous output as an input for a future layer as shown in the above image. So DenseNet was specially developed to improve accuracy caused by the vanishing gradient in high-level neural networks due to the long distance between input and output layers & the information vanishes before reaching its destination.

Here we're going to summarize a convolutional-network architecture called densely-connected-convolutional networks or **DenseNet**. So the problem that they're trying to solve with the density of architecture is to increase the depth of the convolutional neural network.



Source Wikipedia

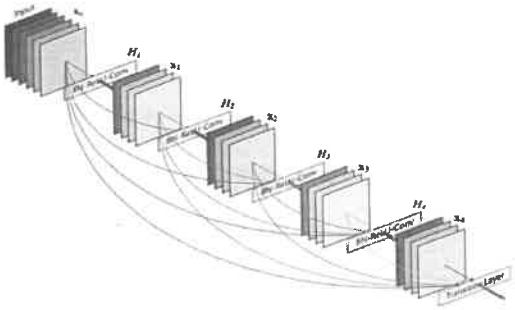
Here we first learn about what is a dense net and why this is useful and then we go with a coding part.

So we all know about a CNN (convolutional neural network) which is useful for image classification. So dense net is densely connected-convolutional networks. It is very similar to a ResNet with some-fundamental differences. ResNet is using

an additive method that means they take a previous output as an input for a future layer, & in DenseNet takes all previous output as an input for a future layer as shown in the above image.

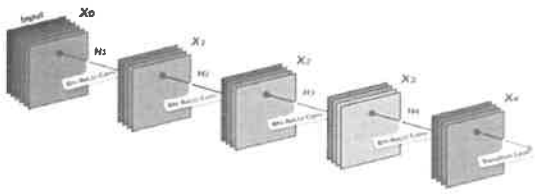
So DenseNet was specially developed to improve accuracy caused by the vanishing gradient in high-level neural networks due to the long distance between input and output layers & the information vanishes before reaching its destination.

DenseNet Architecture VS ResNet Architecture.



DenseNet Structure

$$a^{[l]} = g([a^{[0]}, a^{[1]}, a^{[2]}, \ldots\ldots\ldots\ldots, a^{[l-1]})$$

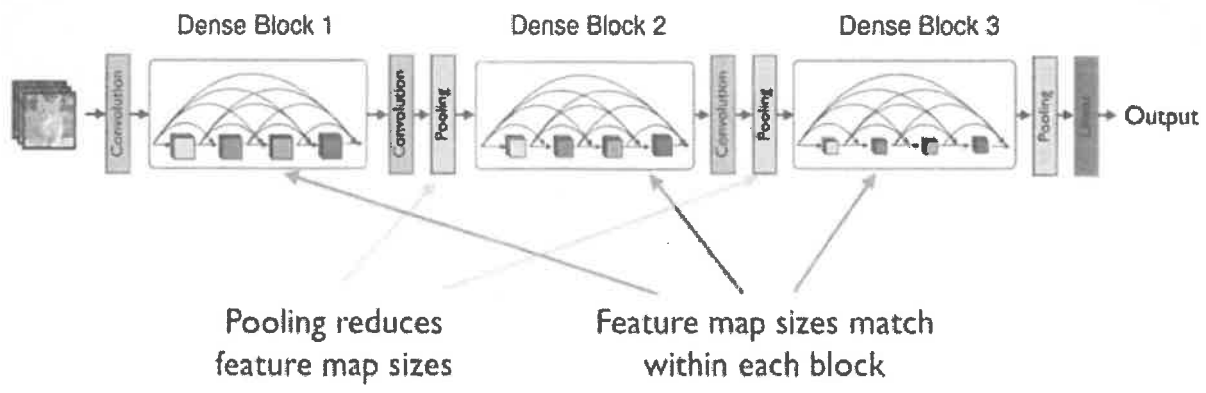


ResNet Structure

$$a^{[l]} = g(z^{[l+1]} + a^{[l]})$$

So suppose we have a capital L number of layers, In a typical network with L layers, there will be L connections, that is, connections between the layers. However, in a DenseNet, there will be about L and L plus one by two connections L(L+1)/2. So in a dense net, we have less number of layers than the other model, so here we can train more than 100 layers of the model very easily by using this technique.

DenseBlocks And Layers



Here as we go deeper into the network this becomes a kind of unsustainable, if you go 2nd layer to 3rd layer so 3rd layer takes an input not only 2nd layer but it takes input all previous layers.

So let's say we have about ten layers. Then the 10th layer will take us to input all the feature maps from the preceding nine layers. Now, if each of these layers, let's produce 128 feature maps and there is a feature map explosion. to overcome this problem we create a dense block here and So each dense block contains a prespecified number of layers inside them and the output from that particular dense block is given to what is called a transition layer and this layer is like one by one convolution followed by Max pooling to reduce the size of the feature maps. So the transition layer allows for Max pooling, which typically leads to a reduction in the size of your feature maps.

As a given fig, we can see two blocks first one is the convolution layer and the second is the pooling layer, and combinations of both are the transition layer.

Advantages:

- Strong Gradient flow
- Efficiency
- Standard connectivity
- Dense connectivity
- Growth rate
- Parameter efficiency

Applications:

- NLP
- Medical Images
- Audio
- Image
- Semantics Segmentation

PixelNet

PixelNet is a neural network architecture designed for pixel-level prediction tasks, such as semantic segmentation, surface normal estimation, and edge

detection. It uses a convolutional neural network to process spatial information and achieve state-of-the-art results across diverse pixel-labeling tasks.

The stratified sampling of pixels allows one to:

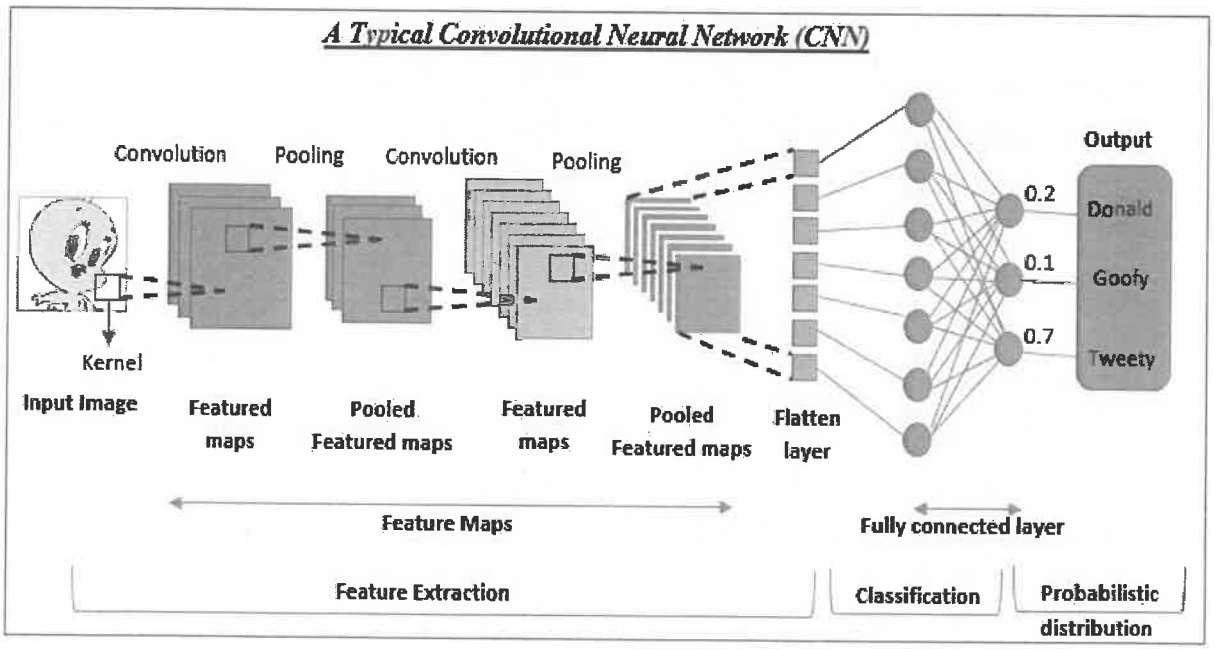
- add diversity during batch updates, speeding up learning;
- explore complex nonlinear predictors, improving accuracy;
- efficiently train state-of-the-art models *tabula rasa* (i.e., *from scratch*) for diverse pixel-labeling tasks.

PixelNet with Convolutional Neural Networks (CNNs):

- This architecture, proposed in 2016, revolutionized pixel-level prediction tasks in deep learning. Instead of predicting for the entire image at once, PixelNet focuses on individual pixels, leading to fine-grained and accurate predictions. PixelNet with CNNs is a powerful tool for pixel-level prediction tasks, offering high accuracy and flexibility.

Key components of PixelNet with CNNs:

1. **Convolutional Layers:**
   - Stacked convolutional layers extract features from the image at different scales and resolutions.
   - These layers act like feature detectors, identifying edges, textures, and other patterns relevant to the prediction task.



2. **Non-linear Activations:**

- Non-linear activation functions like ReLU are applied after each convolutional layer.
- These functions introduce non-linearity, allowing the network to learn complex relationships between features.

3. **Pooling Layers (Optional):**
   - Pooling layers (like max pooling) can be used to reduce the spatial dimensionality of the feature maps.
   - This can help control the model's complexity and prevent overfitting.
4. **Dense Layers:**
   - The final layers of the network are usually dense layers, where each neuron in the previous layer is connected to every neuron in the current layer.
   - These layers take the extracted features and make predictions for each pixel in the image.
5. **Loss Function and Optimization**:
   - A loss function, like cross-entropy for classification or L2 loss for regression, measures the difference between the network's predictions and the ground truth labels.
   - An optimization algorithm, like Adam or SGD, adjusts the network's weights to minimize the loss function and improve prediction accuracy.

Advantages

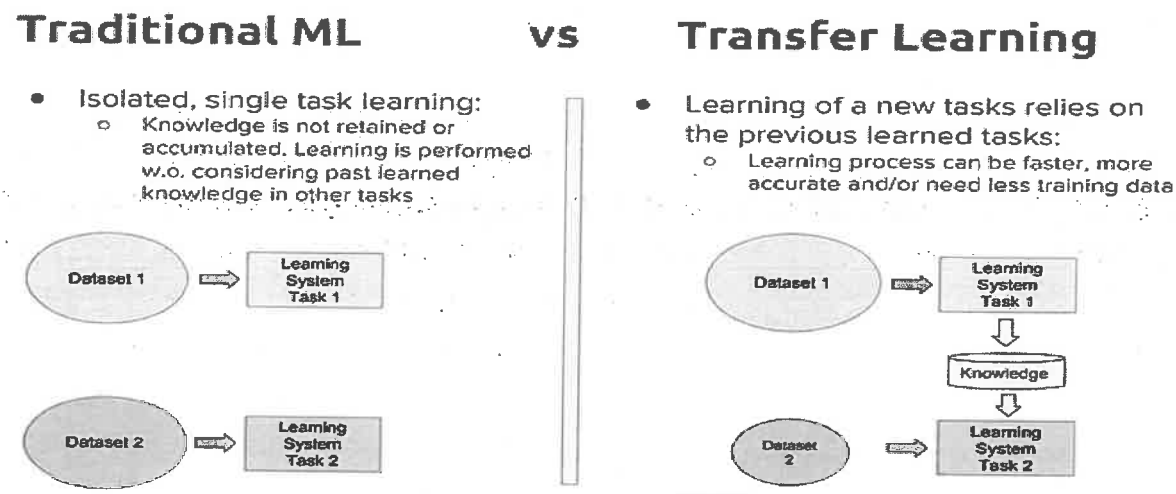
- High Accuracy
- Flexibility
- Data efficiency

Applications:

- Autonomous Driving
- Medical Imaging
- Robotics

Transfer Learning Techniques:

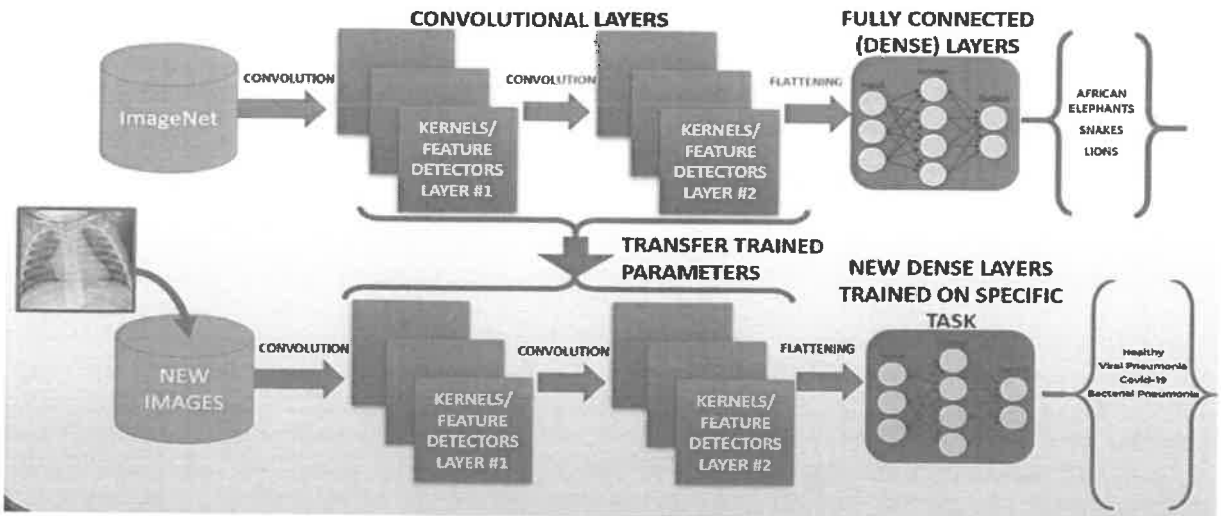
The first thing to remember here is that, transfer learning, is not a new concept which is very specific to deep learning. There is a stark difference between the traditional approach of building and training machine learning models, and using a methodology following transfer learning principles.



Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

What is Transfer Learning?

Transfer learning is a technique in machine learning where a model trained on one task is used as the starting point for a model on a second task. This can be useful when the second task is similar to the first task, or when there is limited data available for the second task. By using the learned features from the first task as a starting point, the model can learn more quickly and effectively on the second task. This can also help to prevent overfitting, as the model will have already learned general features that are likely to be useful in the second task.



- From above figure the first CNN layers are used to extract high level general features
- The last couple of layers are used to perform classification(on a specific task)
- So we copy the first trained layers (base model) and then we add a new custom layers in the output to perform classification on a specific new task.

How does Transfer Learning work?

- **Pre-trained Model:** Start with a model that has previously been trained for a certain task using a large set of data. Frequently trained on extensive datasets, this model has identified general features and patterns relevant to numerous related jobs.
- **Feature Extraction:**
  Use the pre-trained model's lower layers, which extract basic features like edges and textures, and freeze their weights during fine-tuning. Replace the final layers with new ones specific to the target task. This leverages the general feature extraction capability while adapting to the new task.
- **Base Model:** The model that has been pre-trained is known as the base model. It is made up of layers that have utilized the incoming data to learn hierarchical feature representations.
- **Transfer Layers:** In the pre-trained model, find a set of layers that capture generic information relevant to the new task as well as the previous one. Because they are prone to learning low-level information, these layers are frequently found near the top of the network.
- **Fine-tuning:** Using the dataset from the new challenge to retrain the chosen layers. We define this procedure as fine-tuning. The goal is to preserve the knowledge from the pre-training while enabling the model to modify its parameters to better suit the demands of the current assignment.

Benefits:

- Reduce the amount of data required for training
- Improves generalization
- Save time and resources
- Improves model performances

Drawbacks:

- Overfitting
- Limited flexibility
- Limited transparency
- Requires domain expertise