# ASSESMENT-4

**1.What is the purpose of the activation function in a neural network, and what are some commonly used activation functions?**

1. **Introducing non-linearity**: Activation functions introduce non-linearity into the network, allowing it to learn complex patterns and relationships in the data. Without non-linear activation functions, neural networks would behave essentially like linear models, unable to capture intricate patterns in the data.
2. **Enabling the network to learn complex mappings**: By introducing non-linearity, activation functions enable neural networks to approximate complex functions, making them capable of learning and representing highly non-linear relationships between inputs and outputs.
3. **Controlling the output range**: Activation functions often constrain the output of each neuron to a specific range, which can be crucial for ensuring stable and efficient training. For instance, some activation functions squash the output to lie within the range [0, 1] or [-1, 1], which can be advantageous in certain types of networks.

Some commonly used activation functions include:

- **Sigmoid function (Logistic** $f(x)=\frac{1}{1+e^{-(x)}}$ **function)**:

It squashes the input values to the range (0, 1) and is often used in the output layer of binary classification models.

- **Hyperbolic tangent function (tanh)**:

$$Tanh$$

$$f(x) = \frac{(e^{x}-e^{-x})}{(e^{x}+e^{-x})}$$

It squashes the input values to the range (-1, 1) and is commonly used in hidden layers of neural networks.

- **Rectified Linear Unit (ReLU)**:

$$ReLU$$

$$f(x) = max\ (0, x)$$

It outputs the input directly if it is positive, otherwise, it outputs zero. ReLU has become very popular due to its simplicity and effectiveness in training deep neural networks.

**2. Explain the concept of gradient descent and how it is used to optimize the parameters of a neural network during training.**

Gradient descent is an optimization algorithm used to minimize the loss function of a model by adjusting its parameters iteratively. In the context of neural networks, gradient descent is used to update the weights and biases of the network's neurons to reduce the error between the predicted output and the actual target output.

Here's how gradient descent works in the context of optimizing neural network parameters during training:

1. **Initialization**: Initially, the weights and biases of the neural network are randomly initialized.
2. **Forward Pass**: During the forward pass, the input data is fed through the network, and predictions are made based on the current parameter values. Each neuron computes its output based on the weighted sum of its inputs and applies an activation function to produce the output.
3. **Loss Calculation**: After obtaining predictions, the loss function is calculated to measure the difference between the predicted output and the actual target output. The loss function quantifies how well the model is performing on the given data.
4. **Backpropagation**: Backpropagation is used to compute the gradient of the loss function with respect to each parameter of the neural network. This is done by applying the chain rule of calculus, which allows the gradients to be efficiently calculated layer by layer, starting from the output layer and moving backward through the network.
5. **Gradient Descent Update**: Once the gradients have been computed, the parameters of the network (weights and biases) are updated in the opposite direction of the gradient to minimize the loss function. This update is performed according to the following equation for each parameter $\theta\theta$:

   $\theta=\theta-\eta\cdot\nabla J(\theta)$

   where $\eta\eta$ is the learning rate, a hyperparameter that controls the size of the step taken in the direction opposite to the gradient.

6. **Iterative Optimization**: Steps 2-5 are repeated iteratively for a fixed number of epochs or until the loss converges to a satisfactory level. During each iteration, the parameters are gradually adjusted to minimize the loss function and improve the performance of the neural network on the training data.

**3. How does backpropagation calculate the gradients of the loss function with respect to the parameters of a neural network?**

Backpropagation is a key algorithm used to train neural networks by efficiently computing the gradients of the loss function with respect to the parameters of the network. It involves two main steps: forward propagation and backward propagation.

1. **Forward Propagation**:

    - In forward propagation, the input data is fed through the network layer by layer, with each layer applying its weights and activation functions to produce an output.
    - The output of the last layer is compared to the ground truth labels using a loss function, which quantifies the difference between the predicted output and the actual target values.
2. **Backward Propagation**:
    - Backward propagation involves computing the gradients of the loss function with respect to the parameters of the network, which allows us to update the parameters in a direction that minimizes the loss.
    - The gradients are computed layer by layer, starting from the output layer and moving backward through the network.
    - **Output Layer**: The gradient of the loss function with respect to the output of the last layer is computed using the chain rule of calculus. For many common loss functions (e.g., mean squared error, cross-entropy), this gradient can be computed directly.
    - **Hidden Layers**: The gradients are then backpropagated through the network using the chain rule. At each layer, the gradients are computed by multiplying the gradients from the layer above by the local gradients of the activation function and the weights connecting the layers.

By iteratively performing forward and backward propagation on batches of training data, the network learns to adjust its parameters in a way that minimizes the loss function. This process is typically combined with an optimization algorithm such as gradient descent, which updates the parameters in the direction opposite to the gradient, gradually reducing the loss until convergence.

**4. Describe the architecture of a convolutional neural network (CNN) and how it differs from a fully connected neural network.**

A Convolutional Neural Network (CNN) is designed for processing grid-like data such as images. Its architecture is comprised of convolutional layers, pooling layers, activation functions, and fully connected layers.

1. **Convolutional Layers**: These layers apply filters across the input data, detecting spatial patterns like edges and textures. Filters slide across the input, performing element-wise multiplication and summation. Each layer typically has multiple filters.
2. **Pooling Layers**: These layers reduce the spatial dimensions of the feature maps while retaining important information. Max pooling is a common operation where the maximum value in each local region is extracted.
3. **Activation Functions**: Functions like ReLU introduce non-linearity, allowing the network to learn complex patterns. They are applied after convolutional and pooling operations.
4. **Fully Connected Layers**: After several convolutional and pooling layers, high-level features are flattened into a vector and passed through fully connected layers for classification or regression tasks.

**Differences from Fully Connected Neural Networks**:

- CNNs exploit the spatial structure of input data, whereas fully connected networks treat input as flat vectors.
- CNNs are more parameter efficient due to weight sharing and fewer parameters, making them suitable for tasks like image recognition.
- They are tailored for structured grid data like images, whereas fully connected networks are more generic in application.

**5. What are the advantages of using convolutional layers in CNNs for image recognition tasks?**

Using convolutional layers in Convolutional Neural Networks (CNNs) for image recognition tasks provides several advantages:

1. **Hierarchical Feature Extraction**: Convolutional layers automatically learn hierarchical features from low-level edges and textures to high-level objects and patterns. This hierarchical representation allows CNNs to understand the complex visual content of images.
2. **Parameter Sharing**: Convolutional layers use shared weights across the entire image, significantly reducing the number of parameters compared to fully connected networks. This parameter sharing makes CNNs more efficient and scalable, especially for large images and deep architectures.
3. **Translation Invariance**: Convolutional layers exhibit translation-invariant behavior, enabling them to detect features regardless of their location in the image. This property is crucial for recognizing objects in different positions, orientations, and scales within an image.
4. **Spatial Hierarchy Preservation**: Pooling layers within CNNs reduce the spatial dimensions of feature maps while retaining important information. This spatial hierarchy preservation ensures that even as the resolution decreases, the network still maintains essential details about the image's structure.
5. **Generalization**: CNNs trained on large datasets can generalize well to unseen data, capturing generic features that are common across different images. This generalization ability makes CNNs robust to variations in lighting, viewpoint, occlusions, and other transformations.
6. **Efficient Training**: The local connectivity and sparse interactions within convolutional layers lead to more efficient training compared to fully connected networks. This efficiency is particularly beneficial for training deep architectures with many layers.

**6. Explain the role of pooling layers in CNNs and how they help reduce the spatial dimensions of feature maps.**

**1. Dimension Reduction:**

- Pooling layers are crucial for reducing the spatial dimensions (width and height) of feature maps in CNNs.
- By downsampling the feature maps, pooling layers create a more compact representation of the input data.

**2. Types of Pooling:**

- Two common types of pooling operations are:
    - **Max Pooling:** Retains the maximum value within each local region.
    - **Average Pooling:** Computes the average value within each local region.

**3. Spatial Aggregation:**

- Pooling layers aggregate information from adjacent regions of the feature maps.
- They capture the most salient features while discarding less relevant details.

**4. Computational Efficiency:**

- Reducing the spatial dimensions of feature maps also reduces the computational burden of subsequent layers.
- This leads to faster and more efficient processing of the input data.

**5. Translational Invariance:**

- Pooling introduces a degree of translational invariance to the network.
- This makes the network more robust to variations in the position or orientation of features within the input.

**7. How does data augmentation help prevent overfitting in CNN models, and what are some common techniques used for data augmentation?**

1. **Increased Diversity**: Data augmentation artificially increases the diversity of the training dataset by generating variations of the original images. This exposes the model to a broader range of scenarios and helps prevent it from memorizing specific features of the training data.
2. **Regularization Effect**: By introducing variations such as translations, rotations, and flips, data augmentation imposes regularization on the model. This regularization encourages the network to learn more robust and generalizable features, reducing the likelihood of overfitting.
3. **Expanded Training Data**: With data augmentation, the effective size of the training dataset grows substantially. This larger dataset provides more samples for the model to learn from, reducing the risk of overfitting to the limited training samples.

**Common Techniques Used for Data Augmentation:**

1. **Rotation**: Rotating the image by a certain angle (e.g., 90 degrees) to introduce variations in object orientation.
2. **Horizontal and Vertical Flips**: Flipping the image horizontally or vertically to simulate different viewpoints or perspectives.
3. **Translation**: Shifting the image horizontally or vertically within its boundaries, mimicking changes in position.
4. **Scaling**: Zooming in or out of the image to simulate variations in object size or distance.
5. **Shearing**: Applying a shear transformation to the image, deforming it along one axis while keeping the other axis fixed.
6. **Brightness and Contrast Adjustment**: Altering the brightness, contrast, or color balance of the image to simulate changes in lighting conditions.

7. **Noise Injection**: Adding random noise to the image to mimic imperfections in data acquisition or processing.

**8. Discuss the purpose of the flatten layer in a CNN and how it transforms the output of convolutional layers for input into fully connected layers**

The flatten layer in a Convolutional Neural Network (CNN) serves the purpose of transforming the output of convolutional layers into a format that can be input into fully connected layers. Here's how it achieves this transformation:

**Purpose of the Flatten Layer:**

1. **Transition from Convolutional Layers to Fully Connected Layers**: The output of convolutional layers in a CNN is typically in the form of a multi-dimensional array or tensor. Each dimension represents features extracted from different spatial locations of the input image.
2. **Vectorization of Features**: Before passing the features to fully connected layers, it is necessary to convert this multi-dimensional output into a one-dimensional vector. The flatten layer performs this operation by "flattening" or unraveling the multi-dimensional array into a single vector.
3. **Compatibility with Fully Connected Layers**: Fully connected layers require a one-dimensional input where each element corresponds to a specific feature or neuron. By flattening the output of convolutional layers, the features from different spatial locations are concatenated into a single vector, which can then be fed into the fully connected layers.

**Transformation Process by the Flatten Layer:**

- **Input**: The output of convolutional layers consists of feature maps with height, width, and depth dimensions, representing spatial features extracted from different regions of the input image.
- **Flatten Operation**: The flatten layer takes the multi-dimensional feature maps as input and reshapes them into a one-dimensional array. It preserves the depth dimension while combining the spatial dimensions into a single dimension.
- **Output**: The output of the flatten layer is a one-dimensional vector containing all the features extracted by the convolutional layers. Each element of this vector represents a specific feature or neuron, ready to be input into the fully connected layers for further processing.

**9. What are fully connected layers in a CNN, and why are they typically used in the final stages of a CNN architecture?**

Fully connected layers, also known as dense layers, are a fundamental component of Convolutional Neural Networks (CNNs) used for tasks such as classification and regression. Here's an explanation of what fully connected layers are and why they are typically used in the final stages of a CNN architecture:

**Fully Connected Layers:**

1. **Structure**: Fully connected layers consist of neurons where each neuron is connected to every neuron in the previous and subsequent layers. This means that every input feature is connected to every neuron in the fully connected layer.
2. **Parameterization**: Each connection between neurons in fully connected layers is associated with a learnable weight parameter. During training, these weights are adjusted through backpropagation to minimize the loss function, enabling the network to learn complex mappings between inputs and outputs.
3. **Non-Linearity**: Fully connected layers are typically followed by non-linear activation functions such as ReLU (Rectified Linear Unit) or softmax, which introduce non-linearity into the network, allowing it to learn and represent complex relationships in the data.

**Why Fully Connected Layers are Used in the Final Stages of a CNN:**

1. **Global Information Aggregation**: In the final stages of a CNN architecture, convolutional and pooling layers have progressively downsampled and abstracted the input data, resulting in high-level feature representations. Fully connected layers are then used to aggregate global information from these features, allowing the network to make predictions based on the overall context of the input.
2. **Classification or Regression**: Fully connected layers are well-suited for tasks such as classification or regression, where the network needs to make a final decision based on the learned features. The dense connections in fully connected layers enable the network to model complex decision boundaries and output class probabilities or regression values.
3. **Output Layer**: The last fully connected layer in a CNN architecture typically serves as the output layer. For classification tasks, this layer often contains neurons corresponding to the number of classes, with softmax activation to output class probabilities. For regression tasks, the output layer may contain a single neuron with linear activation to output continuous values.

**10. Describe the concept of transfer learning and how pre-trained models are adapted for new tasks.**

Transfer learning is a machine learning technique where a model trained on one task is repurposed or adapted for a different but related task. It leverages the knowledge learned from the source task and applies it to the target task, typically resulting in improved performance, faster convergence, and reduced need for large amounts of labeled data.

**Concept of Transfer Learning:**

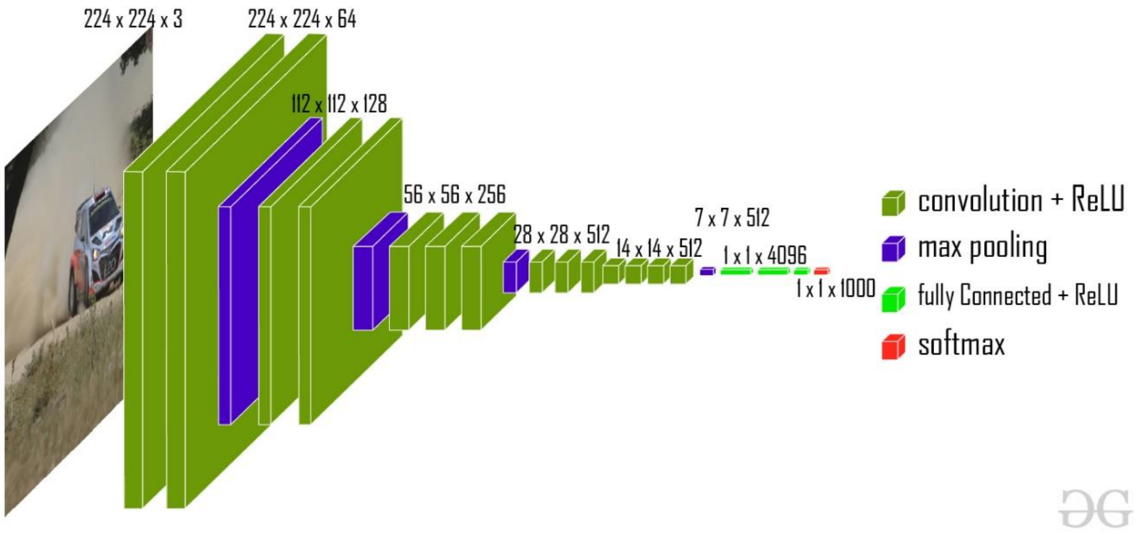
1. **Source and Target Tasks**: In transfer learning, there are typically two tasks: a source task, where a model is trained on a large dataset for a specific task (e.g., image classification), and a target task, where the goal is to solve a related but different task (e.g., object detection).
2. **Learning Representations**: During training on the source task, the model learns to extract relevant features and representations from the input data. These learned representations capture useful information about the underlying structure of the data.
3. **Knowledge Transfer**: Instead of discarding the learned representations after training on the source task, transfer learning involves transferring this knowledge to the target

task. By reusing the learned representations, the model can bootstrap its learning process on the target task, often with minimal additional training data.

**Adapting Pre-trained Models for New Tasks:**

1. **Feature Extraction**: One common approach is to use pre-trained models as feature extractors. In this approach, the weights of the pre-trained model are frozen, and only the final layers (e.g., fully connected layers) are replaced or added and trained on the target task.
2. **Fine-tuning**: Another approach is fine-tuning, where the entire pre-trained model, or part of it, is fine-tuned on the target task. In this approach, the weights of the pre-trained model are updated during training on the target task, allowing the model to adapt its learned representations to better suit the target domain.
3. **Domain Adaptation**: In some cases, the source and target domains may have differences that affect performance. Domain adaptation techniques aim to bridge this gap by aligning the distributions of the source and target domains, either through data augmentation, adversarial training, or other methods.
4. **Task-specific Modifications**: Depending on the complexity and nature of the target task, additional modifications may be made to the pre-trained model architecture, such as adding task-specific layers, adjusting hyperparameters, or applying regularization techniques.

**11. Explain the architecture of the VGG-16 model and the significance of its depth and convolutional layers.**



The VGG-16 model is a convolutional neural network architecture introduced by the Visual Geometry Group (VGG) at the University of Oxford. It is renowned for its simplicity and effectiveness, especially in image classification tasks. Here's an overview of the architecture of the VGG-16 model and the significance of its depth and convolutional layers:

**Architecture of VGG-16:**

1. **Input Layer**: Accepts input images of size 224x224x3 (RGB images).

2. **Convolutional Layers**: The VGG-16 model consists of 13 convolutional layers, each followed by a ReLU activation function. These convolutional layers use small 3x3 filters with a stride of 1 and padding to maintain the spatial dimensions of the input.
3. **Pooling Layers**: After every two convolutional layers, max-pooling layers with 2x2 filters and a stride of 2 are used to reduce the spatial dimensions of the feature maps.
4. **Fully Connected Layers**: Towards the end of the network, there are three fully connected layers followed by a softmax layer for classification. These layers perform high-level feature aggregation and classification based on the features learned by the preceding convolutional layers.
5. **Output Layer**: The final softmax layer outputs class probabilities for classification tasks.

**Significance of Depth:**

1. **Hierarchical Feature Learning**: The depth of the VGG-16 model allows it to learn hierarchical representations of input images. Each layer captures increasingly abstract and complex features, starting from low-level edges and textures to high-level object shapes and patterns.
2. **Improved Discriminative Power**: Deeper networks like VGG-16 can capture more discriminative features, enabling them to learn more sophisticated decision boundaries and achieve better performance on complex tasks such as image classification.

**Significance of Convolutional Layers:**

1. **Feature Extraction**: The convolutional layers in VGG-16 act as feature extractors, capturing local patterns and structures from the input images. These patterns are then progressively combined and aggregated in subsequent layers to form higher-level representations.
2. **Parameter Sharing**: By using shared weights in the convolutional filters, VGG-16 reduces the number of parameters and encourages the network to learn translation-invariant features, making it more robust to variations in object position and orientation.
3. **Sparse Connectivity**: The convolutional layers in VGG-16 exhibit sparse connectivity, meaning each output feature map is only connected to a subset of input feature maps. This promotes efficient learning and reduces computational complexity.

**12. What are residual connections in a ResNet model, and how do they address the vanishing gradient problem?**

Residual connections, also known as skip connections, are a key architectural component introduced in Residual Neural Networks (ResNets). They address the vanishing gradient problem, which can occur in very deep neural networks during training, by facilitating the flow of gradients through the network.

**Concept of Residual Connections:**

1. **Identity Mapping:** In a residual connection, the input to a layer is added directly to the output of one or more layers deeper in the network. Mathematically, if xx represents the input to a layer, and F(x)F(x) represents the output of the layer, the residual connection computes F(x)+xF(x)+x.

2. **Shortcut Connections:** These connections "skip" one or more layers in the network by directly passing the input (or a transformed version of it) to a later layer, bypassing intermediate transformations. The output of the later layer is then added back to the original input.

**Addressing the Vanishing Gradient Problem:**

1. **Gradient Propagation:** Residual connections enable easier gradient propagation through the network. During backpropagation, the gradients can flow directly through the skip connections, allowing the gradients to bypass the intermediate layers where they may become diluted or vanish.
2. **Facilitate Training of Deep Networks:** By mitigating the vanishing gradient problem, residual connections enable the training of very deep neural networks with hundreds or even thousands of layers. This is crucial for capturing increasingly complex patterns and features in large-scale datasets.
3. **Stabilize Training:** Residual connections provide a mechanism for stabilizing the training process in deep networks. By allowing gradients to flow more freely, they prevent degradation in performance as the network depth increases, leading to faster convergence and improved generalization.
4. **Promote Feature Reuse:** The addition of input features to the output encourages feature reuse across layers. This helps in preserving useful information from earlier layers and prevents the loss of valuable information during deep network training.

## 13. Discuss the advantages and disadvantages of using transfer learning with pre-trained models such as Inception and Xception

Residual connections, introduced in the ResNet (Residual Network) model, are a technique used to address the vanishing gradient problem commonly encountered in deep neural networks during training. Here's an explanation of residual connections and how they mitigate the vanishing gradient problem:

**Residual Connections:**

1. **Shortcut Connections**: In a residual block, the input to a layer is added to the output of one or more layers downstream. This is called a shortcut or skip connection.
2. **Identity Mapping**: The shortcut connection enables the network to learn residual functions, allowing the network to fit the identity mapping (i.e., the original input) easily.
3. **Gradient Flow**: During backpropagation, the gradient can flow directly through the shortcut connections, bypassing the layers in between. This helps in alleviating the vanishing gradient problem by providing a direct path for the gradients to propagate through the network.
4. **Residual Learning**: By learning residual functions, ResNet models can effectively tackle the optimization challenges associated with training very deep networks, leading to improved convergence and performance.

**Advantages of Transfer Learning with Pre-trained Models:**

1. **Feature Reuse**: Pre-trained models such as Inception and Xception have been trained on large-scale datasets like ImageNet, learning rich feature representations from

diverse visual data. Transfer learning allows us to leverage these learned features, saving time and computational resources.
2. **Improved Generalization**: Pre-trained models capture generic visual patterns and structures that are useful across different tasks and domains. Transfer learning enables the transfer of this knowledge to new tasks, leading to better generalization and performance, especially when training data is limited.
3. **Faster Convergence**: Transfer learning with pre-trained models often leads to faster convergence during training on the target task. By initializing the model with pre-trained weights, the network starts with a better initialization point, reducing the time required for training and fine-tuning.

**Disadvantages of Transfer Learning with Pre-trained Models:**

1. **Domain Mismatch**: Pre-trained models may not always generalize well to the target domain if there are significant differences between the source and target datasets. In such cases, fine-tuning or domain adaptation techniques may be necessary to adapt the pre-trained model to the target task.
2. **Limited Flexibility**: Pre-trained models are trained on specific tasks and datasets, which may not perfectly align with the requirements of the target task. Fine-tuning the pre-trained model or modifying its architecture may be necessary to achieve optimal performance on the target task.
3. **Model Size and Complexity**: Pre-trained models like Inception and Xception are often large and complex, requiring significant computational resources for training and deployment. This can be a disadvantage, especially in resource-constrained environments.

**14. How do you fine-tune a pre-trained model for a specific task, and what factors should be considered in the fine-tuning process?**

Fine-tuning a pre-trained model involves taking a pre-trained model that has been trained on a large dataset and adapting it to perform a specific task or on a specific dataset. Here's a general overview of the fine-tuning process and the factors to consider:

**Fine-tuning Process:**

1. **Select Pre-trained Model**: Choose a pre-trained model that is well-suited for the target task and dataset. Common choices include models trained on large-scale image datasets like ImageNet, such as ResNet, Inception, or VGG.
2. **Remove or Freeze Layers**: Depending on the similarity between the pre-trained model's original task and the target task, you may choose to freeze some or all of the layers in the pre-trained model. Freezing layers means that their weights will not be updated during fine-tuning. If the pre-trained model's features are highly relevant to the target task, you can keep more layers frozen. Otherwise, you may choose to remove some layers or fine-tune all layers.
3. **Replace or Add Output Layers**: Replace the original output layer(s) of the pre-trained model with new layers suitable for the target task. For classification tasks, this often involves adding a new fully connected layer with the appropriate number of output units (equal to the number of classes). For other tasks, such as object detection or segmentation, you may need to modify the output layers accordingly.

4. **Fine-tune Model**: Train the modified model on the target dataset using the desired optimization algorithm (e.g., stochastic gradient descent) and loss function. During training, the weights of the unfrozen layers are updated to minimize the loss on the target task.
5. **Regularization and Optimization**: Apply regularization techniques such as dropout or weight decay to prevent overfitting during fine-tuning. Additionally, adjust hyperparameters such as learning rate and batch size to optimize training performance.
6. **Monitor Performance**: Monitor the performance of the fine-tuned model on a validation set to assess its generalization ability. Make adjustments to the fine-tuning process as needed based on validation performance.
7. **Evaluate and Deploy**: Once the fine-tuning process is complete, evaluate the fine-tuned model on a separate test set to obtain final performance metrics. If satisfactory, deploy the fine-tuned model for inference on new data.

**Factors to Consider in the Fine-tuning Process:**

1. **Similarity between Tasks**: Consider the similarity between the pre-trained model's original task and the target task. More similar tasks may require less aggressive fine-tuning, while less similar tasks may require more extensive modifications.
2. **Amount of Available Data**: The amount of available labeled data for the target task influences the fine-tuning process. With a small amount of data, more regularization and careful fine-tuning are required to prevent overfitting.
3. **Model Architecture**: The architecture of the pre-trained model can impact the fine-tuning process. Deeper models may require more careful fine-tuning due to a larger number of parameters.
4. **Computational Resources**: Fine-tuning deep neural networks can be computationally intensive, particularly if training on large datasets or with complex models. Consider available computational resources when planning the fine-tuning process.
5. **Performance Requirements**: Consider the desired performance metrics and requirements for the target task. Fine-tuning may need to be adjusted to optimize for specific metrics such as accuracy, precision, or recall.

By carefully considering these factors and following best practices for fine-tuning, you can effectively adapt a pre-trained model to perform well on a specific task or dataset

**15. Describe the evaluation metrics commonly used to assess the performance of CNN models, including accuracy, precision, recall, and F1 score.**

Evaluation metrics are essential for assessing the performance of Convolutional Neural Network (CNN) models in various tasks such as image classification, object detection, and segmentation. Here's an overview of commonly used evaluation metrics for CNN models:

1. **Accuracy**:
    - Accuracy measures the proportion of correctly classified instances out of the total instances.
    - Accuracy= $\frac{TP+TN}{TP+TN+FP+FN}$
    - TP: True Positives (correctly predicted positive instances)
    - TN: True Negatives (correctly predicted negative instances)

   - FP: False Positives (incorrectly predicted positive instances)
   - FN: False Negatives (incorrectly predicted negative instances)
   - Accuracy provides a general measure of the model's overall performance.
2. **Precision**:
   - Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive.
   - Precision= $\frac{TP}{TP+FP}$
   - Precision focuses on the correctness of positive predictions and is useful when minimizing false positives is critical.
3. **Recall (Sensitivity)**:
   - Recall measures the proportion of correctly predicted positive instances out of all actual positive instances.
   - Recall= $\frac{TP}{TP+FN}$
   - Recall focuses on capturing all positive instances and is important when minimizing false negatives is crucial.
4. **F1 Score**:
   - The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics.
   - F1= $\frac{2\times Precision\times Recall}{Precision+Recall}$
   - The F1 score is useful when there is an imbalance between the classes or when both precision and recall are equally important.