MODULE-V
CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a class of deep neural networks designed to process and analyze structured grid-like data, such as images. They are especially effective for tasks involving spatial hierarchies, such as image recognition and classification, and have been adapted for various applications including natural language processing and time series analysis.

Key Components of CNNs

1. Convolutional Layers:
   - Function: Apply convolutional operations to input data using filters (or kernels). Each filter convolves over the input image to detect features such as edges, textures, or patterns.
   - Output: Generates feature maps (or activation maps) that highlight the presence of these features in different regions of the image.
2. Activation Function:
   - Function: Introduces non-linearity into the model, allowing it to learn complex patterns. Typically, a non-linear activation function like ReLU (Rectified Linear Unit) is used.
3. Pooling Layers:
   - Function: Reduce the spatial dimensions of feature maps, which decreases the computational load and helps control overfitting. Max pooling is a common operation, selecting the maximum value from a defined window.
   - Example: Max pooling with a 2x2 window reduces the size of the feature map by a factor of 2.
4. Fully Connected Layers:

    - Function: After convolutional and pooling layers, high-level features are flattened into a vector and passed through fully connected (dense) layers for final predictions or classifications.
    - Output: For classification tasks, the output is often processed through a softmax activation function to produce class probabilities.
5. Normalization Layers (Optional):
    - Function: Layers like Batch Normalization or Layer Normalization stabilize and accelerate training by normalizing the activations from previous layers.
6. Dropout Layers (Optional):
    - Function: A regularization technique that randomly sets a fraction of neurons to zero during training to prevent overfitting.

How CNNs Work

1. Feature Extraction:
    - Convolutional Layers: Extract features from the input image by applying various filters.
    - Activation Layers: Apply non-linear activation functions to capture complex features.
    - Pooling Layers: Reduce dimensionality of feature maps while retaining important information.
2. Classification:
    - Flattening: Convert feature maps into a single vector.
    - Fully Connected Layers: Analyze the features and make the final classification or regression predictions based on learned patterns.

CNN Architecture Example

1. Input Layer: An image of size 224x224x3 (height x width x channels).
2. Convolutional Layer: Apply 32 filters of size 3x3 to produce a feature map.

3. Activation Layer: Apply ReLU activation to the feature map.
4. Pooling Layer: Apply max pooling with a 2x2 window to reduce spatial dimensions.
5. Convolutional Layer: Apply 64 filters of size 3x3, followed by ReLU activation and max pooling.
6. Flattening: Convert the 2D feature maps into a 1D vector.
7. Fully Connected Layer: Connect the flattened vector to a dense layer with 128 units.
8. Output Layer: Use a softmax function to produce class probabilities.

Applications of CNNs

1. Image Classification: Identifying categories of objects in images (e.g., distinguishing between cats and dogs).
2. Object Detection: Locating and classifying objects within an image (e.g., detecting faces in photos).
3. Semantic Segmentation: Classifying each pixel in an image into predefined categories (e.g., segmenting different objects in an image).
4. Face Recognition: Identifying and verifying individuals based on facial features.
5. Medical Image Analysis: Detecting abnormalities in medical scans (e.g., identifying tumors in MRI images).

Advantages of CNNs

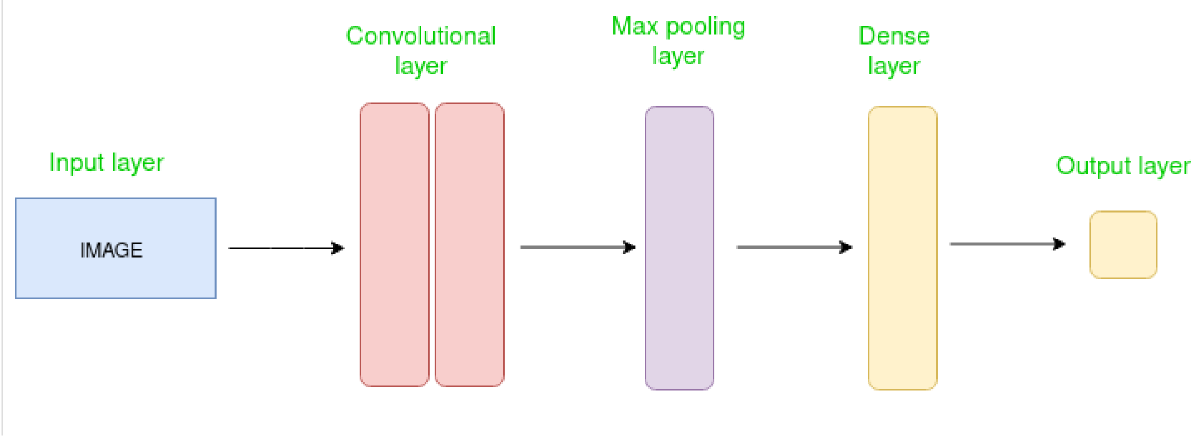
- Automatic Feature Extraction: Automatically learns and extracts relevant features from images, reducing the need for manual feature engineering.
- Spatial Hierarchies: Captures spatial hierarchies and relationships within images effectively.
- Translation Invariance: Achieves invariance to the location of features by using pooling layers, allowing recognition of objects in various positions.

Challenges

- Computational Complexity: Training CNNs requires significant computational resources, particularly for deep networks.
- Data Requirements: Requires large amounts of labeled data to achieve high performance.
- Overfitting: Deep CNNs can overfit to training data if not properly regularized or if the training data is insufficient.

CNN Architecture

A Convolutional Neural Network (CNN) typically consists of an input layer, several convolutional layers, pooling layers, and fully connected layers. This architecture allows CNNs to effectively process and analyze structured data, especially images.



Simple CNN architecture

How Convolutional Layers Work

Convolutional Layers are a key component of Convolutional Neural Networks (CNNs) and are tailored to handle grid-like data, such as images. These layers are responsible for extracting features by applying filters (or kernels) to the input data. Here’s a breakdown of their operation:

1. Concept of Convolution

Convolution in neural networks involves applying a filter to the input data to create a feature map. This process entails sliding the filter over the input and computing the dot product between the filter and the corresponding segment of the input data at each position.

2. Convolution Operation

1. Filter (Kernel):

- Definition: A filter is a small matrix of weights designed to detect specific features in the input data, such as edges, textures, or patterns.
- Size: Filters are generally small compared to the input image, commonly being 3x3 or 5x5 matrices.

2. Stride:

- Definition: The stride is the number of pixels by which the filter moves across the input image during the convolution process.
- Impact: A larger stride results in a smaller output feature map, while a smaller stride maintains a higher spatial resolution in the output.

3. Padding:

- Definition: Padding involves adding extra pixels around the border of the input image to control the dimensions of the output feature map.
- Types: Common padding methods include 'valid' (no padding) and 'same' (padding to ensure the output size matches the input size).

3. Convolution Process

1. Apply Filter:

- Operation: Place the filter on the input image, compute the dot product between the filter’s weights and the corresponding input pixels, and sum the results.

- Result: This produces a single value in the resulting feature map.

2. Slide Filter:

- Operation: Move the filter across the input image according to the stride value, performing the convolution operation at each position.
- Result: This creates a 2D matrix (feature map) that highlights where specific features are present in different regions of the input image.

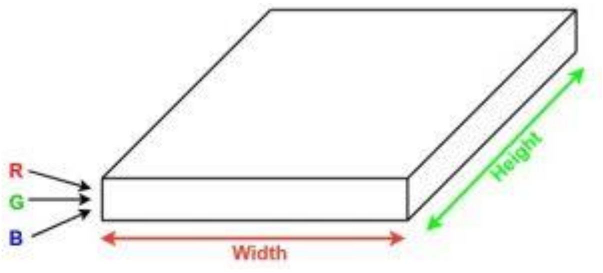
4. Output Feature Map

1. Feature Map:

- Definition: A 2D matrix representing the features detected by the filter throughout the input image.
- Purpose: Displays the locations and presence of specific features, such as edges or textures.

2. Multiple Filters:

- Operation: Multiple filters are applied within a convolutional layer, each detecting different features. Each filter generates its own feature map.
- Result: The feature maps from all filters are stacked together, producing a multi-channel output that captures a comprehensive set of features from the input data.



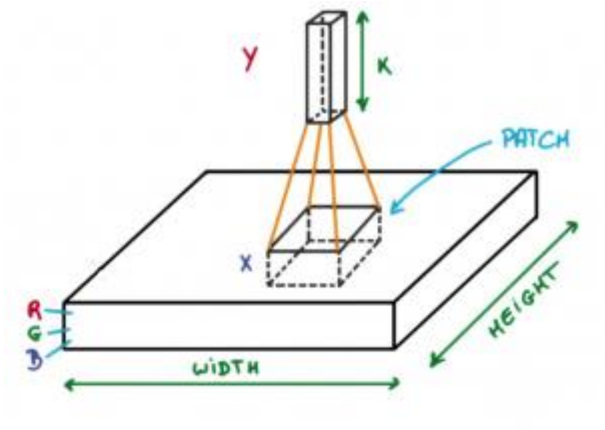


Image source: Deep Learning Udacity

Convolutional Layers: How They Work

Convolutional Layers are integral to Convolutional Neural Networks (CNNs) and are designed to handle grid-like data, such as images. They perform feature extraction by applying filters (or kernels) to the input data. Here's a step-by-step explanation of how convolutional layers function:

1. Concept of Convolution

Convolution in neural networks involves applying a filter to an input matrix to produce a feature map. This process includes sliding the filter over the input data and calculating the dot product between the filter and the section of the input it currently covers.

- Input Data: For example, an image with dimensions 5x5.
- Sliding Window: The filter moves across the input data. For a stride of 1 and no padding, the filter moves one pixel at a time.
- Dot Product: At each position, the filter's values are multiplied element-wise with the corresponding values in the input data. The results are summed to produce a single value in the output feature map.
- Feature Map: The resulting matrix highlights the features detected by the filter.

2. Steps in a Convolutional Layer

1. Apply Filter:

- Initialize: Start with an input matrix (e.g., an image) and a filter (kernel).
- Compute Convolution: Slide the filter across the input matrix. At each position, compute the dot product between the filter and the overlapping section of the input matrix.
- Output: Store each result in a new matrix, known as the feature map.

2. Activation Function:

- Apply Non-linearity: After computing the convolution, an activation function like ReLU (Rectified Linear Unit) is applied. This introduces non-linearity and allows the network to learn more complex patterns.
- ReLU Formula: ReLU(x) = max(0, x)

3. Pooling (Optional):

- Reduce Dimensions: Pooling layers, such as max pooling, are often used after convolution to reduce the spatial dimensions of the feature map while retaining the most significant features.

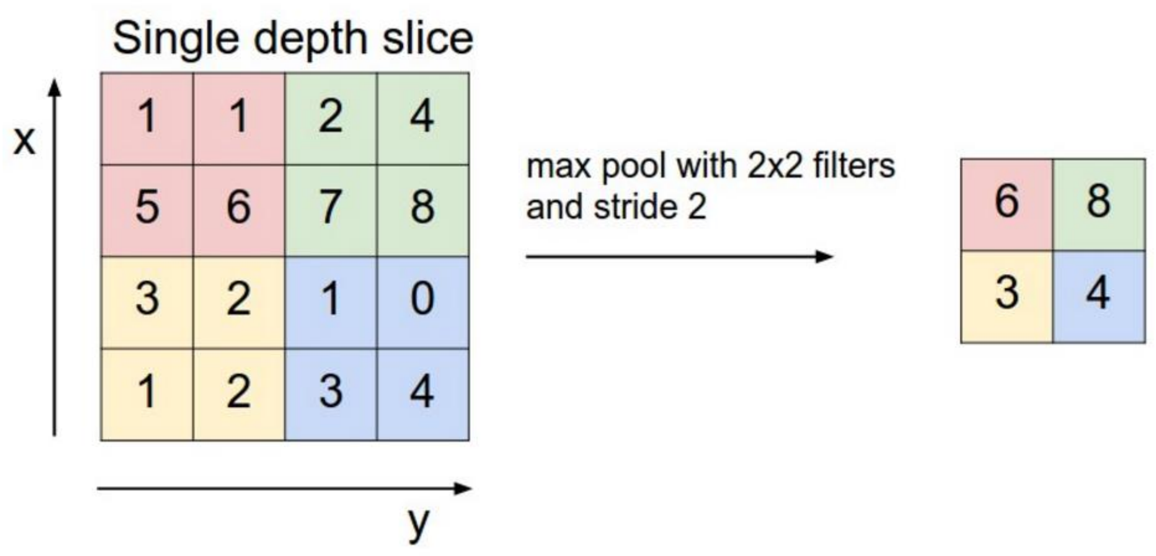


Image source: cs231n.stanford.edu

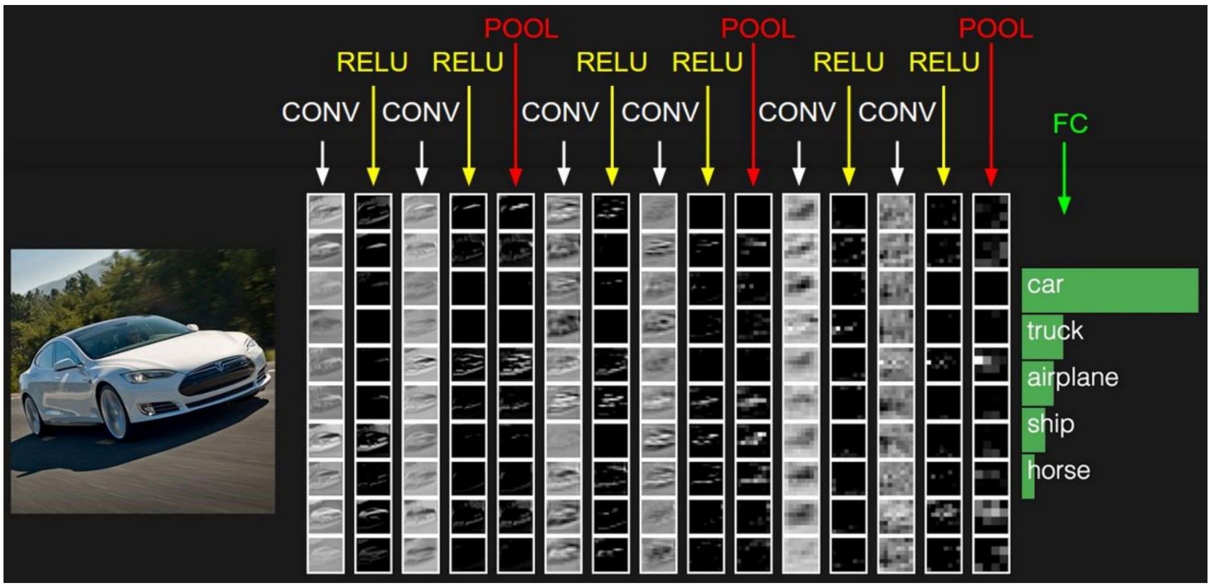


Image source: cs231n.stanford.edu

Convolutional Neural Network Architectures: AlexNet, VGG, GoogleNet, and ResNet

Several pivotal Convolutional Neural Network (CNN) architectures have shaped the field of computer vision, each contributing unique innovations. Here’s an overview of four influential architectures: AlexNet, VGG, GoogleNet, and ResNet.

1. AlexNet

Introduced: Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012

Achievements: Won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a significant margin.

Architecture:

- Layers: Comprises 5 convolutional layers followed by 3 fully connected layers.
- Activation Function: Utilizes ReLU (Rectified Linear Unit), which accelerated training compared to traditional sigmoid/tanh functions.
- Pooling: Applies max pooling after some convolutional layers to reduce spatial dimensions.
- Normalization: Incorporates Local Response Normalization (LRN) to normalize activations across feature maps.

Key Innovations:

- Deep Network: Demonstrated the effectiveness of deep networks for image classification.
- GPU Training: Leveraged GPUs to drastically reduce training time.
- Data Augmentation: Used data augmentation to enhance model generalization.

Impact: AlexNet’s success highlighted the potential of deep learning and GPUs, leading to widespread adoption and further research.

2. VGG (VGGNet)

Introduced: Visual Geometry Group, University of Oxford, 2014

Achievements: Known for its simplicity and depth.

Architecture:

- Layers: Includes 16 or 19 weight layers (both convolutional and fully connected).
- Convolutional Layers: Employs 3x3 convolutional filters with a stride of 1 and 2x2 max pooling with a stride of 2.
- Fully Connected Layers: Ends with 3 fully connected layers (in VGG-16).

Key Innovations:

- Small Filters: Utilizes small (3x3) filters, effectively capturing fine details and patterns.
- Uniform Architecture: Consistent use of small filters and pooling layers simplifies the network design.

Impact: VGG demonstrated that deeper networks with small convolutional filters could achieve high performance, influencing subsequent architectures.

3. GoogleNet (Inception V1)

Introduced: Google, 2014

Achievements: Won the ILSVRC 2014.

Architecture:

- Layers: Contains 22 layers with a complex design based on the Inception module.
- Inception Module: Employs parallel convolutions of different sizes (1x1, 3x3, 5x5) and pooling operations to capture multi-scale features.
- Global Average Pooling: Replaces fully connected layers with global average pooling to reduce overfitting and parameter count.

Key Innovations:

- Inception Modules: Combines convolutions of varying sizes and pooling in parallel to capture features at different scales.

- Efficient Computation: Reduces computational cost and number of parameters compared to earlier architectures.

Impact: GoogleNet’s multi-scale approach and efficiency set new standards for scalable and effective network designs.

4. ResNet (Residual Networks)

Introduced: Kaiming He et al., 2015

Achievements: Won the ILSVRC 2015.

Architecture:

- Layers: Ranges from 18 to 152 layers, with deeper versions using residual blocks.
- Residual Blocks: Incorporates shortcut connections that bypass one or more layers to facilitate gradient flow and prevent vanishing gradients.
- Bottleneck Architecture: Uses 1x1 convolutional layers to reduce and then restore dimensions around a 3x3 convolutional layer.

Key Innovations:

- Residual Learning: Introduces shortcut connections to improve training of very deep networks by addressing vanishing gradient issues.
- Training Deep Networks: Demonstrates effective training of very deep networks (e.g., 152 layers).

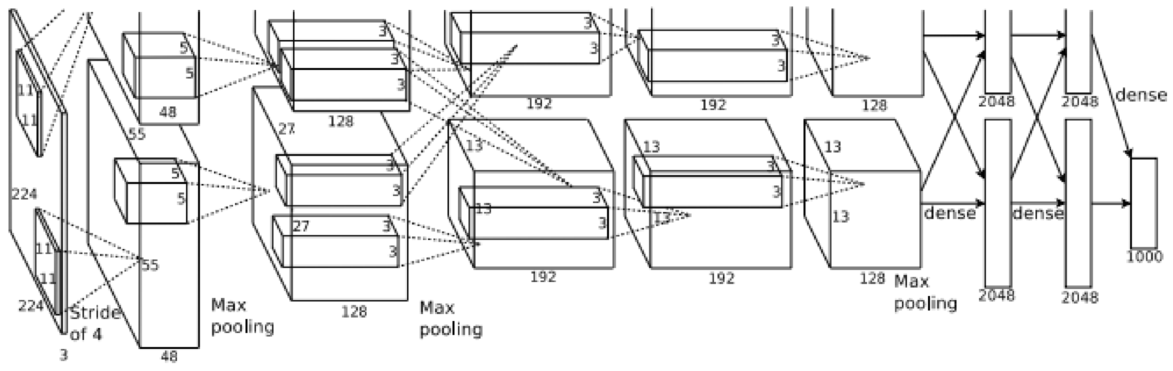
Impact: ResNet’s ability to train extremely deep networks improved performance across various computer vision tasks and demonstrated that very deep networks could be practical.

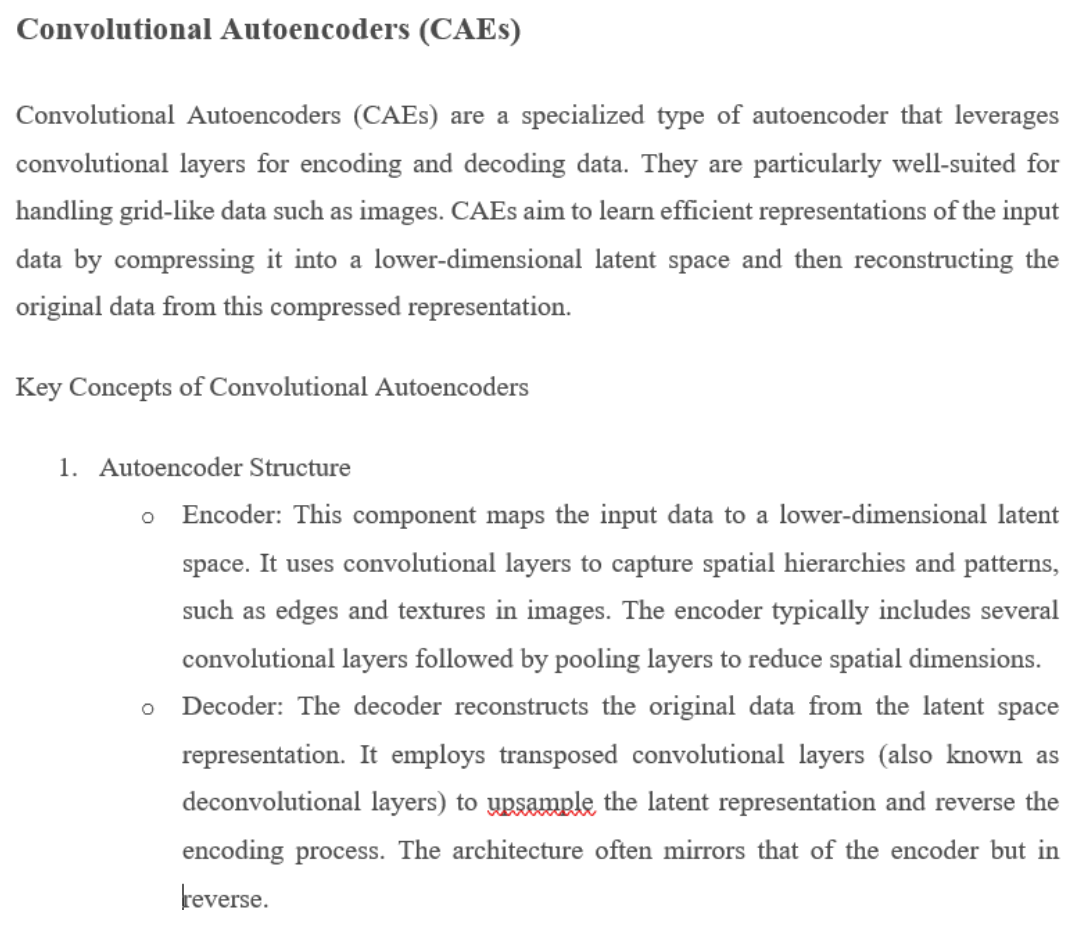
Summary of Architectures

- AlexNet: Pioneered deep learning for image classification with ReLU activation and GPU training.
- VGG: Focused on depth and small convolutional filters with a uniform architecture.
- GoogleNet: Introduced the Inception module for efficient multi-scale feature learning.

- ResNet: Innovated residual learning to enable training of very deep networks effectively.

Each of these architectures has made significant contributions to CNN development and has influenced subsequent advancements in deep learning.





Convolutional Autoencoders (CAEs)

Convolutional Autoencoders (CAEs) are a specialized type of autoencoder that leverages convolutional layers for encoding and decoding data. They are particularly well-suited for handling grid-like data such as images. CAEs aim to learn efficient representations of the input data by compressing it into a lower-dimensional latent space and then reconstructing the original data from this compressed representation.

Key Concepts of Convolutional Autoencoders

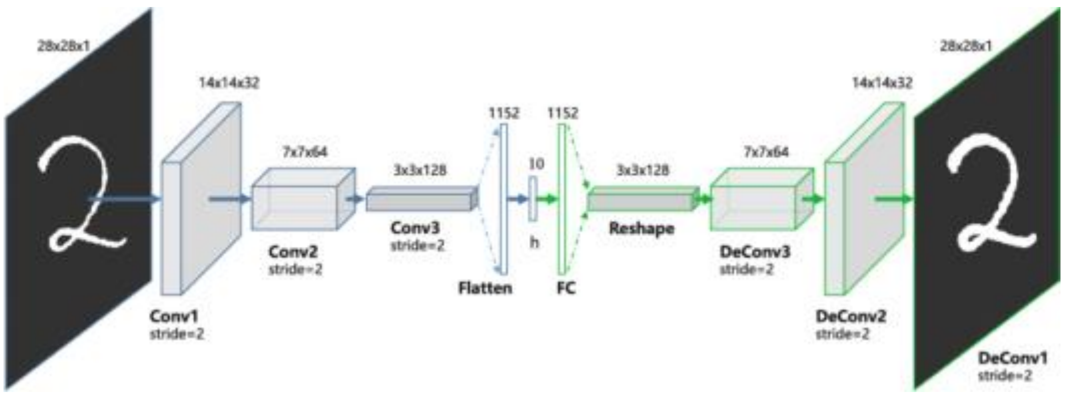
1. Autoencoder Structure
   - Encoder: This component maps the input data to a lower-dimensional latent space. It uses convolutional layers to capture spatial hierarchies and patterns, such as edges and textures in images. The encoder typically includes several convolutional layers followed by pooling layers to reduce spatial dimensions.
   - Decoder: The decoder reconstructs the original data from the latent space representation. It employs transposed convolutional layers (also known as deconvolutional layers) to upsample the latent representation and reverse the encoding process. The architecture often mirrors that of the encoder but in reverse.

1. Convolutional Layers
    - Encoder: Utilizes convolutional layers to extract features from the input data, capturing spatial hierarchies and patterns. Pooling layers are used to downsample feature maps and reduce spatial dimensions.
    - Decoder: Applies transposed convolutional layers to upsample the compressed latent space representation, aiming to reconstruct the original input data.

How Convolutional Autoencoders Work

1. Encoding
    - Process: The input data (e.g., an image) is passed through a sequence of convolutional layers. These layers extract features, while pooling layers downsample the feature maps.
    - Latent Space: The final layer in the encoder produces a compressed, lower-dimensional representation of the input data, known as the latent space.
2. Decoding
    - Process: The latent space representation is fed through a series of transposed convolutional layers. These layers upsample the data to reconstruct the original input.
    - Reconstruction: The network aims to recreate the original input as closely as possible. The quality of the reconstruction is evaluated by comparing it to the original input.
3. Training
    - Objective: The network is trained to minimize the reconstruction loss, such as Mean Squared Error (MSE) or Binary Cross-Entropy, between the original input and the reconstructed output.
    - Learning: During training, the encoder learns to compress the data effectively, while the decoder learns to reconstruct it with high accuracy.

Convolutional Autoencoders are valuable for tasks such as image denoising, dimensionality reduction, and feature extraction, as they leverage the power of convolutional layers to capture and reconstruct complex spatial patterns.



Content-Based Image Retrieval (CBIR)

Content-Based Image Retrieval (CBIR) is a technique used to search and retrieve images from a database based on their visual content rather than relying on metadata or textual descriptions. CBIR systems analyze the actual content of images—such as colors, textures, shapes, and patterns—to find similar or relevant images.

Key Concepts of CBIR

1. Feature Extraction
   - Objective: Extract meaningful features from images for comparison purposes.
   - Types of Features:
     - Color: Color histograms, color moments, color distribution.
     - Texture: Statistical measures, Gabor filters, Local Binary Patterns (LBP).
     - Shape: Edges, contours, shape descriptors like Hough transforms or contour-based features.
     - Spatial Information: Relative positions of features within the image.
2. Feature Representation
   - Objective: Represent the extracted features in a format that enables efficient comparison.
   - Common Methods:
     - Feature Vectors: Numerical representations of extracted features.
     - Histograms: Distribution of feature values, such as color histograms.
3. Similarity Measurement

   - Objective: Assess the similarity between images based on their feature representations.
   - Methods:
     - Distance Metrics: Euclidean distance, Manhattan distance, cosine similarity.
     - Similarity Measures: Correlation, Chi-square distance.
4. Indexing and Search
   - Objective: Efficiently store and retrieve images based on their feature representations.
   - Techniques:
     - Spatial Indexing: KD-trees, R-trees, or other spatial indexing structures to organize feature vectors.
     - Hashing: Techniques like Locality-Sensitive Hashing (LSH) for fast similarity search.
5. Relevance Feedback
   - Objective: Enhance search results through user feedback.
   - Process:
     - User Interaction: Users provide feedback on the relevance of retrieved images.
     - Model Update: Adjust feature representation or similarity measures based on feedback to refine search results.

Example Workflow of CBIR

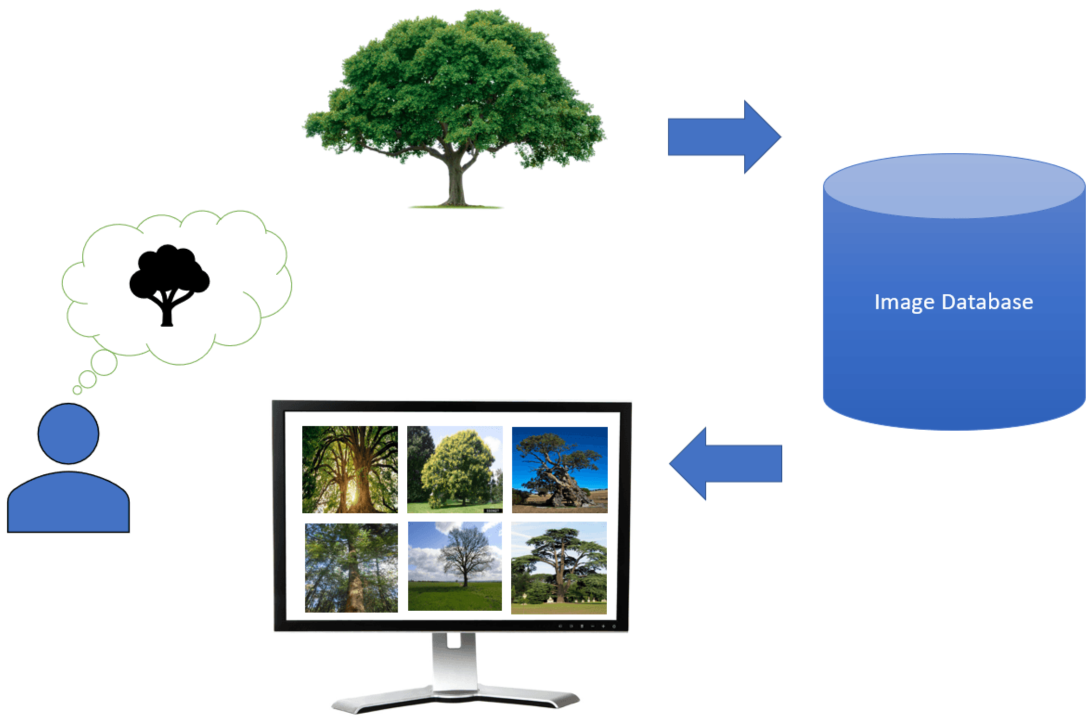
1. Feature Extraction:
   - Extract features from both the query image and all images in the database.
2. Feature Representation:
   - Represent these features using feature vectors or histograms.
3. Similarity Measurement:
   - Compute the similarity between the query image and database images using chosen distance metrics or similarity measures.
4. Retrieve Results:
   - Rank and retrieve images from the database based on their similarity to the query image.
5. Relevance Feedback (Optional):
   - Use user feedback to improve search results.

Applications of CBIR

1. Image Search Engines:
    - Enable users to search for images based on visual content, such as Google Images or Bing Image Search.
2. Medical Image Analysis:
    - Retrieve similar medical images for diagnostic purposes or comparison.
3. Security and Surveillance:
    - Match and retrieve images or video frames for security and surveillance applications.
4. Digital Asset Management:
    - Organize and retrieve digital assets like photographs, art, and media based on content.
5. Fashion and Retail:
    - Allow users to search for clothing items or accessories based on visual similarities.

Key Challenges

1. Feature Representation:
    - Selecting and designing effective feature extraction methods for diverse types of images.
2. Scalability:
    - Efficiently handling large databases in terms of both storage and retrieval speed.
3. Semantic Gap:
    - Bridging the gap between low-level visual features and high-level semantic concepts.
4. Computational Complexity:
    - Ensuring efficient computation for feature extraction, similarity measurement, and retrieval.
5. Relevance Feedback:
    - Effectively incorporating user feedback to enhance search results.



Object Detection

Object detection using deep learning focuses on identifying and locating objects within images or videos. Unlike image classification, which only identifies the presence of an object, object detection provides both the location and class of objects through bounding boxes. Deep learning has greatly enhanced the accuracy and efficiency of object detection. Here's an overview of how deep learning is applied to object detection:

Key Components of Deep Learning-Based Object Detection

1. Feature Extraction
   - Objective: Extract relevant features from images that are essential for detecting objects.
   - Method: Convolutional Neural Networks (CNNs) are commonly used to capture various details and patterns from different levels of the image.
2. Bounding Box Prediction
   - Objective: Predict the location and dimensions of bounding boxes around detected objects.
   - Method: Networks estimate the coordinates of bounding boxes along with class labels.
3. Classification

    - Objective: Classify objects within the predicted bounding boxes.
    - Method: After detecting regions of interest, a classifier determines the category of each object within the bounding boxes.
4. Post-Processing
    - Objective: Refine detection results and remove duplicates.
    - Method: Techniques like Non-Maximum Suppression (NMS) are used to eliminate overlapping bounding boxes and retain the most accurate ones.

Popular Deep Learning Architectures for Object Detection

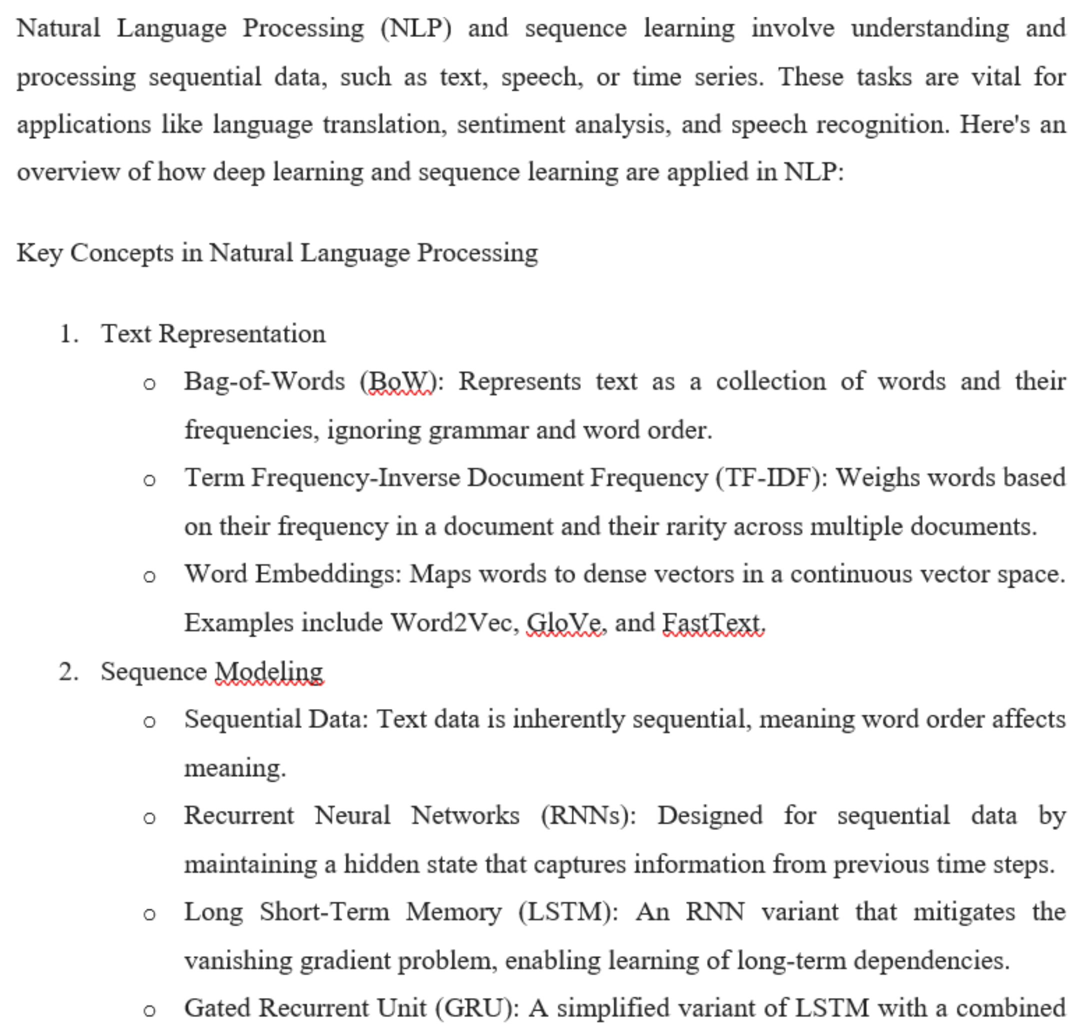
1. R-CNN (Regions with CNN Features)
    - Introduction: Proposed by Ross Girshick et al. in 2014.
    - Approach:
        - Region Proposal: Uses selective search to generate region proposals.
        - Feature Extraction: Applies CNNs to extract features from each region proposal.
        - Classification and Bounding Box Regression: Utilizes SVMs for classification and linear regression for bounding box refinement.
    - Challenges: Computationally intensive due to separate processing of region proposals.
2. Fast R-CNN
    - Introduction: Proposed by Ross Girshick in 2015.
    - Approach:
        - Region of Interest (RoI) Pooling: Extracts features for each proposal from a shared feature map.
        - End-to-End Training: Trains the entire network end-to-end, enhancing speed and accuracy.
    - Advantages: Faster and more accurate than R-CNN by processing the entire image once and using RoI pooling.
3. Faster R-CNN
    - Introduction: Proposed by Shaoqing Ren et al. in 2015.
    - Approach:

        - Region Proposal Network (RPN): Integrates a region proposal network that shares convolutional features with the detection network, removing the need for selective search.
        - End-to-End Training: Trains both RPN and detection networks together.
    - Advantages: Much faster than Fast R-CNN due to the integrated RPN.
4. YOLO (You Only Look Once)
    - Introduction: Proposed by Joseph Redmon et al. in 2016.
    - Approach:
        - Single Pass Detection: Divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell in one pass.
        - Real-Time Detection: Designed for high-speed, real-time object detection.
    - Advantages: Fast and capable of processing video streams in real-time.
5. SSD (Single Shot MultiBox Detector)
    - Introduction: Proposed by Wei Liu et al. in 2016.
    - Approach:
        - Multi-Scale Feature Maps: Uses multiple feature maps at different scales to detect objects of varying sizes.
        - Single Pass Detection: Performs detection in one pass, similar to YOLO.
    - Advantages: Balances speed and accuracy, suitable for real-time applications.
6. RetinaNet
    - Introduction: Proposed by Tsung-Yi Lin et al. in 2017.
    - Approach:
        - Focal Loss: Introduces focal loss to address class imbalance between foreground and background.
        - Single Shot Detection: Detects objects in one pass.
    - Advantages: Effective at detecting small and difficult objects with improved accuracy.
7. EfficientDet
    - Introduction: Proposed by Mingxing Tan et al. in 2020.
    - Approach:

- Compound Scaling: Uses compound scaling to balance model accuracy, size, and speed.
- Efficient Backbone: Employs a highly efficient backbone for feature extraction.

- Advantages: Achieves state-of-the-art performance with high efficiency.

Natural Language Processing and Sequence Learning



Natural Language Processing (NLP) and sequence learning involve understanding and processing sequential data, such as text, speech, or time series. These tasks are vital for applications like language translation, sentiment analysis, and speech recognition. Here's an overview of how deep learning and sequence learning are applied in NLP:

Key Concepts in Natural Language Processing

1. Text Representation
   - Bag-of-Words (BoW): Represents text as a collection of words and their frequencies, ignoring grammar and word order.
   - Term Frequency-Inverse Document Frequency (TF-IDF): Weighs words based on their frequency in a document and their rarity across multiple documents.
   - Word Embeddings: Maps words to dense vectors in a continuous vector space. Examples include Word2Vec, GloVe, and FastText.
2. Sequence Modeling
   - Sequential Data: Text data is inherently sequential, meaning word order affects meaning.
   - Recurrent Neural Networks (RNNs): Designed for sequential data by maintaining a hidden state that captures information from previous time steps.
   - Long Short-Term Memory (LSTM): An RNN variant that mitigates the vanishing gradient problem, enabling learning of long-term dependencies.
   - Gated Recurrent Unit (GRU): A simplified variant of LSTM with a combined

mechanisms

1. Attention Mechanisms
   - Self-Attention: Allows the model to focus on different parts of the input sequence when making predictions.
   - Transformer Architecture: Utilizes self-attention and feed-forward networks to process sequences in parallel, overcoming the limitations of RNNs and LSTMs.

Key Models and Architectures

1. Recurrent Neural Networks (RNNs)
   - Usage: Effective for tasks where sequential information is crucial, such as language modeling and time series forecasting.
   - Challenges: Struggle with long-term dependencies due to vanishing or exploding gradients.
2. Long Short-Term Memory (LSTM)
   - Usage: Handles long-term dependencies better than standard RNNs by using gating mechanisms to control information flow.
   - Components: Includes input gate, forget gate, and output gate.
3. Gated Recurrent Unit (GRU)
   - Usage: A variant of LSTM that merges input and forget gates into a single update gate, simplifying the architecture.
   - Advantages: Often performs similarly to LSTM with fewer parameters.
4. Transformer
   - Introduced: In "Attention is All You Need" by Vaswani et al. (2017).
   - Architecture: Consists of an encoder and decoder, both utilizing self-attention mechanisms and feed-forward layers.
   - Self-Attention: Allows the model to weigh the importance of different words in a sequence.
   - Advantages: Handles long-range dependencies efficiently and processes sequences in parallel.
5. BERT (Bidirectional Encoder Representations from Transformers)
   - Introduced: By Google AI in 2018.
   - Architecture: A transformer-based model trained bidirectionally on large text corpora.
   - Usage: Effective for various NLP tasks through pre-training on large datasets and fine-tuning for specific tasks.
6. GPT (Generative Pre-trained Transformer)
   - Introduced: By OpenAI.
   - Architecture: A series of transformer-based models trained on extensive text data.
   - Usage: Generates human-like text, performs text completion, and more.

7. T5 (Text-To-Text Transfer Transformer)
   - Introduced: By Google Research in 2020.
   - Architecture: Treats all NLP tasks as text-to-text problems, enabling a unified model for diverse tasks.
   - Usage: Versatile model for translation, summarization, question answering, and more.

Applications of Sequence Learning in NLP

1. Language Translation
   - Objective: Translate text from one language to another.
   - Approach: Models like Seq2Seq with attention and transformers (e.g., Google Translate, BERT) are used.
2. Sentiment Analysis
   - Objective: Determine the sentiment expressed in text (e.g., positive, negative, neutral).
   - Approach: Models like LSTM, GRU, and BERT classify sentiment based on textual input.
3. Named Entity Recognition (NER)
   - Objective: Identify and classify entities (e.g., names, dates, locations) in text.
   - Approach: Models like CRF (Conditional Random Fields) and BERT-based NER systems are used.
4. Text Summarization
   - Objective: Generate concise summaries of longer documents.
   - Approach: Abstractive summarization uses models like transformers, while extractive summarization identifies key sentences or phrases.
5. Question Answering
   - Objective: Provide accurate answers to questions based on context or documents.
   - Approach: Models like BERT and GPT-3 understand questions and extract or generate answers from the context.
6. Speech Recognition
   - Objective: Convert spoken language into written