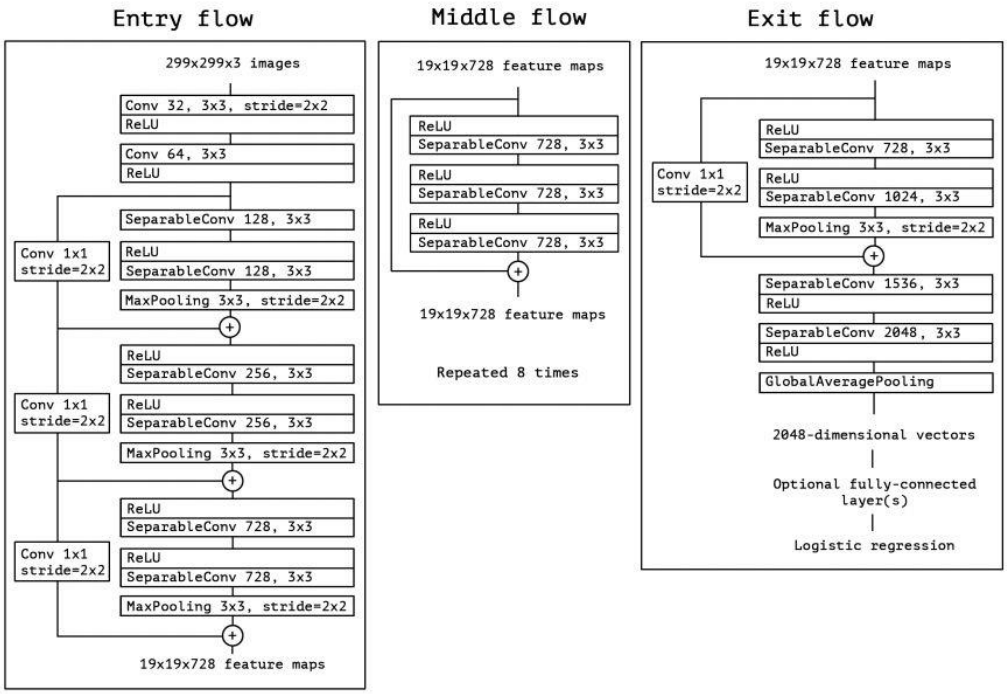
1- Xception model

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions.

What does it look like?



How does it work?

1- Input Layer

- The model begins with an input layer that typically accepts images of size 299×299×3299 \times 299 \times 3299×299×3 (height, width, and RGB channels).

2- Entry Flow

- Initial Convolution and Max Pooling
- Residual Convolutional Blocks

Step 3: Middle Flow

- Depthwise Separable Convolutions

Step 4: Exit Flow

- Feature Extraction with Downsampling
- Global Average Pooling
- Output Layer

XCeption is an efficient architecture that relies on two main points :

- Depthwise Separable Convolution.
- Shortcuts between Convolution blocks as in ResNet.

1. Depthwise Separable Convolution

Depthwise separable convolution is a more efficient alternative to standard convolution, where the operation is split into two smaller steps:

a. Standard Convolution (Traditional Approach)

- Combines spatial filtering and channel-wise transformations in a single operation.
- Computationally expensive as the operation involves a kernel of size k×kk \times kk×k applied to all input channels.
- Number of parameters: k×k×Input Channels×Output Channelsk \times k \times \text{Input Channels} \times \text{Output Channels}k×k×Input Channels×Output Channels.

b. Depthwise Separable Convolution (Used in Xception)

- Breaks the standard convolution into **two distinct operations**:
    1. **Depthwise Convolution:**
        - Applies a single convolutional filter per input channel.
        - Captures spatial patterns within each channel independently.
        - Reduces computation significantly because each channel is filtered separately.
    2. **Pointwise Convolution:**
        - Applies a 1×11 \times 11×1 convolution across all channels to combine the outputs of the depthwise convolution.

    - This step is responsible for learning inter-channel correlations.

2. Shortcuts (Residual Connections) between Convolution Blocks

The Xception architecture also incorporates **shortcuts** or **residual connections**, a concept introduced in **ResNet**, to enhance training stability and model performance.

How Residual Connections Work

- Residual connections create **shortcuts** by directly adding the input of a convolutional block to its output.
- Mathematically: Output=F(Input)+Input\text{Output} = \text{F(Input)} + \text{Input}Output=F(Input)+Input Where FFF represents the series of convolutional operations within the block.

Purpose of Shortcuts

- **Avoid Vanishing Gradients:** Shortcuts help gradients flow back to earlier layers during backpropagation, ensuring stable training in deep networks.
- **Efficient Learning:** Residual connections allow the network to learn the difference between input and output (the "residual") rather than the entire transformation, which simplifies optimization.
- **Reusability of Features:** Encourages the reuse of features across layers, leading to better generalization.

In Xception

- Shortcuts are used between **convolution blocks**:

- Each block contains one or more depthwise separable convolutions followed by batch normalization and ReLU activation.
- The shortcut skips the depthwise separable convolution stack, directly connecting the input of the block to its output.

Advantages of Xception

1. **Efficient Use of Parameters**:
    a. By replacing standard convolutions with **depthwise separable convolutions**, Xception reduces the number of parameters and computational cost without sacrificing accuracy.
    b. This efficiency allows the model to scale well to large datasets.
2. **Improved Feature Extraction**:
    a. Xception excels in capturing spatial and channel-wise relationships separately, leading to better feature extraction compared to standard convolutional networks or even the Inception architecture.
3. **Simpler Architecture**:
    a. The architecture is conceptually simpler than Inception models because it eliminates the need for complex modules. Depthwise separable convolutions replace inception blocks entirely.
4. **State-of-the-Art Performance**:
    a. Xception achieves excellent results on image classification benchmarks like ImageNet, often outperforming models with higher complexity.
5. **Flexibility in Transfer Learning**:
    a. Pre-trained Xception models are highly effective for transfer learning across various domains, such as object detection, semantic segmentation, and medical imaging.
6. **Scalability**:

a. The model can handle high-dimensional inputs and scale to larger datasets or tasks requiring fine-grained features.

Limitations of Xception

1. **Computational Overhead for Small Datasets**:
    a. While depthwise separable convolutions are computationally efficient, the full potential of Xception is realized on large datasets. On smaller datasets, its complexity may lead to overfitting.
2. **Hardware Requirements**:
    a. Despite parameter efficiency, the operations involved in depthwise separable convolutions may require optimized hardware (e.g., GPUs or TPUs) to run efficiently, especially for real-time applications.
3. **Sensitivity to Hyperparameters**:
    a. Xception's performance can be sensitive to choices such as learning rate, batch size, and regularization techniques, requiring careful tuning.
4. **Not Ideal for All Tasks**:
    a. Tasks involving 3D data, sequential data (e.g., text or time series), or datasets where features require inter-channel dependencies not well-separated spatially may benefit less from Xception.
5. **Relatively High Memory Usage**:
    a. Although efficient in computation, the deeper nature of the model may still require significant memory resources for training and inference compared to smaller architectures like MobileNet.

2- DenseNet model

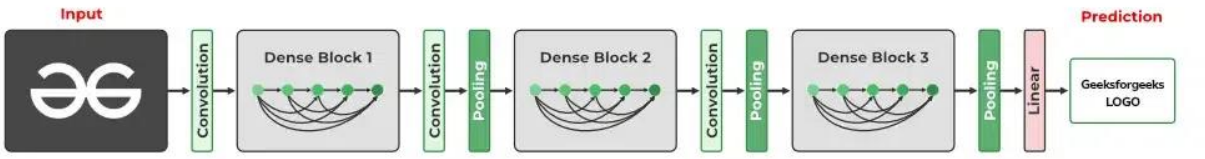
DenseNet, short for Dense Convolutional Network, is a deep learning architecture for convolutional neural networks (CNNs) introduced by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger in their paper titled "Densely Connected Convolutional Networks" published in 2017. DenseNet revolutionized the field of computer vision by proposing a novel connectivity pattern within CNNs, addressing challenges such as feature reuse, vanishing gradients, and parameter efficiency. Unlike traditional CNN architectures where each layer is connected only to subsequent layers, DenseNet establishes direct connections between all layers within a block. This dense connectivity enables each layer to receive feature maps from all preceding layers as inputs, fostering extensive information flow throughout the network.

Key Characteristics of DenseNet

1. **Alleviated Vanishing Gradient Problem**: Dense connections ensure that gradients can flow directly to earlier layers, mitigating the vanishing gradient issue common in deep networks.
2. **Improved Feature Propagation**: Each layer has direct access to the gradients from the loss function and the original input signal, promoting better feature propagation.
3. **Feature Reuse**: By concatenating features from all preceding layers, DenseNet encourages feature reuse, reducing redundancy and improving efficiency.
4. **Reduced Parameters**: Despite its dense connections, DenseNet is parameter-efficient. It eliminates the need to relearn redundant features, resulting in fewer parameters compared to traditional networks.

Architecture of DenseNet

DenseNet introduces a paradigm shift by connecting each layer to every other layer in a feed-forward manner. Unlike traditional CNNs, which have a single connection between consecutive layers, DenseNet ensures that each layer receives inputs from all preceding layers and passes its output to all subsequent layers. This results in a network with $L(L+1)/2$ direct connections for L layers, significantly enhancing information flow.



Dense Block

Dense blocks are the building blocks of DenseNet architectures. Each dense block consists of multiple convolutional layers, typically followed by batch normalization and a non-linear activation function (e.g., ReLU). Importantly, each layer within a dense block receives feature maps from all preceding layers as inputs, facilitating feature reuse and propagation.

Within a dense block, each layer receives the concatenated output of all preceding layers as its input. If a dense block has m layers, and each layer produces k feature maps (where k is known as the growth rate), the l-th layer will have k×(l+l0)*k*×(*l*+*l*0) input feature maps (where l0*l*0 is the number of input channels to the dense block).

Example of a Dense Block:

1. **Layer 1**: Receives input feature maps.
2. **Layer 2**: Receives input feature maps + output of Layer 1.
3. **Layer 3**: Receives input feature maps + output of Layer 1 + output of Layer 2.

This pattern continues for all layers within the block, ensuring a highly interconnected architecture.

Transition Layer

Transition layers are used to connect dense blocks. They serve two main purposes: reducing the number of feature maps and downsampling the spatial dimensions of the

feature maps. This helps in maintaining the computational efficiency and compactness of the network. A typical transition layer consists of:

- **Batch Normalization**: Normalizes the feature maps.
- **1x1 Convolution**: Reduces the number of feature maps.
- **Average Pooling**: Downsamples the spatial dimensions.

Growth Rate (k)

The growth rate (k) is a critical hyperparameter in DenseNet. It defines the number of feature maps each layer in a dense block produces. A larger growth rate means more information is added at each layer, but it also increases the computational cost. The choice of k affects the network's capacity and performance.

Advantages of DenseNet

1. **Reduced Vanishing Gradient Problem**: Dense connections improve gradient flow and facilitate the training of very deep networks.
2. **Feature Reuse**: Each layer has access to all preceding layers' feature maps, promoting the reuse of learned features and enhancing learning efficiency.
3. **Fewer Parameters**: DenseNets often have fewer parameters compared to traditional CNNs with similar depth due to efficient feature reuse.
4. **Improved Accuracy**: DenseNets have shown high accuracy on various benchmarks, such as ImageNet and CIFAR.

Limitations of DenseNet

1. **High Memory Consumption**: Dense connections increase memory usage due to the storage requirements for feature maps, making DenseNet less practical for devices with limited memory.
2. **Computational Complexity**: The extensive connectivity leads to increased computational demands, resulting in longer training times and higher computational costs, which may not be ideal for real-time applications.
3. **Implementation Complexity**: Managing and concatenating a large number of feature maps adds complexity to the implementation, requiring careful tuning of hyperparameters and regularization techniques to maintain performance and stability.

4. **Risk of Overfitting**: Although DenseNet reduces overfitting through better feature reuse, there is still a risk, particularly if the network is not properly regularized or if the training data is insufficient.

Applications of DenseNet

DenseNet is versatile and can be applied to various tasks in computer vision, including:

- **Image Classification**: DenseNet's ability to extract rich feature representations makes it suitable for image classification tasks.
- **Object Detection**: DenseNet can be used as a backbone for object detection networks, providing detailed feature maps for accurate detection.
- **Semantic Segmentation**: DenseNet's dense connections help in capturing fine details, making it effective for semantic segmentation tasks.

3- ResNet model

ResNet, short for **Residual Network**, is a deep learning model introduced in the groundbreaking 2015 paper "Deep Residual Learning for Image Recognition" by He et al. It addressed a critical challenge in training very deep neural networks: the vanishing gradient problem. This problem often makes deeper models harder to train effectively as gradients diminish during backpropagation, leading to poor convergence.

ResNet introduced the concept of **residual learning**, which allowed networks to be significantly deeper than before without suffering from degradation in performance. This innovation earned ResNet a decisive victory in the ImageNet 2015 competition and established it as a foundational architecture in modern deep learning.

Architecture of ResNet

At the heart of ResNet is the **residual block**, which introduces **skip connections** (or shortcuts). These connections allow the network to bypass one or more layers, effectively enabling identity mapping for certain transformations. The architecture consists of:

1. **Convolutional Layers**

2. These extract spatial features from input images. Each convolutional layer is followed by Batch Normalization and a ReLU activation function to normalize and introduce non-linearity.

3. **Residual Blocks**

A residual block contains:

   a. **Two or more convolutional layers**: Extract features.
   b. **Skip connection**: Adds the input of the block directly to its output using an element-wise addition.

Mathematically, if the input to a block is xxx, and the transformation by convolutional layers is $F(x)F(x)$F(x), the output is:

$Output=F(x)+x|text{Output}} = F(x) + x$Output=F(x)+x

This bypass mechanism is the core of ResNet's success, as it ensures that the network learns the residual $F(x)F(x)$F(x) instead of trying to learn a direct mapping.

4. **Pooling Layers**

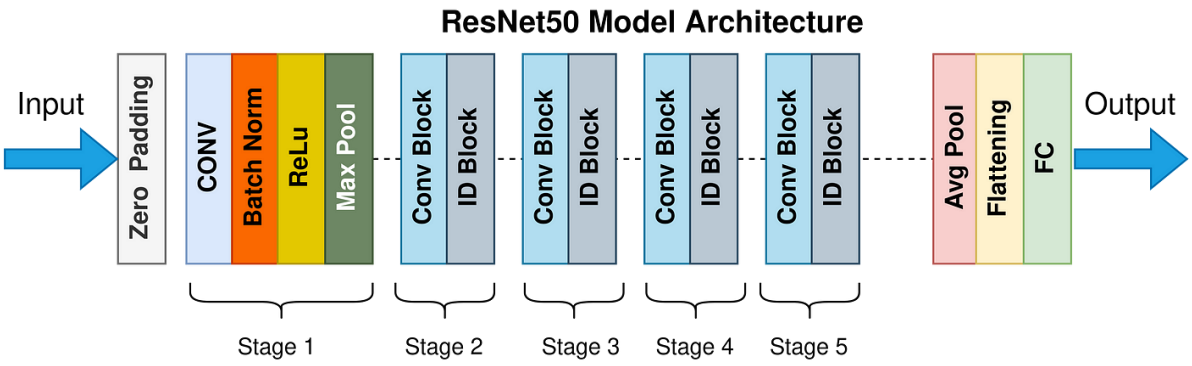
Max-pooling or average-pooling layers are used to downsample the feature maps, reducing their spatial dimensions while retaining important features.

5. **Fully Connected Layers**

At the end, fully connected layers are used to perform classification or regression based on the features extracted by the convolutional layers.

6. **Softmax Layer**

For multi-class classification tasks, the final layer typically uses softmax activation.



When is ResNet Useful?

1. **Image Classification**

ResNet has become the de facto choice for image classification tasks, performing exceptionally well on datasets like ImageNet and CIFAR.

2. **Feature Extraction**

In transfer learning, pre-trained ResNet models are often used as feature extractors for new datasets, saving computational resources.

3. **Object Detection and Segmentation**

ResNet serves as the backbone for advanced models like Faster R-CNN and Mask R-CNN in computer vision tasks.

4. **Tasks Requiring Deep Networks**

When very deep architectures are needed (e.g., medical imaging, satellite data analysis), ResNet's residual learning makes these tasks feasible.

Advantages of ResNet

1. **Enables Very Deep Architectures**:
2. ResNet can support networks with hundreds or even thousands of layers without degradation in performance, a task nearly impossible with traditional architectures.
3. **Improved Gradient Flow**:

Skip connections help mitigate the vanishing gradient problem, ensuring that gradients flow freely back to earlier layers during training.

4. **Simplified Optimization**:

Residual learning focuses on learning the residual mapping $F(x)=H(x)-xF(x) = H(x)$ - xF(x)=H(x)−x, which is often easier to optimize than learning the full mapping $H(x)H(x)$H(x).

5. **Robust Generalization**:

ResNet models generalize well across different datasets and tasks, from image classification to transfer learning and object detection.

6. **Transfer Learning Ready**:

Pre-trained ResNet models on large datasets like ImageNet are highly effective as feature extractors, saving computational time and resources for new tasks.

7. **Scalability**:

The modular structure of residual blocks allows the architecture to scale effortlessly, making it adaptable to tasks requiring varied levels of complexity.

Limitations of ResNet

1. **Computational Cost**:

Deep ResNet architectures with many layers can be computationally expensive in terms of memory and processing power, particularly on limited hardware.

2. **Diminishing Returns with Depth**:

While ResNet enables deeper architectures, excessively deep models (e.g., over 1000 layers) may not significantly improve performance on some tasks, leading to diminishing returns.

3. **Overfitting Risk**:

If not managed properly, very deep ResNet models can overfit smaller datasets, necessitating careful regularization techniques.

4. **Difficulty in Designing Optimal Depth**:

Choosing the appropriate depth for a ResNet model requires experimentation and domain expertise, as overly shallow or excessively deep models may underperform.

5. **Model Size**:

ResNet models can have a large number of parameters, making them unsuitable for deployment on resource-constrained devices like smartphones.

Applications of ResNet

1. Image Classification

ResNet excels at image classification tasks, as demonstrated in its performance on benchmarks like ImageNet, where it significantly outperformed competitors.

2. Transfer Learning

Pre-trained ResNet models are widely used for transfer learning, where the learned weights from large datasets like ImageNet are fine-tuned on smaller, task-specific datasets.

3. Object Detection

ResNet serves as a backbone for object detection frameworks like Faster R-CNN and YOLO, enabling these models to identify and localize objects in images effectively.

4. Semantic Segmentation

In tasks like medical imaging and autonomous driving, ResNet is often the backbone for segmentation models such as U-Net or DeepLab.

5. Generative Models

ResNet has been adapted in architectures like StyleGAN and CycleGAN to provide a stable and robust foundation for generating high-quality images.

6. Video Processing

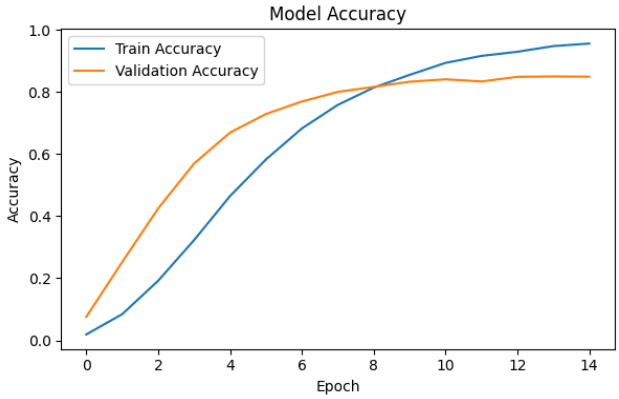
Variants of ResNet, such as ResNet-3D, are used for video classification and action recognition, extracting spatiotemporal features from video frames.

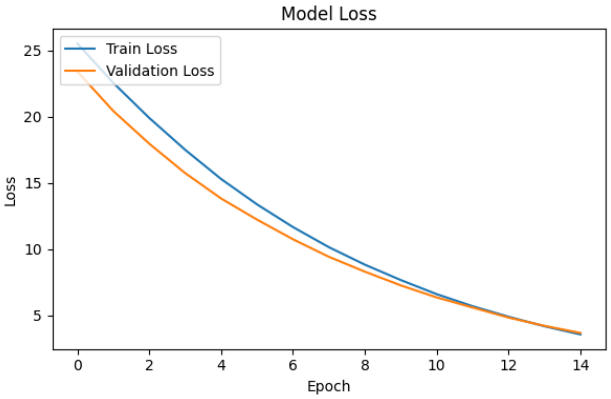
7. Medical Imaging

ResNet has found applications in detecting anomalies in X-rays, MRIs, and CT scans, where deep features help identify patterns missed by conventional methods.

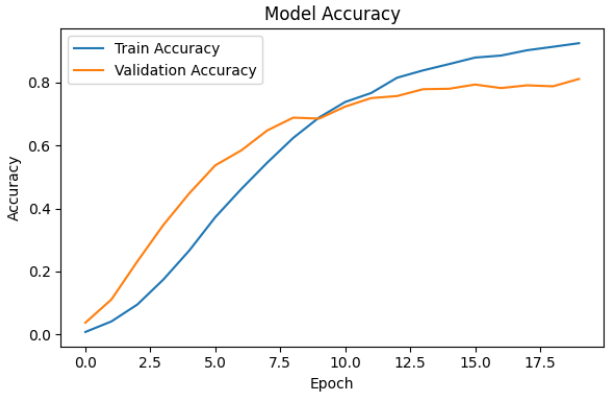
Graphs:

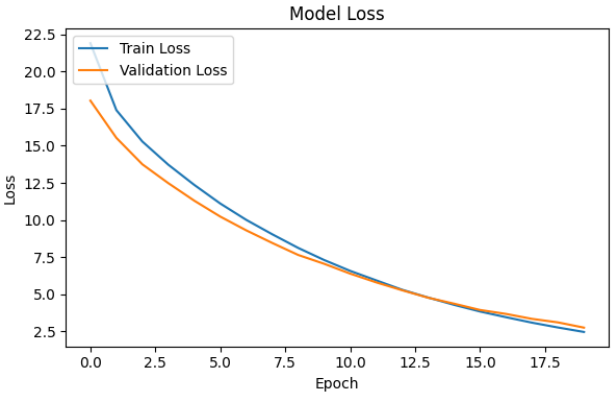
Xception model:



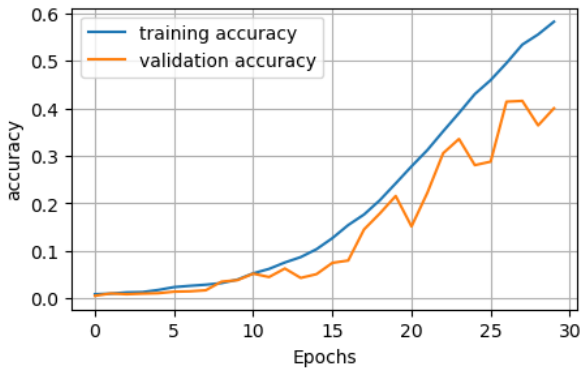


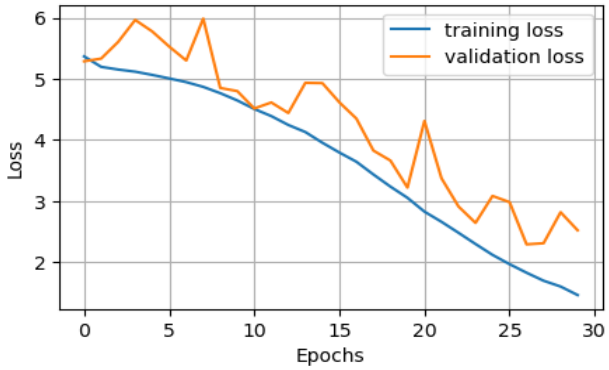
DenseNet model:



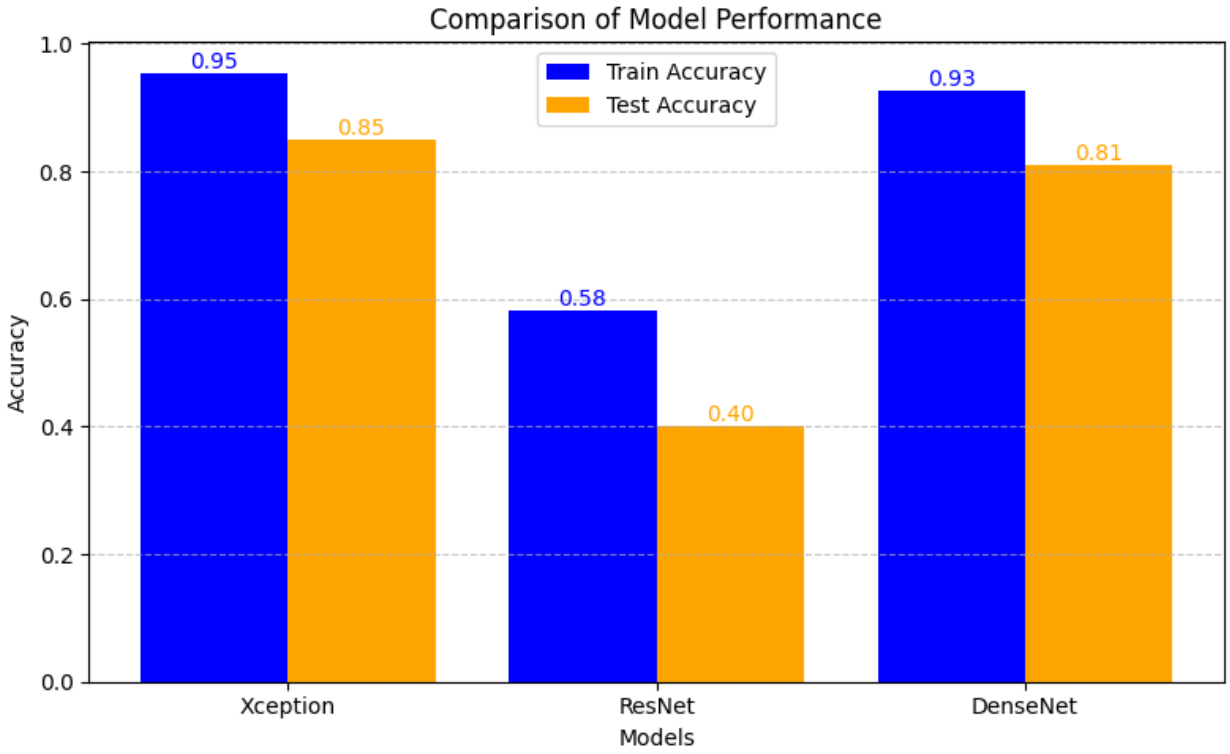


Resnet model:





Comparison Graph :



Which is Better for Car Type Classification?

1. **Small Dataset**:
    a. **DenseNet** is preferable due to its efficient parameter usage and ability to leverage small data effectively.
2. **Moderate Dataset with Computational Constraints**:

    a. **ResNet** (e.g., ResNet-50 or ResNet-101) strikes a balance between performance and efficiency.
3. **Large Dataset with Fine-Grained Details**:
    a. **Xception** is the best choice due to its ability to handle complex, high-dimensional data and extract fine-grained features.

Since we have a large Dataset it was a better choice to pick **Xception** and from the results graphs above we can Cleary see that **Xception** was best fit for our project

Also another reason to choose **Xception** over Resnet and Densenet is ：

High-Resolution Images: If you're working with detailed, high-resolution car images, **Xception** handles them more efficiently than the other two.

Sources:

Xception model: https://maelfabien.github.io/deeplearning/xception/#

Paper:
https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf

DenseNet201 model:

Paper: https://arxiv.org/pdf/1608.06993 , https://arxiv.org/abs/1608.06993v5

https://www.geeksforgeeks.org/densenet-explained

ResNet model: https://medium.com/@ibtedaazeem/understanding-resnet-architecture-a-deep-dive-into-residual-neural-network-2c792e6537a9

Paper:

https://arxiv.org/pdf/1512.03385