Contents

[1. What is a Convolutional Neural Network (CNN)? 2](#_Toc200349846)

[2. Explain the architecture of a typical CNN. 3](#_Toc200349847)

[3. What are the key components of a CNN? 4](#_Toc200349848)

[4. What is the purpose of convolutional layers in CNNs? 5](#_Toc200349849)

[5. How do pooling layers contribute to CNNs? 5](#_Toc200349850)

[6. What is the difference between convolution and pooling operations in CNNs? 6](#_Toc200349851)

[7. What are activation functions, and why are they used in CNNs? 7](#_Toc200349852)

[8. Describe the concept of padding in CNNs. Why is it used? 7](#_Toc200349853)

[9. What is the role of filters (kernels) in convolutional layers? 8](#_Toc200349854)

[10. How does the stride affect the output size of convolutional layers? 9](#_Toc200349855)

[11. Explain the term "feature map" in the context of CNNs. 10](#_Toc200349856)

[12. What is the purpose of the fully connected (dense) layers in CNNs? 10](#_Toc200349857)

[13. How is weight sharing beneficial in CNNs? 11](#_Toc200349858)

[Summary Table: 11](#_Toc200349859)

[14. Describe the process of backpropagation in CNNs. 12](#_Toc200349860)

[15. What are some popular activation functions used in CNNs, and why? 12](#_Toc200349861)

[Summary : 12](#_Toc200349862)

[16. What is batch normalization, and how does it help in training CNNs? 13](#_Toc200349863)

[17. How do dropout layers prevent overfitting in CNNs? 13](#_Toc200349864)

[18. Explain the terms "data augmentation" and its significance in CNNs. 14](#_Toc200349865)

[19. What is transfer learning, and how can it be applied in CNNs? 15](#_Toc200349866)

[20. How would you evaluate the performance of a CNN model? 15](#_Toc200349867)

[Common Metrics: 15](#_Toc200349868)

[21. What loss functions are commonly used in CNNs for classification tasks? 16](#_Toc200349869)

[22. How does the learning rate impact the training of CNN models? 16](#_Toc200349870)

[23. What are some challenges specific to training CNN models? 16](#_Toc200349871)

[24. How can you improve the training speed of CNNs? 17](#_Toc200349872)

[25. What are some applications of CNNs in computer vision? 17](#_Toc200349873)

[26. What are the limitations of CNNs compared to other neural network architectures? 18](#_Toc200349874)

[27. Explain the concept of receptive field in CNNs. 19](#_Toc200349875)

[28. How does the concept of stride affect the receptive field size? 19](#_Toc200349876)

## What is a Convolutional Neural Network (CNN)?

A Convolutional Neural Network (CNN) is a type of deep learning architecture specifically designed for analyzing visual imagery. Here's a breakdown of its core concept:

**Understanding Images:**

* Unlike traditional neural networks that process data in a flat format, CNNs leverage the inherent structure of images. Images are essentially grids of pixels, and CNNs exploit this by applying filters that learn features directly from the pixel data.

**Key Components:**

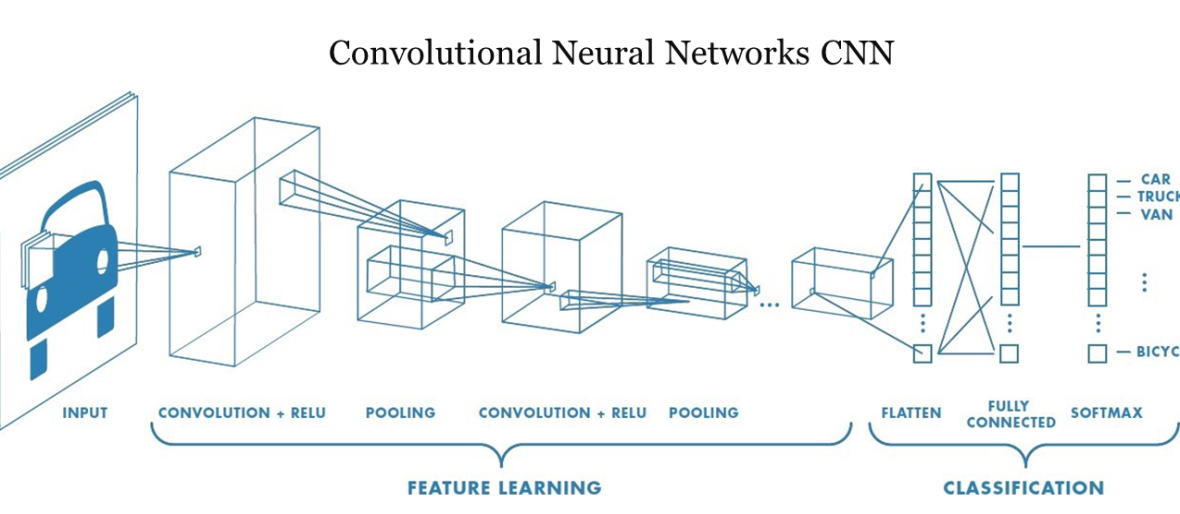
* Convolutional layers: These are the heart of a CNN. They apply filters that slide across the image, extracting features like edges, shapes, and textures. Each filter learns to detect a specific pattern in the image.
* Pooling layers: These layers downsample the output of convolutional layers, reducing the number of parameters and computational cost. They also introduce a level of invariance to small shifts in the image. Common pooling techniques include max pooling (taking the maximum value) and average pooling (taking the average value) within a specific region.
* Activation layers: These layers introduce non-linearity into the network, allowing it to learn complex relationships between features. A popular activation function used in CNNs is ReLU (Rectified Linear Unit).
* Fully connected layers: In the later stages of the network, fully connected layers similar to traditional neural networks are used to combine the extracted features and make predictions (e.g., image classification).

**Advantages of CNNs:**

* **Feature Extraction:** CNNs excel at automatically learning relevant features from images, eliminating the need for manual feature engineering, a common challenge in traditional computer vision approaches.
* **Spatial Relationships:** They can capture the spatial relationships between pixels in an image, which is crucial for tasks like object recognition and image segmentation.
* **Parameter Efficiency:** By using shared weights in convolutional filters, CNNs can achieve good performance with fewer parameters compared to fully connected networks processing flattened images.

**In essence, CNNs are powerful tools for analyzing visual data by learning features directly from the image structure, making them a cornerstone of many computer vision applications.**

## Explain the architecture of a typical CNN.



The architecture of a typical Convolutional Neural Network (CNN) is composed of a series of layers, each serving a specific purpose to process and analyze the input data. Here is a step-by-step explanation of the layers and components commonly found in a CNN:

1. **Input Layer:**

The input layer receives the raw pixel values of the image. For instance, a colored image with a size of 32x32 pixels and three color channels (RGB) would have an input shape of (32, 32, 3).

2. **Convolutional Layers:**

These layers apply convolutional filters to the input image or previous layer's output to extract features such as edges, textures, and patterns.

Each convolutional layer uses a set of learnable filters (kernels) that slide over the input, computing dot products and producing feature maps.

Example: A convolutional layer with 32 filters of size 3x3 applied to an input image would output 32 feature maps.

3. **Activation Functions:**

Non-linear activation functions are applied to the output of convolutional layers to introduce non-linearity and allow the network to learn complex patterns.

The most common activation function used in CNNs is ReLU (Rectified Linear Unit), which replaces all negative values in the feature map with zero.

4. **Pooling Layers:**

Pooling layers, such as max pooling or average pooling, downsample the feature maps, reducing their spatial dimensions (width and height) while retaining important features.

This helps to reduce the computational load and the number of parameters, making the network more efficient.

Example: A max pooling layer with a pool size of 2x2 applied to a feature map reduces its dimensions by half.

5. **Dropout Layers (Optional):**

Dropout layers randomly deactivate a fraction of neurons during training, which helps prevent overfitting by ensuring that the network does not rely too heavily on specific neurons.

Example: A dropout layer with a rate of 0.5 will randomly deactivate 50% of the neurons during each training step.

6. **Fully Connected (Dense) Layers:**

Fully connected layers connect every neuron in one layer to every neuron in the next layer. They are typically used at the end of the network.

These layers take the high-level features extracted by the convolutional and pooling layers and use them to make predictions.

Example: A fully connected layer with 128 neurons.

7. **Output Layer:**

The final layer of the network, which produces the output predictions. In a classification task, this is typically a softmax layer that outputs probabilities for each class.

Example: For a 10-class classification problem, the output layer would have 10 neurons, each representing the probability of one class.

## What are the key components of a CNN?

Covered above

## What is the purpose of convolutional layers in CNNs?

Covered above

## How do pooling layers contribute to CNNs?

Pooling layers play a crucial role in Convolutional Neural Networks (CNNs) by offering several key benefits:

**Dimensionality Reduction:**

* One primary function of pooling layers is to reduce the dimensionality of the data coming from convolutional layers.
* Convolutional layers generate feature maps, which can be quite large depending on the number of filters used.
* Pooling layers shrink the size of these feature maps by summarizing the information within a predefined window. This reduces the number of parameters in the network, making it more computationally efficient and faster to train.

**Shift Invariance:**

* Pooling layers introduce a degree of invariance to small shifts in the position of features within the image.
* By taking the maximum or average value within a local region, pooling layers are less sensitive to slight variations in an object's placement in the image.
* This helps the network focus on the presence of a feature rather than its exact location, improving robustness to noise and slight misalignments.

**Abstraction of Features:**

* Pooling layers provide a summarized representation of the features learned by convolutional layers.
* The max pooling operation, for example, emphasizes the most prominent features within a local region. This can help the network focus on the dominant aspects of the features and reduce sensitivity to irrelevant details.

**Computational Efficiency:**

* As mentioned earlier, dimensionality reduction by pooling layers leads to fewer parameters in the network. This translates to lower memory requirements and faster computation during training and inference.

Here's a table summarizing the key points:

|  |  |
| --- | --- |
| **Benefit** | **Description** |
| Dimensionality Reduction | Shrinks feature map size, reducing parameters and computation cost. |
| Shift Invariance | Makes the network less sensitive to small shifts in features. |
| Abstraction of Features | Provides a summarized representation of learned features. |
| Computational Efficiency | Reduces memory usage and speeds up training/inference. |

**Choosing the Right Pooling Technique:**

The specific pooling technique used (max pooling, average pooling) can influence the features learned by the network. Here's a brief comparison:

* **Max pooling:** Emphasizes the most prominent features, useful for detecting dominant characteristics.
* **Average pooling:** Captures the average information within a region, potentially preserving more spatial information.

The best choice often depends on the specific task and the type of features you want the network to focus on.

**In conclusion, pooling layers are essential components in CNN architectures. They contribute to efficiency, robustness, and feature abstraction, playing a vital role in the overall performance of the network.**

## What is the difference between convolution and pooling operations in CNNs?

* **Convolution Operation:**
  + The convolution operation is used to extract features from the input image. It applies a set of learnable filters (kernels) to the input data, producing feature maps that highlight specific patterns such as edges, textures, and shapes.
* **Pooling Operation:**
  + The pooling operation is used to reduce the spatial dimensions of the feature maps generated by the convolutional layers. This helps in reducing the computational complexity and the number of parameters, and also provides a form of spatial invariance.

**In essence, convolution focuses on identifying and extracting features, while pooling focuses on summarizing and reducing the data while introducing some level of invariance to small shifts.** These operations work together in CNNs to efficiently learn hierarchical representations of the input data, ultimately leading to accurate image recognition or analysis.

## What are activation functions, and why are they used in CNNs?

Covered in ANN

## Describe the concept of padding in CNNs. Why is it used?

Padding in Convolutional Neural Networks (CNNs) refers to the process of adding extra pixels (usually zeros) to the border of an input image or feature map before applying a convolution operation. The purpose of padding is to control the spatial dimensions (width and height) of the output feature maps and to preserve information at the borders of the input.

**Why Padding is Used:**

1. **Preserve Spatial Dimensions:**
   * Padding allows the spatial dimensions of the output feature map to match the input dimensions, which can be important for certain tasks, such as image segmentation where spatial resolution must be maintained.
2. **Prevent Information Loss at Borders:**
   * Without padding, the convolution operation reduces the size of the feature map with each layer, causing information at the borders to be lost progressively. Padding helps in retaining this information.
3. **Control Output Size:**
   * By adding padding, you can control the output size of the convolution operation, allowing more flexibility in designing the network architecture.
4. **Enable Deeper Networks:**
   * With padding, deeper networks can be constructed without excessive reduction in feature map size. This enables more complex hierarchical feature extraction.
5. **Maintain Centered Features:**
   * Padding helps maintain the central position of features within the input image. This ensures that features detected at the borders are treated similarly to those detected in the center

## What is the role of filters (kernels) in convolutional layers?

Filters (also known as kernels) in convolutional layers play a fundamental role in Convolutional Neural Networks (CNNs). They are responsible for detecting various features in the input data, such as edges, textures, shapes, and more complex patterns. Here's a detailed look at the role and importance of filters in convolutional layers:

**Role of Filters in Convolutional Layers:**

1. **Feature Extraction:**
   * Filters are used to extract features from the input image or feature map. Each filter is designed to detect a specific type of feature, such as horizontal or vertical edges, textures, or other patterns.
   * When a filter convolves over the input, it produces a feature map that highlights the presence of the particular feature detected by that filter.
2. **Learnable Parameters:**
   * Filters contain learnable parameters (weights) that are optimized during the training process using backpropagation. The values of these weights are adjusted to minimize the loss function, enabling the network to learn useful features from the data.
   * Each filter learns to detect specific features that are important for the given task (e.g., image classification, object detection).
3. **Spatial Hierarchy of Features:**
   * Convolutional layers with filters allow CNNs to learn a hierarchy of features. In the initial layers, filters may detect simple features like edges and textures. In deeper layers, filters can detect more complex patterns, such as parts of objects or even entire objects.
   * This hierarchical feature learning enables CNNs to understand and represent the input data at multiple levels of abstraction.
4. **Shared Weights:**
   * Filters in convolutional layers use shared weights across the entire input image. This weight sharing reduces the number of parameters in the network, making it more efficient and less prone to overfitting.
   * Shared weights also ensure that the same feature can be detected regardless of its position in the input image, contributing to the translational invariance of CNNs.
5. **Local Connectivity:**
   * Filters are locally connected to the input data, meaning they only operate on a small region (receptive field) of the input at a time. This local connectivity allows CNNs to focus on local patterns and spatial relationships.
   * As the network goes deeper, the receptive field of the filters increases, allowing the network to capture more global information.

## How does the stride affect the output size of convolutional layers?

The **stride** in a convolutional layer determines how far the filter moves at each step when scanning across the input. It directly affects the **spatial dimensions** (height and width) of the **output feature map**.

**Effect of Stride:**

1. **Stride = 1**:
   * The filter moves one unit at a time.
   * Output size is **maximally retained** (assuming padding).
2. **Stride > 1**:
   * The filter skips positions.
   * Output size is **reduced**, downsampling the input.
3. **Larger stride → Smaller output**:
   * E.g., going from stride 1 to stride 2 approximately **halves** the spatial size (if padding is unchanged).
4.  Stride **controls how much the filter moves** over the input.
5.  Larger stride = **more aggressive downsampling**.
6.  It's commonly used for **spatial reduction** in CNN architectures (e.g., in ResNet, MobileNet).

## Explain the term "feature map" in the context of CNNs.

In the context of **Convolutional Neural Networks (CNNs)**, a **feature map** refers to the **output of a convolutional layer** after applying a filter (or kernel) to the input data.

A **feature map** is a 2D matrix (or 3D if counting multiple channels) that captures the **activation values** resulting from a convolution operation over a specific **region of the input**, highlighting certain **features or patterns** like edges, textures, or shapes.

**Why is it called a "feature" map?**

* Each filter is trained to detect a specific type of **feature** (e.g., horizontal edge, corner, texture).
* The resulting map shows **where and how strongly** that feature appears in the input.

**Summary:**

* **Feature map** = output of a convolution applied to an input.
* It reveals **where specific learned features are present**.
* CNNs stack multiple feature maps to build **hierarchical representations** of the input (from edges to objects).

## What is the purpose of the fully connected (dense) layers in CNNs?

**Fully connected (FC) layers**, also called **dense layers**, are typically found at the **end of a Convolutional Neural Network (CNN)**. Their main purpose is to perform **high-level reasoning** and **final decision-making** based on the features extracted by the earlier convolutional and pooling layers.

**Key Functions:**

1. **Flatten and Interpret Features:**
   * The feature maps from the convolutional layers are **flattened** into a 1D vector.
   * The dense layer takes this vector and **learns non-linear combinations** of these features to make predictions.
2. **Classification or Regression Output:**
   * In **classification tasks**, the last dense layer typically outputs a vector of **class probabilities** (via softmax).
   * In **regression**, it may output a single continuous value or multiple numbers.

## How is weight sharing beneficial in CNNs?

**Weight sharing** is a key concept in Convolutional Neural Networks (CNNs), where the **same set of weights (filter/kernel)** is applied across different spatial locations of the input.

**Benefits of Weight Sharing:**

**1. Dramatically Reduces the Number of Parameters**

* Instead of learning a separate weight for every pixel connection (as in fully connected layers), CNNs use the **same weights (kernel)** across the whole input.
* **Example:**
  + A fully connected layer for a 32×32 image would need thousands of weights.
  + A 3×3 convolution only has 9 weights (plus bias), **regardless of image size**.

**2. Captures Translation Invariance**

* Features like edges or textures can appear **anywhere** in an image.
* By applying the same filter across the image, the CNN can **detect features in different positions**, making it **position-invariant** to a degree.

**3. Improves Training Efficiency**

* Fewer parameters → **faster training** and **less overfitting**.
* Less memory and computation needed.

**4. Encourages Generalization**

* Since filters are not specialized to one location, the network learns **general-purpose features** (like edge detectors) that work across the entire input.

### ****Summary Table:****

| **Benefit** | **Explanation** |
| --- | --- |
| **Fewer parameters** | Less memory, faster training |
| **Translation invariance** | Same feature detected across image |
| **Better generalization** | Prevents overfitting to specific locations |
| **Efficient computation** | Reuse of filters lowers cost |

## Describe the process of backpropagation in CNNs.

**Backpropagation** in Convolutional Neural Networks (CNNs) is the process of **updating the weights** (filters/kernels and dense layer weights) based on the **error** (loss) between the predicted and actual outputs. It ensures the model **learns** over time.

**Steps:**

1. **Forward Pass**: Input goes through convolutional, pooling, and dense layers → prediction made.
2. **Loss Computation**: Compare prediction to actual label using a loss function.
3. **Backward Pass**:
   * Compute gradients of loss w.r.t. output.
   * Backpropagate through fully connected, pooling, and convolutional layers using the **chain rule**.
   * Update **filter weights** and **biases**.
4. **Update**: Use an optimizer (e.g., SGD, Adam) to adjust parameters.

## What are some popular activation functions used in CNNs, and why?

### Summary :

| **Activation** | **Use Case** | **Output Range** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **ReLU** | Hidden layers | [0,∞) | Fast, avoids vanishing gradient | Can cause "dying" neurons |
| **Leaky ReLU** | Hidden layers | (−∞,∞) | Fixes ReLU issue | Slightly more complex |
| **Sigmoid** | Output (binary) | (0,1) | Outputs probabilities | Vanishing gradient, not zero-centered |
| **Tanh** | Hidden layers | (−1,1) | Zero-centered | Still suffers vanishing gradient |
| **Softmax** | Output (multi-class) | 0,1) (sums to 1) | Converts to probability distribution | Only used in final layer |

## What is batch normalization, and how does it help in training CNNs?

**Batch Normalization (BatchNorm)** is a technique used to **normalize the inputs** of each layer in a neural network so that they have **zero mean and unit variance** within a mini-batch.

**How It Helps in Training CNNs:**

1. **Stabilizes Training**
   * Keeps the outputs of layers close to zero and prevents them from becoming too large or too small.
2. **Speeds Up Convergence**
   * Allows higher learning rates and faster training.
3. **Improves Generalization**
   * Acts as a form of regularization (like dropout), helping prevent overfitting.
4. **Reduces Dependency on Initialization**
   * Less sensitive to weight initialization.

## How do dropout layers prevent overfitting in CNNs?

**Dropout** randomly **“drops” (sets to zero)** a fraction of neurons during training.

**Why this helps:**

1. **Reduces reliance on specific neurons:**  
   The network can’t depend too much on any single neuron because it might be dropped out randomly.
2. **Promotes redundancy:**  
   Forces the network to learn **multiple independent features** that work well together.
3. **Acts like model averaging:**  
   During training, it’s like training many smaller networks and averaging them at test time, improving generalization.

## Explain the terms "data augmentation" and its significance in CNNs.

**Data augmentation** is a technique to **artificially increase** the size and diversity of a training dataset by applying random transformations to the existing images.

**Common Augmentation Techniques:**

* Rotation
* Flipping (horizontal/vertical)
* Scaling/Zooming
* Cropping
* Translation (shifting)
* Adding noise
* Color jitter (changing brightness, contrast)

**Why Is It Important in CNNs?**

1. **Prevents Overfitting:**  
   More varied data helps the model generalize better instead of memorizing training images.
2. **Improves Robustness:**  
   Makes the model invariant to changes like rotation or lighting, improving performance on real-world data.
3. **Increases Dataset Size:**  
   Helps when you have limited labeled data by creating more training examples.

## What is transfer learning, and how can it be applied in CNNs?

**Transfer learning** is a technique where a model **pre-trained on a large dataset** is adapted to solve a different but related task with less data.

**How It’s Applied in CNNs:**

1. **Use a Pre-trained Model:**  
   Take a CNN model trained on a big dataset (like ImageNet).
2. **Reuse Early Layers:**  
   Freeze the initial convolutional layers that capture general features (edges, textures).
3. **Fine-tune Later Layers:**  
   Retrain the last few layers or add new layers to specialize in the new task.

**Why Use Transfer Learning?**

* Saves time and computational resources.
* Requires less labeled data.
* Often improves performance on smaller datasets.

## How would you evaluate the performance of a CNN model?

**Use a Separate Test Set**

* Evaluate the model on data it has **never seen** during training.

### ****Common Metrics:****

| **Task** | **Metrics** | **Description** |
| --- | --- | --- |
| **Classification** | - Accuracy - Precision - Recall - F1 Score - Confusion Matrix | Measures how well the model predicts classes |
| **Regression** | - Mean Squared Error (MSE) - Mean Absolute Error (MAE) | Measures prediction error magnitude |

## What loss functions are commonly used in CNNs for classification tasks?

| **Loss Function** | **Use Case** | **Description** |
| --- | --- | --- |
| **Cross-Entropy Loss** | Multi-class classification | Measures difference between predicted probabilities and true labels; most common for classification. |
| **Binary Cross-Entropy** | Binary classification | Special case of cross-entropy for two classes. |
| **Sparse Categorical Cross-Entropy** | Multi-class with integer labels | Similar to cross-entropy but uses integer class labels directly. |
| **Hinge Loss** | Sometimes used with SVM-like CNNs | Encourages correct classification with margin. |

## How does the learning rate impact the training of CNN models?

A hyperparameter that controls **how big a step** the optimizer takes when updating weights during training.

| **Learning Rate** | **Effect** |
| --- | --- |
| **Too High** | - Training becomes unstable- Loss may bounce or diverge- Model might never converge |
| **Too Low** | - Training is very slow- Can get stuck in local minima- Longer training time, possible underfitting |
| **Just Right** | - Smooth and efficient convergence- Faster learning with good accuracy |

## What are some challenges specific to training CNN models?

| **Challenge** | **Explanation** |
| --- | --- |
| **Overfitting** | CNNs can memorize training data, performing poorly on new data. |
| **Vanishing/Exploding Gradients** | Gradients can become too small or too large, slowing or destabilizing training. |
| **Large Computational Cost** | CNNs require significant GPU memory and processing power. |
| **Need for Large Datasets** | CNNs perform best with lots of labeled data, which may be hard to obtain. |
| **Choosing Hyperparameters** | Selecting learning rate, batch size, architecture, etc., can be complex. |
| **Sensitivity to Initialization** | Poor weight initialization can slow or prevent learning. |
| **Internal Covariate Shift** | Changing input distributions during training can slow convergence (mitigated by BatchNorm). |
| **Difficulty Interpreting Features** | CNNs can be black boxes, making it hard to understand what they learn. |

## How can you improve the training speed of CNNs?

| **Method** | **Explanation** |
| --- | --- |
| **Use GPUs/TPUs** | Parallelize computations to handle large matrix operations faster. |
| **Batch Normalization** | Stabilizes and speeds up training by normalizing layer inputs. |
| **Transfer Learning** | Start from pre-trained weights to reduce training time. |
| **Reduce Model Complexity** | Use smaller or shallower architectures when possible. |
| **Use Efficient Architectures** | Use models like MobileNet, EfficientNet designed for speed. |
| **Data Augmentation on the Fly** | Generate augmented data during training instead of saving all. |
| **Mixed Precision Training** | Use lower-precision (e.g., float16) to speed up computations. |
| **Adjust Batch Size** | Larger batch sizes improve GPU utilization but watch memory limits. |
| **Learning Rate Schedules** | Helps converge faster and avoid wasting time on slow training phases. |

## What are some applications of CNNs in computer vision?

| **Application** | **Description** |
| --- | --- |
| **Image Classification** | Assigning labels to images (e.g., cats vs dogs). |
| **Object Detection** | Identifying and localizing objects within images (e.g., detecting cars in a street). |
| **Image Segmentation** | Classifying each pixel to understand object boundaries (e.g., medical image analysis). |
| **Face Recognition** | Identifying or verifying people’s faces. |
| **Image Generation** | Creating new images using GANs or other CNN-based models. |
| **Pose Estimation** | Detecting human body joints and posture from images. |
| **Optical Character Recognition (OCR)** | Recognizing text from images or scanned documents. |
| **Video Analysis** | Action recognition, object tracking in videos. |
| **Super-Resolution** | Enhancing image resolution and details. |
| **Medical Imaging** | Detecting diseases from X-rays, MRIs, etc. |

## What are the limitations of CNNs compared to other neural network architectures?

| **Limitation** | **Explanation** |
| --- | --- |
| **Limited to Grid-like Data** | CNNs work best on images or data with spatial structure; less suited for sequential or graph data (where RNNs or GNNs excel). |
| **Require Large Amounts of Data** | Need lots of labeled data to generalize well, unlike some architectures that can work better with less data. |
| **Lack of Global Context** | CNNs focus on local features via kernels and may miss long-range dependencies unless combined with other methods. |
| **High Computational Cost** | Especially for deep or large CNNs, training requires significant GPU resources. |
| **Difficulty Handling Variable Input Sizes** | Standard CNNs expect fixed-size inputs; more flexible architectures like transformers handle variable sizes more naturally. |
| **Interpretability Issues** | Like many deep networks, CNNs are often seen as black boxes. |
| **Poor at Capturing Temporal Dependencies** | RNNs and Transformers are better suited for time-series or sequential data. |

## Explain the concept of receptive field in CNNs.

The **receptive field** of a neuron (or feature) in a CNN is the **region of the input image** that affects that neuron's output.

**Key Points:**

* In early layers, neurons "see" small local patches (e.g., 3x3 pixels).
* As you go deeper, neurons aggregate information from larger areas—so their receptive field grows.
* Larger receptive fields allow the network to capture more global context.

## How does the concept of stride affect the receptive field size?

 **Stride** is the step size with which the convolutional filter moves across the input.

 Increasing the stride **makes the receptive field grow faster** with each layer because the filter "jumps" over input positions.

| **Stride Value** | **Receptive Field Growth** |
| --- | --- |
| **Stride = 1** | Receptive field grows slowly, covering adjacent pixels layer by layer. |
| **Stride > 1** | Receptive field grows faster, skipping input pixels and covering a larger area more quickly. |