

Chapter-5

More Deep Learning Models, Architectures & Applications

Instructor Name: B N V Narasimha Raju

5.1 Alexnet

AlexNet is a classic convolutional neural network (CNN) that gained prominence after its success in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It consists of eight layers, including five convolutional layers and three fully connected layers, using traditional stacked convolutional layers with max-pooling in between. This deep network structure allows AlexNet to effectively extract complex features from images.

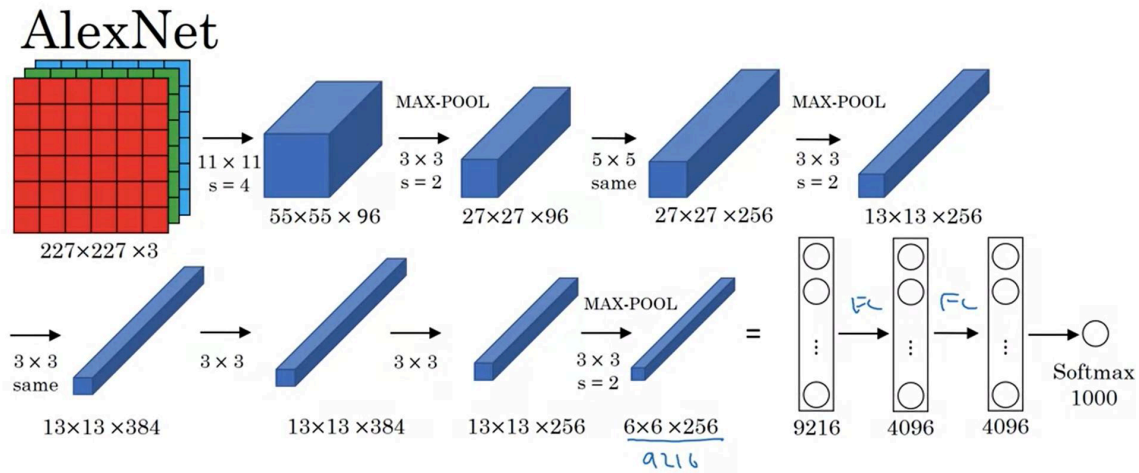
A key feature of its architecture is the use of overlapping pooling layers, which help reduce spatial dimensions while retaining spatial relationships among neighboring features. AlexNet also employs the ReLU (Rectified Linear Unit) activation function, which captures nonlinear relationships within the data and speeds up training by avoiding the vanishing gradient problem. The ReLU activation function is used after every convolutional and fully connected layer, except for the final output layer which uses softmax. Dropout regularization is applied to the fully connected layers to prevent overfitting, which is crucial due to the large number of parameters in the network.

One of the notable innovations of AlexNet was its computational efficiency; it introduced the use of parallel computing by utilizing two GPUs during training, enabling faster processing and handling large-scale data. In the 2012 ImageNet challenge, AlexNet produced groundbreaking results, outpacing previous CNN architectures and setting the stage for the resurgence of deep learning in computer vision.

Several architectural improvements, including the use of ReLU, overlapping pooling, and dropout, were key to its success, enhancing the model's performance and generalization ability. The network architecture is given below:

Model Explanation: The model described uses an input image of size 227x227x3, representing an RGB image with a height and width of 227 pixels. The first layer is a convolutional layer with 96 filters of size 11x11, applied with 'valid' padding and a stride of 4. The formula for the output size is given by the equation:

$$\text{floor}(((n + 2 * \text{padding} - \text{filter}) / \text{stride}) + 1) * \text{floor}(((n + 2 * \text{padding} - \text{filter}) / \text{stride}) + 1)$$



This formula is for square input with $height = width = n$. Explaining the first Layer with input 227x227x3 and Convolutional layer with 96 filters of 11x11, 'valid' padding and stride = 4, output dims will be

$$\begin{aligned} &= \text{floor}(((227 + 0 - 11) / 4) + 1) * \text{floor}(((227 + 0 - 11) / 4) + 1) \\ &= \text{floor}((216 / 4) + 1) * \text{floor}((216 / 4) + 1) \\ &= \text{floor}(54 + 1) * \text{floor}(54 + 1) \\ &= 55 * 55 \end{aligned}$$

For an input size of 227 and filter size of 11, the output dimensions become 55x55. Since there are 96 filters, the output of this convolutional layer is 55x55x96. This is followed by a max-pooling layer with a 3x3 filter and a stride of 2, reducing the output dimensions to 27x27x96. The next convolutional layer uses 256 filters of size 5x5 with 'same' padding, keeping the dimensions at 27x27x256.

Another max-pooling layer follows, reducing the size further to 13x13x256. The model then applies two convolutional layers sequentially, both using 384 filters of size 3x3 with 'same' padding, producing an output of 13x13x384. The model then applies a convolutional layer using 256 filters resulting in an output of 13x13x256.

A final max-pooling layer reduces the size to 6x6x256. The output is then flattened into a vector of 9216 units (6x6x256). This is followed by two fully connected layers with 4096 units each, and finally, a softmax layer with 1000 units that classifies the input image into one of 1000 classes, providing probabilities for each class.

5.2 VGGNet

The Visual Geometry Group (VGG) models, particularly VGG-16 and VGG-19, have significantly influenced the field of computer vision since their inception. These models stood out for their deep convolutional neural networks (CNNs) with a uniform architecture. VGG-19, the deeper variant of the VGG models, has garnered considerable attention due to its simplicity and effectiveness.

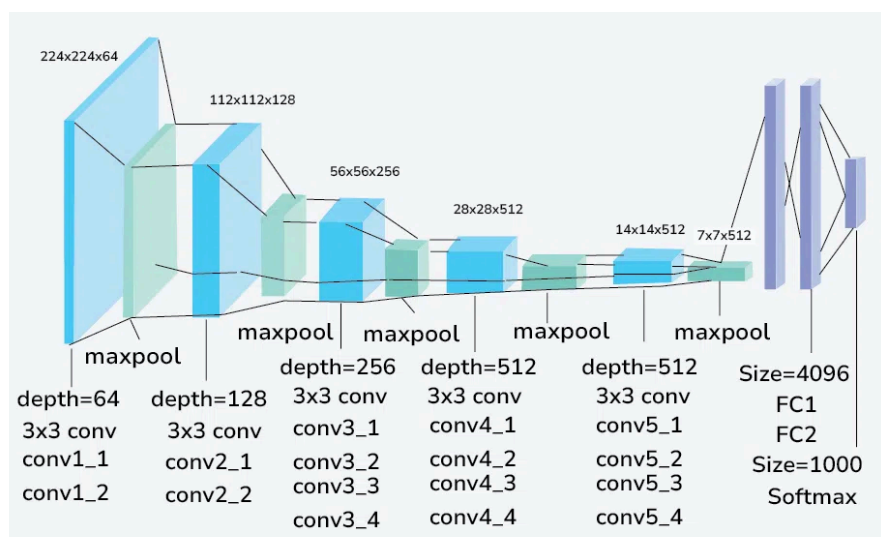
5.2.1 VGG-19 Architecture

VGG-19 is a deep convolutional neural network with 19 weight layers, comprising 16 convolutional layers and 3 fully connected layers. The architecture follows a straightforward and repetitive pattern, making it easier to understand and implement.

The key components of the VGG-19 architecture are:

- Convolutional Layers: 3x3 filters with a stride of 1 and padding of 1 to preserve spatial resolution.
- Activation Function: ReLU (Rectified Linear Unit) applied after each convolutional layer to introduce non-linearity.
- Pooling Layers: Max pooling with a 2x2 filter and a stride of 2 to reduce the spatial dimensions.
- Fully Connected Layers: Three fully connected layers at the end of the network for classification.
- Softmax Layer: Final layer for outputting class probabilities.

The VGG-19 model consists of five blocks of convolutional layers, followed by three fully connected layers. Here is a detailed breakdown of each block:



- Block 1
 - Conv1_1: 64 filters, 3x3 kernel, ReLU activation
 - Conv1_2: 64 filters, 3x3 kernel, ReLU activation
 - Max Pooling: 2x2 filter, stride 2
- Block 2
 - Conv2_1: 128 filters, 3x3 kernel, ReLU activation
 - Conv2_2: 128 filters, 3x3 kernel, ReLU activation
 - Max Pooling: 2x2 filter, stride 2
- Block 3
 - Conv3_1: 256 filters, 3x3 kernel, ReLU activation
 - Conv3_2: 256 filters, 3x3 kernel, ReLU activation
 - Conv3_3: 256 filters, 3x3 kernel, ReLU activation
 - Conv3_4: 256 filters, 3x3 kernel, ReLU activation
 - Max Pooling: 2x2 filter, stride 2
- Block 4
 - Conv4_1: 512 filters, 3x3 kernel, ReLU activation
 - Conv4_2: 512 filters, 3x3 kernel, ReLU activation
 - Conv4_3: 512 filters, 3x3 kernel, ReLU activation
 - Conv4_4: 512 filters, 3x3 kernel, ReLU activation
 - Max Pooling: 2x2 filter, stride 2
- Block 5
 - Conv5_1: 512 filters, 3x3 kernel, ReLU activation
 - Conv5_2: 512 filters, 3x3 kernel, ReLU activation
 - Conv5_3: 512 filters, 3x3 kernel, ReLU activation
 - Conv5_4: 512 filters, 3x3 kernel, ReLU activation
 - Max Pooling: 2x2 filter, stride 2
- Fully Connected Layers
 - FC1: 4096 neurons, ReLU activation
 - FC2: 4096 neurons, ReLU activation
 - FC3: 1000 neurons, softmax activation (for 1000-class classification)

5.2.2 Architectural Design Principles

The VGG-19 architecture follows several key design principles:

- Uniform Convolution Filters: Consistently using 3x3 convolution filters simplifies the architecture and helps maintain uniformity.
- Deep Architecture: Increasing the depth of the network enables learning more complex features.
- ReLU Activation: Introducing non-linearity helps in learning complex patterns.
- Max Pooling: Reduces the spatial dimensions while preserving important features.
- Fully Connected Layers: Combines the learned features for classification.

5.2.3 Impact and Legacy of VGG-19

VGG-19's primary legacy is demonstrating that increasing network depth with a simple, uniform architecture significantly boosts image recognition performance.

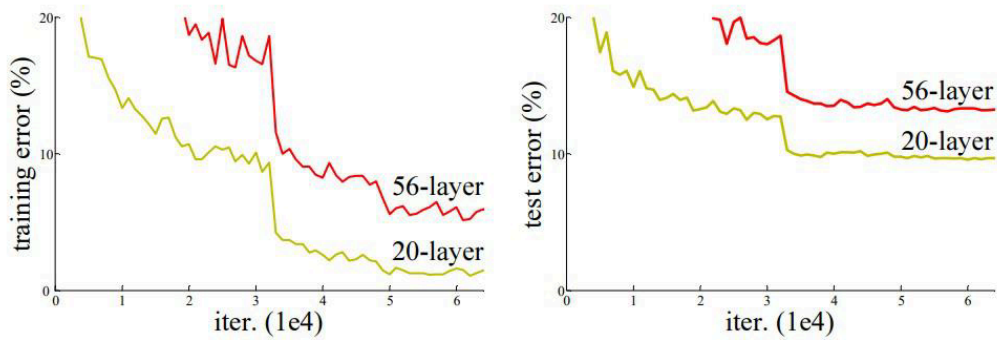
- **Architectural Influence:** Its design, which exclusively uses small 3x3 convolutional filters, became a foundational principle for subsequent advanced models like ResNet and Inception.
- **Transfer Learning Powerhouse:** It is extensively used as a pre-trained feature extractor. Models trained on ImageNet are fine-tuned for diverse tasks like object detection, medical imaging, and style transfer.
- **Research Benchmark:** It serves as a crucial baseline in academic and industry research for comparing and validating new architectures.

5.3 ResNet

ResNet, short for Residual Network, is a deep neural network architecture introduced by Kaiming He et al. in 2015. It became highly successful due to its use of residual (or skip) connections, which help mitigate the vanishing gradient problem and enable training of much deeper networks. In the context of Faster R-CNN, replacing the VGG-16 backbone with ResNet-101 resulted in a significant improvement, with detection accuracy increasing by approximately 28%. Additionally, ResNet demonstrated the feasibility of training networks with exceptionally deep architectures, such as 100-layer and even 1000-layer models, without suffering from the typical issues encountered in deep learning, like degradation in performance. The incorporation of residual connections allowed ResNet to maintain manageable levels of training error even in very deep networks.

5.3.1 Need for ResNet

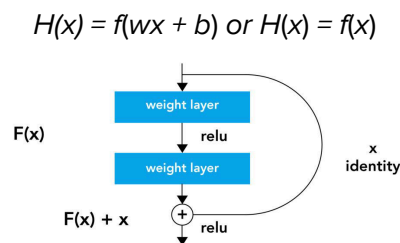
In deep learning, increasing the depth of a network by adding more layers is often considered a strategy for addressing complex problems, as it allows the network to learn more complex and abstract features. For example, in image recognition tasks, initial layers may detect simple features like edges, while deeper layers may identify more complex structures like textures and objects. This approach typically improves accuracy and performance. However, in traditional Convolutional Neural Networks (CNNs), there is a threshold beyond which adding more layers can lead to performance degradation. This phenomenon is illustrated in the plot, which compares the error percentages of a 20-layer network to a 56-layer network.




As shown in the plot, the error percentage for the 56-layer network is higher than that of the 20-layer network for both training and testing data. This suggests that increasing the depth of the network beyond a certain point may not necessarily lead to better performance. Several factors could contribute to this decline, including issues with the optimization function, improper network initialization, and the vanishing gradient problem. While overfitting might be a potential concern in some cases, it is not the cause in this scenario. Overfitting typically results in low training error but higher testing error, while the 56-layer network shows higher error percentages for both training and testing data, indicating that the performance degradation is likely due to the network's excessive depth rather than overfitting. ResNet addresses these issues by using residual connections or skip connections, which allow gradients to flow more easily through the network during training.

5.3.2 Residual Block

The challenge of training very deep networks has been significantly alleviated with the introduction of ResNet (Residual Networks), which are composed of residual blocks. The key feature of ResNet is the presence of a "skip connection" that bypasses one or more layers, allowing the output of a layer to be directly added to the output of a deeper layer. This skip connection is a core component of residual blocks. In traditional networks, the output of a layer is calculated by multiplying the input x by the layer's weights, adding a bias term, and passing the result through an activation function $f()$ to obtain the output, $H(x)$. This can be represented as:



However, with the skip connection in ResNet, the output becomes $H(x)=f(x)+x$, where the input x is added directly to the output of the activation function. A potential issue with this approach



arises when the dimensions of the input x differ from those of the output, which can occur due to operations like convolution or pooling. In such cases, two solutions can address the dimensional mismatch:

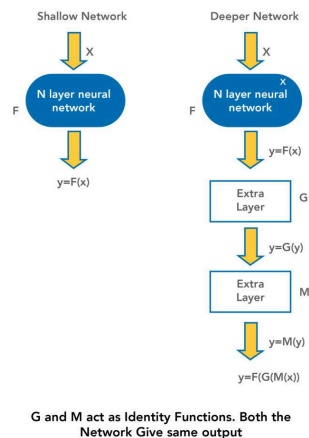
- **Padding**, where the skip connection is padded with zeros to match the dimensions of the output.
- **Projection**, where a 1×1 convolutional layer is used to match the dimensions, introducing an additional parameter w_1 . In this case, the output becomes: $H(x) = f(x) + w_1 \cdot x$

The projection method differs from padding in that it introduces extra parameters, providing more flexibility for dimension matching without losing information.

5.3.3 How ResNet helps

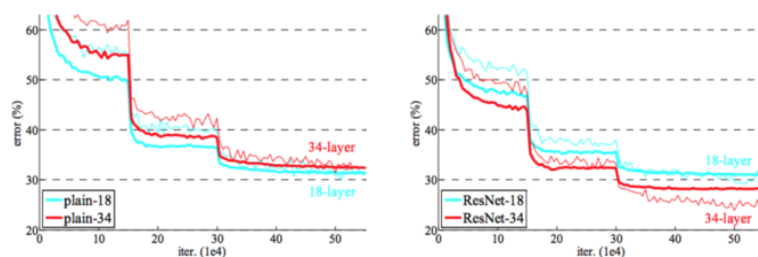
ResNet helps address the vanishing gradient problem in deep neural networks through the use of skip connections. These connections provide an alternate shortcut path for the gradient to flow through, allowing gradients to bypass certain layers and reach deeper layers more easily. This ensures that the network can continue to learn effectively, even with a deeper architecture. Additionally, skip connections enable the model to learn identity functions, meaning that higher layers can perform at least as well as lower layers, preventing performance degradation that typically occurs in plain neural networks without residual blocks.

To understand this more clearly, consider a shallow network and a deep network, both mapping an input ' x ' to an output ' y ' using the function $H(x)$. Ideally, the deep network should perform at least as well as the shallow network, without any degradation in performance. One way to achieve this is for the additional layers in the deep network to learn the identity function, where their outputs equal their inputs, ensuring no loss in performance despite the added depth. In traditional networks, learning the identity function is difficult, as the output is $H(x) = f(x)$, and for the identity function to hold, $f(x)$ must equal x . In contrast, ResNet simplifies this by introducing skip connections, where the output becomes $H(x) = f(x) + x$. Now, for the network to learn the identity function, all that is needed is for $f(x)$ to equal zero, making it easier to achieve $H(x) = x$.



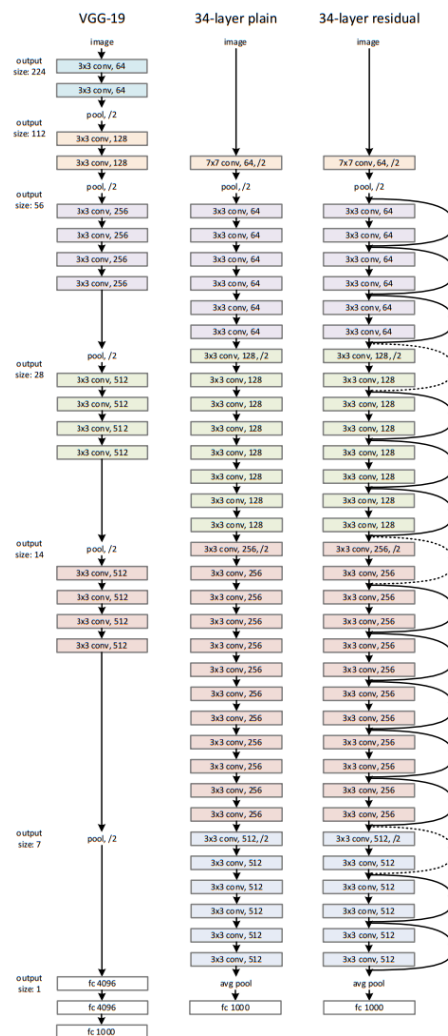
Skip connections and identity functions are crucial components of ResNet, allowing the network to train deeper architectures without suffering from the vanishing gradient problem. The use of residual blocks helps gradients flow more effectively through the network, enabling it to learn from information at different depths. Furthermore, the introduction of identity functions allows the network to focus on learning residual functions, the difference between the input and the desired output, rather than the entire mapping, which is easier to optimize. In the best-case scenario, deeper networks in ResNet can better approximate the mapping from 'x' to 'y' than shallower networks, reducing error rates significantly.

As demonstrated in a plot comparing error rates, ResNet-34 (with 34 layers) achieves a much lower error rate compared to a plain 34-layer network. For 18-layer networks, the error rates of plain-18 and ResNet-18 are nearly identical, indicating that ResNet's benefits are more pronounced in deeper networks. This illustrates how ResNet has dramatically improved the performance of deeper networks by effectively addressing the vanishing gradient problem and enabling the optimization of much deeper architectures.



5.3.4 ResNet Architecture

The ResNet network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into the residual network as shown in the figure below:



5.4 Transfer learning

Traditional machine learning involves isolated, single-task learning, where each model is trained independently on a specific dataset for a specific task. The knowledge acquired during training is not retained or reused for future tasks, meaning each new task requires training a model from scratch—often demanding large amounts of labeled data. This can become inefficient and ineffective, especially when the new task lacks sufficient data.

Transfer learning, on the other hand, is a machine learning technique that allows a model trained on one task to be repurposed for a second, related task. The core idea is to leverage the knowledge—such as learned features or model weights—from a previously trained model to improve learning on a new but related task. For instance, if a model has been trained to identify objects in restaurant images (Task T1), its learned features—like edges, shapes, and textures—can be transferred to help recognize objects in park or café images (Task T2), even

when there is less data available for T2. This process can lead to faster training, improved accuracy, and reduced data requirements for the new task.

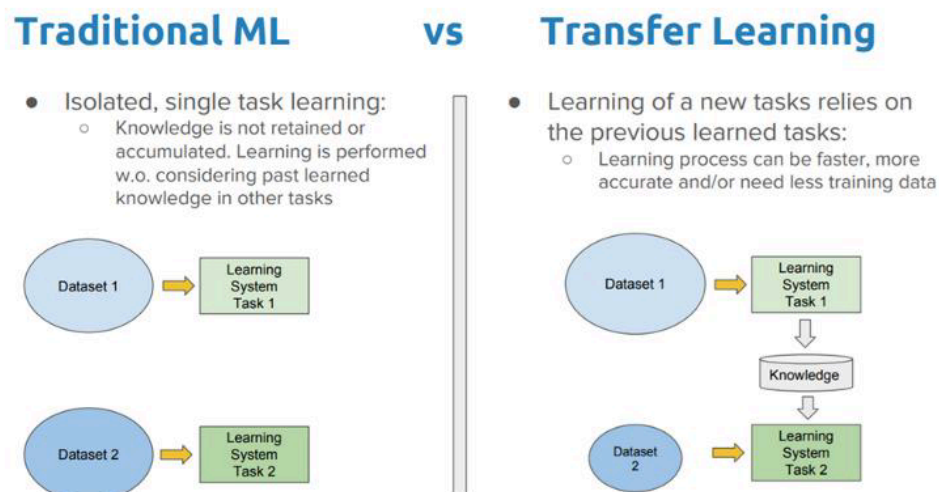


Figure: Traditional ML vs Transfer Learning

The above figure illustrates the key differences between the two approaches to machine learning. Traditional machine learning is shown as an isolated learning process. Each task—represented by Dataset 1 and Dataset 2—is learned separately using independent models for Task 1 and Task 2. There is no sharing or retention of knowledge between tasks. The model trained on Dataset 1 cannot assist or enhance the training of Dataset 2. This approach becomes inefficient when data is limited, as every new task starts from scratch without leveraging prior experience.

In contrast, transfer learning enables the reuse of knowledge from a previously learned task to enhance performance on a new, related task. The model is first trained on Dataset 1 for Task 1, gaining valuable knowledge—such as learned features or network weights. This knowledge is then transferred to assist in learning Task 2 using Dataset 2. As a result, the second task benefits from a more informed starting point, leading to faster training, reduced data needs, and better accuracy.

In the realm of deep learning, transfer learning is typically implemented by pretraining a neural network on a large benchmark dataset (e.g., ImageNet) and then fine-tuning it on a target dataset. This method is widely used because training deep networks from scratch is often costly in terms of time and computational resources. Transfer learning has proven to be highly effective for rapid prototyping, performance improvement, and resource-efficient training—especially when labeled data is limited. By mimicking the human ability to apply prior knowledge to new contexts, transfer learning brings a more adaptive and scalable approach to machine learning.

5.4.1 Transfer Learning Base Models

To design an efficient neural network using transfer learning, it's important to understand the various base models available. These base models serve as the foundation from which knowledge—referring to the network architecture and learned weights—is transferred to a new task-specific model. Broadly, base models can be categorized into two groups: the first is based on convolutional neural network (CNN) architectures commonly used in computer vision and classification tasks; the second is based on models designed for sequential data, primarily used in natural language processing (NLP) tasks.

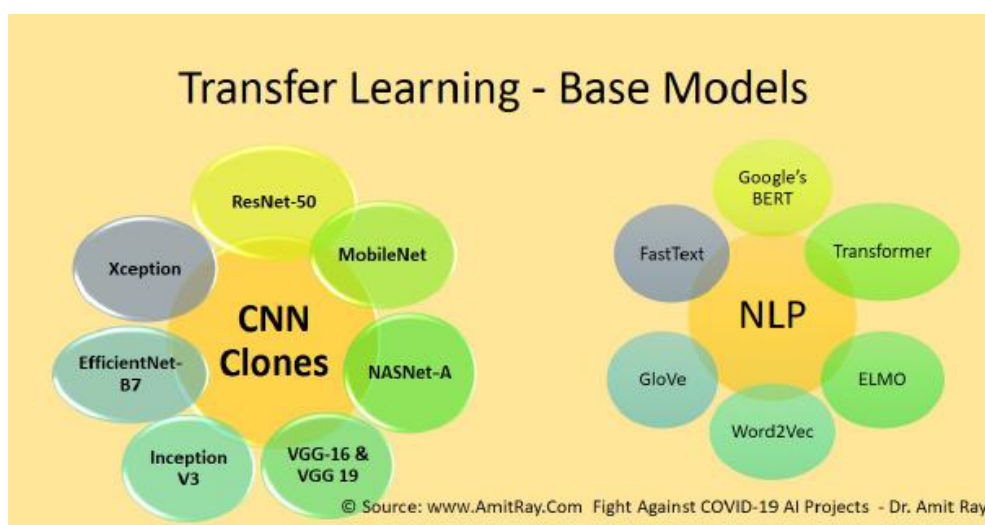


Figure: Transfer Learning - Base Models

In computer vision, ImageNet pre-trained CNN models are widely used as the standard base models for transfer learning. ImageNet is a large-scale database containing over 1.2 million labeled natural images across more than 1000 classes, with each class having at least 1000 images. CNN models trained on this dataset have demonstrated remarkable performance and generalization, making them ideal candidates for transfer learning in various vision tasks. Early architectures such as LeNet-5, AlexNet, VGGNet, GoogLeNet, and ResNet form the foundation of this approach. More advanced and widely adopted base models include ResNet-50, MobileNet, NASNet-A, VGG-16, VGG-19, Inception V3, EfficientNet-B7, and Xception.

For sequential transfer learning, especially in NLP tasks, popular base models include ULMFiT, Word2Vec, GloVe, FastText, and more recent transformer-based models like BERT, ELMO, and Transformer. These models have been trained on large text corpora and can capture rich semantic representations, making them highly effective for downstream tasks like sentiment analysis, question answering, and text classification.



Figure: Steps in Transfer Learning

Among the steps in transfer learning, Step 7 – Building the New Model – is the most critical. This step involves integrating the pre-trained base model into a new architecture tailored to the target task, often by adding new layers and fine-tuning them as needed. The success of transfer learning heavily relies on how well this step is executed, ensuring that the transferred knowledge is effectively adapted to the new domain.

5.4.2 Model Building Strategies in Transfer Learning

Feature Extraction: In this strategy, the model uses a pre-trained network as a feature extractor. The weights of the feature extraction layers (typically the initial layers) are frozen, meaning they are not updated during training. The output from these frozen layers is then passed to a new model or a traditional machine learning algorithm like random forest (RF), support vector machine (SVM), k-nearest neighbors (kNN), decision tree (DT), or naive Bayes (NB) for further processing. This approach is computationally efficient since feature extraction is done once and reused for similar tasks, saving on resources.

Fine-Tuning: Unlike feature extraction, fine-tuning involves unfreezing some of the layers of the pre-trained model and adjusting the weights for the specific target task. This allows the model to adapt more specifically to the new task, leveraging the general features learned from the original data. Fine-tuning speeds up convergence compared to training a model from scratch. While all the weights are typically updated during fine-tuning, only the final layers are adjusted in feature extraction, and only these weights are updated in the training process for the target task. Depending on the nature of the task, a combination of both approaches might be beneficial.

5.4.3 Steps for Fine-tuning the New Model

- Build the new network model on top of an already trained base model.

- Freeze the base network.
- Train the part you added.
- Unfreeze some layers in the base network.
- Jointly train both these layers and the part you added.

5.4.4 Basic Building Blocks of CNN Based Transfer Learning Models

In CNN-based transfer learning models, there are several key building blocks, which are crucial for feature extraction and classification tasks. The figure explains the basic building blocks of transfer learning.

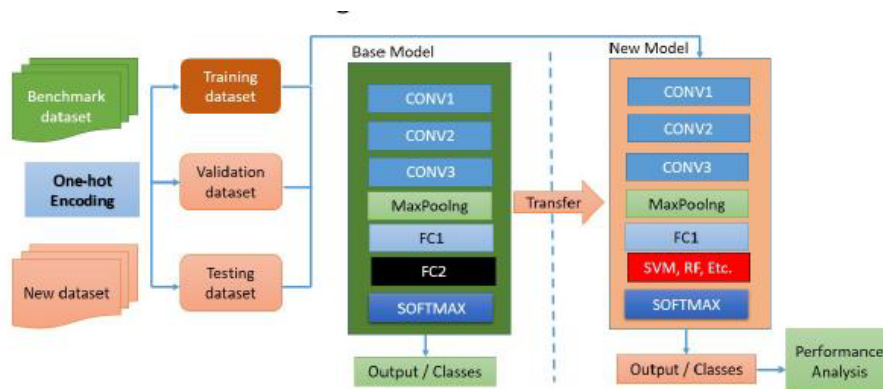


Figure: basic building blocks of transfer learning

5.4.4.1 Convolution, ReLU and Pooling Blocks

The first set of blocks in the CNN architecture typically includes convolutional layers (CONV), ReLU (Rectified Linear Unit) activation functions, and pooling layers. These blocks collectively perform feature extraction, which is similar to feature extraction in traditional machine learning models. Each convolutional layer uses a set of convolutional kernels to detect various patterns or features in the input data.

5.4.4.2 Pooling Layers

The pooling layer serves two primary purposes: it performs feature selection and reduces the dimensionality of the data while maintaining the most important features. Common pooling methods include maximum pooling, mean pooling, and random pooling. Maximum pooling selects the largest value in a specified region.

5.4.4.3 Fully connected layers

The fully connected (FC) layers typically sit near the end of a CNN architecture. These layers integrate the information from the earlier layers and are used for classification tasks. Output layer is also called the softmax layer, which maps the output of the fully connected layer to (0, 1) using the softmax function.

5.5 Object Detection

Object detection is a computer vision technique used to find and identify objects within an image or video. It has wide-ranging applications across many different industries. At its core, object detection answers two main questions about an image: "Which objects are present?" and "Where are these objects located?". This involves two simultaneous processes:

- Classification: Figuring out what the object is (e.g., a dog, a car, or a person).
- Localization: Pinpointing the exact position of the object in the image, usually by drawing a box around it.

5.5.1 Key Components of Object Detection

Object detection combines the following key ideas:

- Image Classification: This is the process of assigning a single label to an entire image. For example, labeling a picture as "a day at the park." It tells you what's in the image but not where.
- Object Localization: This step goes further by finding the position of an object and drawing a bounding box around it.
- Object Detection: This merges both classification and localization. It finds multiple different objects in an image, identifies what they are, and draws a box around each one to show its location.

5.5.2 Working of Object Detection

The process generally follows these steps:

- Input Image: It all starts with an image or a video that needs to be analyzed.
- Pre-processing: The image is prepared and formatted so the detection model can use it properly.
- Feature Extraction: A model, often a Convolutional Neural Network (CNN), scans the image to find interesting regions and pull out key features (like shapes, colors, and textures) that help identify objects.
- Classification: Based on the extracted features, each region is classified into a category, like "cat" or "dog". This is often done using a neural network that calculates the probability of an object being in that region.
- Localization: At the same time, the model calculates the coordinates for a bounding box to draw around each object it finds.
- Non-max Suppression: Sometimes the model detects the same object multiple times with overlapping boxes. This technique cleans up the extras by keeping only the box with the highest confidence score and removing the rest.

- Output: The final result is the original image with labeled bounding boxes showing all the detected objects.

5.5.3 Deep Learning Methods for Object Detection

Deep learning has revolutionized object detection. The methods are primarily divided into two main types:

- Two-Stage Detectors: These methods first propose potential regions where objects might be and then classify those regions in a second step. Examples include R-CNN, Fast R-CNN, and Faster R-CNN.
- Single-Stage Detectors: These models are more direct. They predict the bounding boxes and the object classes in a single pass, making them very fast. Popular examples are YOLO and SSD.

5.5.4 Two-Stage Detectors for Object Detection

These detectors are known for their high accuracy and work in two steps. The main types are:

- R-CNN (Regions with Convolutional Neural Networks): This technique first generates about 2000 "region proposals" from an image using a selective search algorithm. Each proposed region is then passed through a CNN model to classify the object inside it.
- Fast R-CNN: An improvement on R-CNN, this method processes the entire image with a CNN first to create a feature map. It then uses this map to extract features for each region, making the process much faster.
- Faster R-CNN: This version introduces a Region Proposal Network (RPN), which allows the model to learn where to look for objects directly from the feature maps. This makes the proposal step much more efficient than the selective search used in the original R-CNN.

5.5.5 Single-Stage Detectors for Object Detection

These detectors are built for speed and combine localization and classification into a single step. The two most popular models are:

- SSD (Single Shot MultiBox Detector): SSD examines the image once and predicts bounding boxes and classes using feature maps of different sizes. This allows it to detect objects of various scales efficiently in a single neural network pass.
- YOLO (You Only Look Once): YOLO also processes the entire image in one go. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell. This approach is extremely fast, making it ideal for real-time object detection.



5.5.6 Applications of Object Detection

Object detection is a powerful technology that drives innovation in many fields. Some key applications include:

- **Autonomous Vehicles:** Self-driving cars like those from Tesla and Waymo use object detection to "see" their surroundings. They can identify other cars, pedestrians, and road signs to navigate safely.
- **Security and Surveillance:** Smart cameras can automatically detect intruders or suspicious activities, and facial recognition systems can identify individuals for security purposes.
- **Healthcare:** In medical imaging, object detection helps doctors find tumors in X-rays and MRIs, leading to earlier and more accurate diagnoses.
- **Retail:** Stores like Amazon Go use object detection to create a cashier-less shopping experience by tracking which items you pick up. It's also used to monitor shelf stock in real-time.
- **Robotics:** Warehouse robots can identify and pick items for orders, while service robots can recognize and interact with people to provide assistance.

5.6 Natural Language Classification

5.7 Generative Adversarial Networks

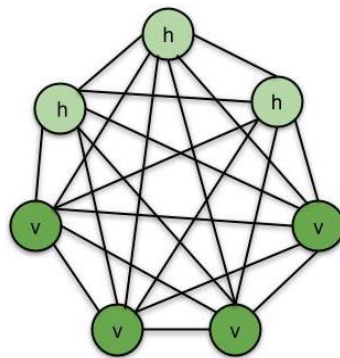
5.8 Boltzmann Machines

Boltzmann Machines (BM)s are a class of unsupervised deep learning models that differ significantly from other neural network architectures like Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Self-Organizing Maps (SOMs). One of the key differences is that BMs are undirected or bidirectional networks, meaning that each node is connected to every other node in the network, unlike other models which typically have directed connections. This bidirectional structure allows for the representation of complex dependencies between nodes, making BMs well-suited for modeling data distributions in an unsupervised manner. Boltzmann Machines are also stochastic or generative models, meaning they are not deterministic.

Instead, they explore various possible configurations or states, with the aim of learning the underlying distribution of the data.

In a Boltzmann Machine, the nodes are categorized into two primary types: visible and hidden nodes.

- The visible nodes represent observable or measurable features of the system, such as pixels in an image or features in a dataset.
- The hidden nodes represent unobservable factors or latent variables that influence the behavior of the system.



v - visible nodes, h - hidden nodes

Figure: Boltzmann Machine

5.8.1 Energy-Based Models

A Boltzmann Machine uses the Boltzmann distribution to sample from the system. This distribution describes how the probability of the system being in a particular state is related to the energy of that state. The formula for the Boltzmann distribution is given by:

$$P_i = \frac{e^{-\frac{E_i}{kT}}}{Z}$$

P_i - probability of system being in state i

E_i - Energy of system in state i

T - Temperature of the system

k - Boltzmann constant

Z - partition function, which normalizes the probabilities over all possible states of the system

Lower-energy states are more probable because the exponent in the Boltzmann distribution becomes less negative (closer to zero), making the exponential factor larger. During training, the model adjusts the weights of connections between nodes to minimize the energy of the

system, helping it reach a state of equilibrium where the model's learned distribution closely matches the distribution of the input data.

The training process of a Boltzmann Machine involves adjusting the weights of the connections between visible and hidden nodes in such a way that the network learns the statistical properties of the data. This process typically uses methods such as contrastive divergence to approximate the gradient of the log-likelihood function. Direct sampling from the Boltzmann distribution is computationally expensive, which is why approximation methods like contrastive divergence are essential in making the training process more efficient. The goal of training is to adjust the system's weights so that the Boltzmann Machine can generate new samples that are similar to the training data, making it a powerful generative model.

5.9 Restricted Boltzmann Machines (RBMs)

In a full Boltzmann Machine, every node is connected to every other node, resulting in a complex and computationally expensive structure as the number of nodes increases. Restricted Boltzmann Machines address this issue by simplifying the network architecture. In RBMs, connections are only allowed between visible and hidden nodes—visible-to-visible and hidden-to-hidden connections are not permitted. This structure makes learning and inference more efficient and manageable.

The energy function $E(v, h)$ captures the interaction between the visible units (v_i), the hidden units (h_j), the associated weights (w_{ij}), and their biases (a_i for visible units, b_j for hidden units). The probability of the network being in a certain state (v, h) is given by $P(v, h)$

$$E(v, h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{ij} h_j$$

a, b - biases in the system - constants
 v_i, h_j - visible node, hidden node
 $P(v, h)$ is the probability of being in a certain state, where $P(v, h) = e^{-E(v, h)} / Z$
 Z - sum of values for all possible states

In the context of a movie recommender system, RBMs can be effectively trained to discover latent features (such as genre, tone, or pacing preferences) from user ratings. Users express their preferences through binary values: 1 if they liked a movie, 0 if they disliked it, and missing values for unseen movies. During training, the RBM uses a learning algorithm called Contrastive Divergence (CD), which updates the weights to minimize the difference between the original data distribution and the model's reconstructed data. Through this process, the model learns to identify patterns in user behavior and can predict a user's potential preferences for unrated movies.

By uncovering these hidden patterns, RBMs serve as an effective collaborative filtering approach, especially useful when dealing with sparse datasets and binary preferences.

5.9.1 Contrastive Divergence

Contrastive Divergence (CD) is the primary training algorithm used in Restricted Boltzmann Machines to adjust the weights between visible and hidden nodes. The process begins with an initial set of random weights. Using these weights, the RBM calculates the states of the hidden nodes based on a given visible input (i.e., the observed data). These hidden node activations are then used to reconstruct the visible layer, essentially generating a synthetic version of the original input. Because the weights are initially random, this reconstruction typically differs from the original input. The hidden nodes are influenced by all visible nodes, and likewise, the reconstructed visible nodes are generated from all hidden nodes, forming a circular relationship.

This iterative back-and-forth process between visible and hidden layers is a simplified form of Gibbs Sampling. It samples from the conditional probabilities of each layer alternately, moving the system toward a stable state where the input and its reconstruction become increasingly similar.

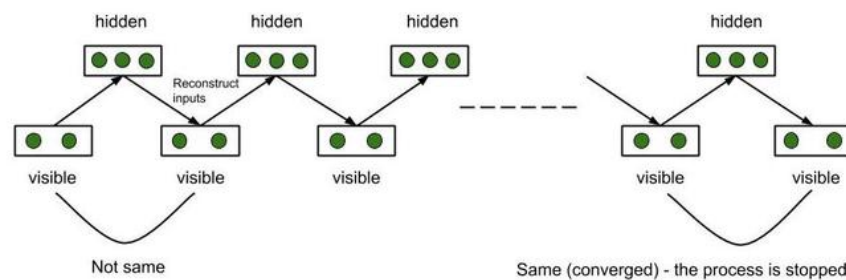


Figure: Gibbs Sampling

However, instead of running the Gibbs Sampling process until full convergence (which can be computationally expensive), CD takes only a few iterations—often just one step—making the training process much faster.

The adjustment of weights in CD is guided by the gradient of the log-likelihood of the data. This is expressed by the formula:

$$\frac{d}{dw_{ij}}(\log(P(v^0))) = \langle v_i^0 * h_j^0 \rangle - \langle v_i^\infty * h_j^\infty \rangle$$

v - visible state, h - hidden state
 $\langle v_i^0 * h_j^0 \rangle$ - initial state of the system
 $\langle v_i^\infty * h_j^\infty \rangle$ - final state of the system
 $P(v^0)$ - probability that the system is in state v^0
 w_{ij} - weights of the system

Importantly, Hinton's shortcut is one of the key figures behind RBMs—acknowledges that full convergence isn't necessary for effective learning. By limiting the number of Gibbs Sampling steps (e.g., CD-1), the system still approximates the necessary gradient descent on the energy landscape, reaching a sufficiently low-energy, stable state without exhaustive computation. This practical shortcut dramatically speeds up training while still producing high-quality results in applications such as recommender systems.

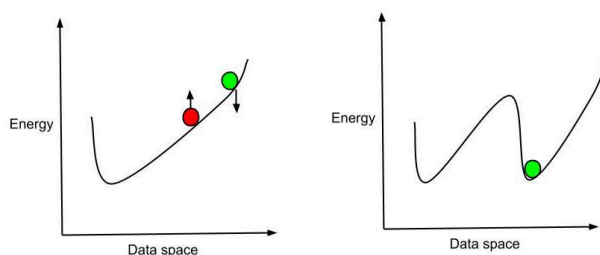


Figure: Hinton's Shortcut

5.9.2 Working of RBM – Illustrative Example

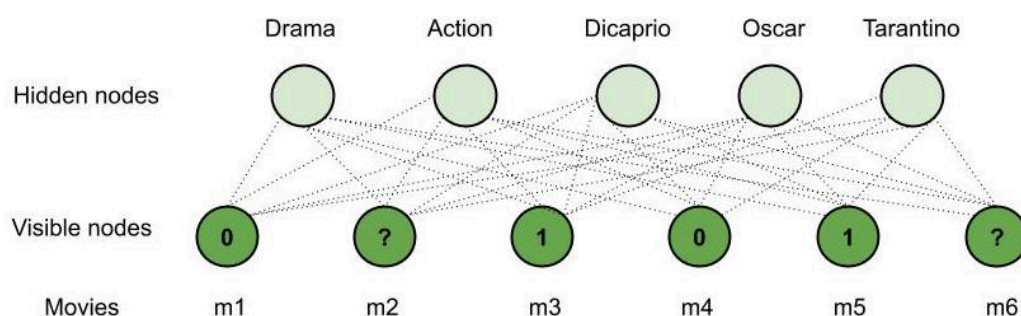



Figure: Working of RBM

Consider – Mary watches four movies out of the six available movies and rates four of them. Say, she watched m1, m3, m4 and m5 and likes m3, m5 (rated 1) and dislikes the other two, that is m1, m4 (rated 0) whereas the other two movies – m2, m6 are unrated. Now, using our RBM, we will recommend one of these movies for her to watch next. Say –

- m3, m5 are of the 'Drama' genre.
- m1, m4 are of the 'Action' genre.
- 'Dicaprio' played a role in m5.
- m3, m5 have won 'Oscar.'
- 'Tarantino' directed m4.
- m2 is of the 'Action' genre.
- m6 is of both the genres 'Action' and 'Drama', 'Dicaprio' acted in it and it has won an 'Oscar'.

We have the following observations

- 
- Mary likes m3, m5 and they are of the genre 'Drama', she probably likes 'Drama' movies.
 - Mary dislikes m1, m4 and they are of the action genre, she probably dislikes 'Action' movies.
 - Mary likes m3, m5 and they have won an 'Oscar', she probably likes an 'Oscar' movie.
 - Since 'Dicaprio' acted in m5 and Mary likes it, she will probably like a movie in which 'Dicaprio' acted.
 - Mary does not like m4 which is directed by Tarantino, she probably dislikes any movie directed by 'Tarantino'.

Therefore, based on the observations and the details of m2, m6; our RBM recommends m6 to Mary ('Drama', 'Dicaprio' and 'Oscar' matches both Mary's interests and m6). This is how an RBM works and hence is used in recommender systems. Thus, RBMs are used to build Recommender Systems.