# First: ResNet

**ResNet (Residual Networks)** is a deep learning architecture that revolutionized training very deep neural networks by introducing "skip connections," or "residual connections." These connections allow gradients to flow more easily during backpropagation, preventing issues like vanishing gradients in very deep models. The ResNet architecture was introduced in 2015 by Kaiming He et al. and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that year

- **Problem: Vanishing Gradient**

In very deep networks, Gradients often get smaller as they move backward through many layers. making it difficult to train effectively. This happens because the gradients tend to shrink when passed through many layers of activation functions, especially sigmoid or tanh.

- **Residual Connections: The Core Idea**

The main idea behind **ResNet** is the use of **residual block**, which introduces **skip connections** that "skip" one or more layers. Instead of learning the direct mapping (direct transformation) each residual block learns only the difference **(residual)** between the input and the output. Therefore, the network is learning the difference between the desired output and the input, which makes it easier to learn the residual function than the original function. **This addition helps avoid the vanishing gradient problem, making it easier to train very deep networks.**

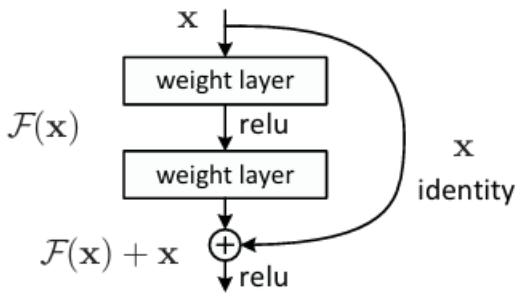


Figure 2. Residual learning: a building block.

## ResNet Building Blocks

- **Basic Block :** It consists of two convolutional layers, each followed by a ReLU activation and batch normalization. The input is added directly to the output through a skip connection. This type of block is used in shallower ResNet architectures, such as ResNet-18 and ResNet-34, where the network depth is relatively smaller, and the basic block design helps maintain efficient training and good performance without introducing the complexity of deeper residual blocks.

**Structure of a basic block:**

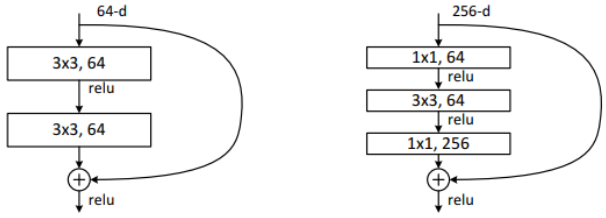
1. Convolution Layer (3x3)
2. Batch Normalization
3. ReLU Activation
4. Convolution Layer (3x3)
5. Batch Normalization
6. Addition of the input to the output (residual connection)

- **Bottleneck Block (for deeper networks):** This is a more efficient version used in deeper networks like ResNet-50, ResNet-101, and ResNet-152. It reduces the number of parameters by applying a 1x1 convolution to reduce the dimensionality before applying the 3x3 convolution. Then, a second 1x1 convolution is applied to restore the dimensionality to the original input size. This structure enables the network to maintain performance while significantly reducing computational cost and memory usage, making it ideal for very deep networks .

**Structure of a bottleneck block:**

1. 1x1 Convolution (to reduce dimensionality)
2. 3x3 Convolution
3. 1x1 Convolution (to restore the dimensionality)
4. Addition of the input to the output (residual connection)

The difference between the Basic block and Bottleneck Block




| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

## How ResNet Works: Key Steps

**Step 1: Input Layer**

- The first step in a ResNet (Residual Network) is the input layer, where the network receives an image (or a set of images). The image can have different dimensions, typically defined by its height, width, and the number of color channels. This layer is where the network begins processing the image, preparing it for the deeper layers.

**Step 2: Initial Convolution Layer**

- The first layer is a standard convolutional layer (with a large kernel size like 7x7) to capture low-level features from the input image, followed by batch normalization and a ReLU activation function. This layer is often followed by a max-pooling layer to downsample the image.

**Step 3: Residual Blocks**

- The input then passes through a series of residual blocks. Each block consists of multiple convolutional layers, batch normalization, ReLU, and skip connections. The skip connection adds the original input to the output of the convolutional layers, forming the residual mapping.

There are typically multiple stages in ResNet where the number of filters increases while the spatial dimensions decrease. These stages are made up of residual blocks.

**Step 4: Global Average Pooling**

- After passing through the residual blocks, a **global average pooling** layer is applied. This operation reduces each feature map to a single number by averaging all its spatial locations. This helps reduce the number of parameters and overfitting.

**Step 5: Fully Connected Layer (Dense Layer)**

- The output from the pooling layer is then passed to a fully connected (dense) layer, which typically has as many neurons as there are classes in the classification task

**Step 6: Softmax Layer (for classification)**

- Finally, a **softmax activation function** is applied in the output layer for multi-class classification, which converts the network's output into probability scores for each class.

## Advantages of ResNet

- **Mitigates Vanishing Gradient:** Residual connections help gradients flow easily during backpropagation, enabling the training of very deep networks without vanishing gradients.
- **Effective in Deep Networks:** ResNet allows training of very deep networks (e.g., ResNet-50, ResNet-101) without performance degradation, unlike traditional deep networks.
- **Better Gradient Flow:** Skip connections allow for better gradient flow, speeding up convergence and improving training stability.
- **Prevents Overfitting (Large Datasets):** Works well on large datasets like ImageNet, with fewer parameters than similar architectures, reducing overfitting.
- **Transfer Learning Friendly:** Pre-trained ResNet models are easily fine-tuned for new tasks, saving time and resources in transfer learning applications.

## Disadvantages of ResNet

- **High Computational Cost (Deeper Models):** Deeper models (e.g., ResNet-101, ResNet-152) are computationally expensive, requiring significant memory and processing power.
- **Overfitting on Small Datasets:** On small datasets, ResNet can overfit due to its large number of parameters and high model complexity.
- **Training Time:** Training very deep ResNet models requires a lot of time and computational resources, even with improved gradient flow.
- **Large Model Size:** Deep ResNet architectures have large model sizes, which can be challenging to deploy in resource-constrained environments.
- **Complexity in Hyperparameter Tuning:** Fine-tuning the hyperparameters (e.g., learning rate, number of layers) is crucial and can be complex for optimal performance.

# Second: DenseNet

**DenseNet (Densely Connected Convolutional Networks)** is a deep neural network architecture introduced by Gao Huang and colleagues in 2017. Unlike traditional Convolutional Neural Networks (CNNs) where each layer receives input only from the previous layer, DenseNet connects each layer to every other layer in a dense manner. This means that every layer gets input from all preceding layers, facilitating efficient feature reuse and improving gradient flow during backpropagation.

**There are several variants of DenseNet depending on how the network is structured and the number of layers like DenseNet-121, DenseNet-169 and DenseNet-201**

## Key concepts in DenseNet

- **Dense Blocks:** DenseNet consists of a series of dense blocks, each of which contains multiple layers. In each block, each layer is connected to every other layer in a feedforward fashion. The output of each layer is passed to all subsequent layers in the block.

- **Transition Layers:** Between dense blocks, DenseNet uses transition layers that perform two operations:
    - **1x1 Convolutions:** To reduce the number of feature maps.
    - **Average Pooling:** To down-sample the feature maps spatially, reducing the spatial resolution.

- **Growth Rate:** The "growth rate" determines how many new feature maps each layer adds to the network. If a dense block has n layers and growth rate k, then the output feature map size at the end of the block is $n \times k$.

## How DenseNet Works: Key Steps

**Step 1: Input Layer**

- The network takes an input image, which is processed through the first convolutional layer, often with a kernel size of 7x7 and stride 2, followed by a batch normalization layer and ReLU activation.

**Step 2: Dense Blocks**

- The input passes through multiple dense blocks. Each dense block contains several convolutional layers:
    - Every layer in a dense block has access to the feature maps from all previous layers.
    - Features are concatenated at each layer to form the input to the next one.

**Step 3: Transition Layers**

- After each dense block, a transition layer reduces the number of feature maps and down-samples the feature maps using:
    - **1x1 Convolution**: To reduce the number of feature maps (compression).
    - **Average Pooling**: To reduce spatial dimensions (down-sampling).

**Step 4: Global Average Pooling**

- After the last dense block, a global average pooling layer is applied, which reduces each feature map to a single value by averaging over its spatial dimensions.

**Step 5: Fully Connected Layer**

- The output of the global average pooling is passed to a fully connected layer, followed by a softmax or sigmoid activation function for classification.

**Step 6: Output Layer**

- The output layer corresponds to the number of classes for classification or the number of units for regression tasks.
-

| Layers | Output Size | DenseNet-121 | DenseNet-169 | DenseNet-201 | DenseNet-264 |
|---|---|---|---|---|---|
| Convolution | 112 × 112 | 7 × 7 conv, stride 2 | | | |
| Pooling | 56 × 56 | 3 × 3 max pool, stride 2 | | | |
| Dense Block (1) | 56 × 56 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$ |
| Transition Layer (1) | 56 × 56 | 1 × 1 conv | | | |
| | 28 × 28 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (2) | 28 × 28 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$ |
| Transition Layer (2) | 28 × 28 | 1 × 1 conv | | | |
| | 14 × 14 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (3) | 14 × 14 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$ |
| Transition Layer (3) | 14 × 14 | 1 × 1 conv | | | |
| | 7 × 7 | 2 × 2 average pool, stride 2 | | | |
| Dense Block (4) | 7 × 7 | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$ | $\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$ |
| Classification Layer | 1 × 1 | 7 × 7 global average pool | | | |
| | | 1000D fully-connected, softmax | | | |

## Advantages of DenseNet

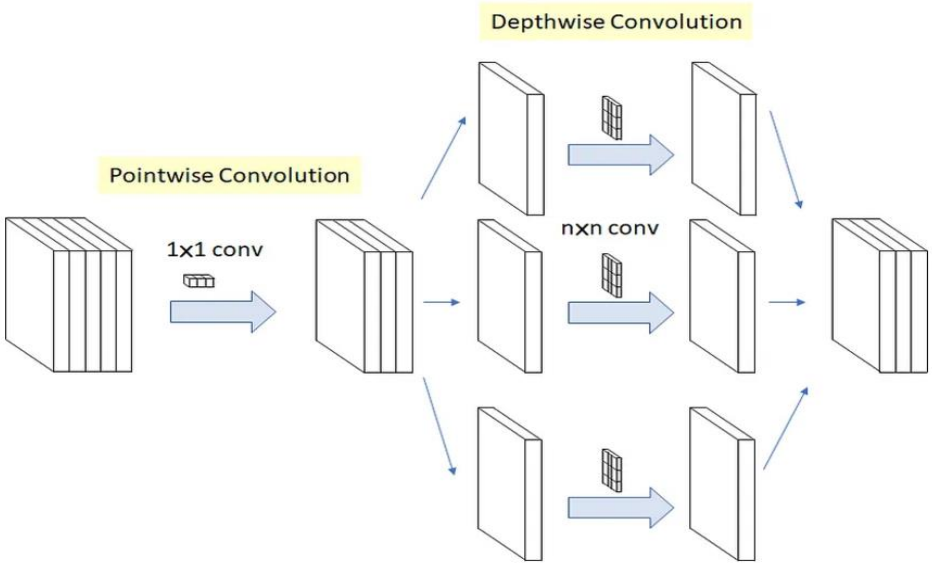
- **Improved Gradient Flow:** Dense connections help gradients flow more effectively during backpropagation, reducing the vanishing gradient problem. This makes training deep networks easier.
- **Feature Reuse:** Each layer gets access to all previous layers' features, promoting feature reuse, which leads to more efficient representation and fewer parameters needed compared to traditional CNNs.
- **Reduced Parameters:** DenseNet's dense connections and feature reuse help reduce the number of parameters compared to traditional deep networks (e.g., ResNet), while maintaining high performance.
- **Better Accuracy:** DenseNet often outperforms traditional CNN architectures (like ResNet) in terms of accuracy, especially on tasks like image classification, due to better feature propagation.
- **Efficient Representation:** By concatenating feature maps from multiple layers, DenseNet allows for a richer, more expressive feature representation.

### Disadvantages of DenseNet

- **Increased Memory Consumption:** The concatenation of all previous feature maps increases memory usage, especially for deep networks. This can be a limitation when working with very large networks or limited hardware.
- **Slower Training:** Due to the dense connections, the computation required for each forward pass and backpropagation can be more expensive, leading to slower training times compared to simpler architectures.
- **Complex Architecture:** The dense connections add complexity to the model structure, which might make it harder to modify or optimize for certain tasks compared to simpler models.
- **Limited Scalability:** While DenseNet is efficient in terms of parameters, its dense connectivity can become a bottleneck as the network grows deeper, making it less scalable in some scenarios compared to architectures like ResNet.
- **Inference Time:** The dense connections can also slow down inference time, as all feature maps from previous layers need to be stored and processed, making it less ideal for real-time applications.

## Third: Xception

The **Xception** architecture is a deep convolutional neural network (CNN) designed for image classification and other computer vision tasks. It was introduced by François Chollet in 2017 as an extension and refinement of the **Inception** architecture. Xception stands for **"Extreme Inception"**, emphasizing that it takes the key concepts of Inception to their extreme limits by using depthwise separable convolutions.



## Key concepts in DenseNet

- **Depthwise Separable Convolutions**
  In a standard convolution, each filter operates on all input. However, in **depthwise separable convolutions**, the convolution is split into two parts:

1. **Depthwise Convolution**: A separate filter is applied to each input channel (each depth slice of the input feature map) individually. This reduces the number of operations significantly.
2. **Pointwise Convolution**: A 1x1 convolution is applied to the outputs of the depthwise convolution, combining them to create new feature maps.

**The depthwise separable convolutions replace the standard convolutions in the model and are believed to be more efficient while maintaining or improving the model's performance.**

- **Entry Flow**
  - The entry flow consists of several **standard convolutions** (with kernel sizes of 3x3 or 5x5) and depthwise separable convolutions to extract basic features from the input image.
  - This part is designed to extract low-level features like edges, textures, and patterns.
- **Middle Flow**
  - The middle flow is composed of **repeated blocks of depthwise separable convolutions**. These blocks apply depthwise separable convolutions multiple times and allow the network to learn hierarchical, abstract representations of the data.
- **Exit Flow**
  - The exit flow is the final part of the architecture, where features from the middle flow are further processed and downsized to the final output, usually by applying global average pooling, followed by a fully connected layer for classification.
  - The exit flow reduces the output dimension and prepares it for the final classification output.

## How DenseNet Works: Key Steps

**Step 1: Input Layer**

- The input to Xception is an image of size 299×299×3. It can handle any RGB image of this size.

**Step 2: Convolutional Layers (Entry Flow)**

- The first step in the entry flow is a 3x3 convolution with strides of 2, followed by batch normalization and ReLU activation. This helps reduce the image size and extract initial features.

**Step 3: Depthwise Separable Convolution Blocks**

- The input undergoes a series of **depthwise separable convolutions**. These blocks reduce the computational cost by decomposing convolutions into depthwise and pointwise operations.

**Step 4: Middle Flow (Repetitive Blocks)**

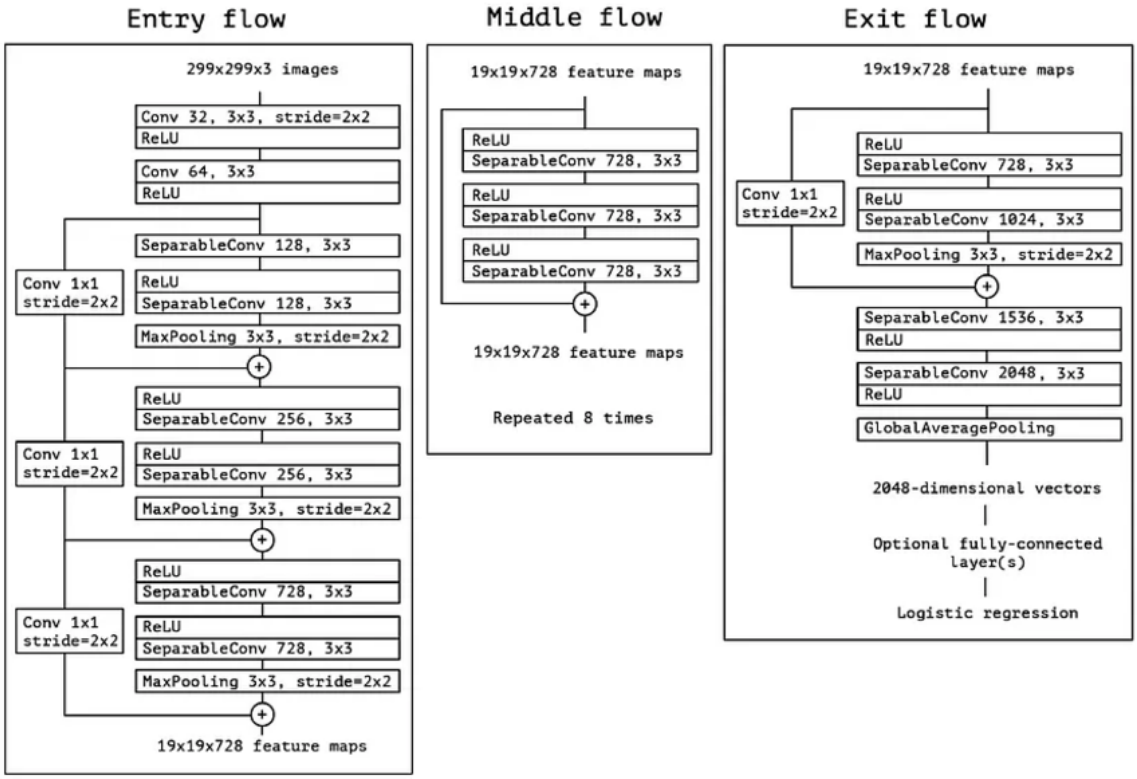
- The middle flow consists of multiple repetitions (usually 8) of **depthwise separable convolution blocks**. These blocks consist of depthwise separable convolutions followed by pointwise convolutions. These layers learn complex features and hierarchical representations from the input.

**Step 5: Exit Flow**

- After the middle flow, the network applies a few more layers of depthwise separable convolutions followed by global average pooling to reduce the dimensions.
- This is followed by a **fully connected layer** that outputs the final class probabilities.

**Step 6: Final Output**:

- The final layer of the network is typically a **softmax** activation function for classification, which outputs a probability distribution over the classes.




Overall Architecture of Xception (Entry Flow > Middle Flow > Exit Flow)

## Advantages of Xception

- **Efficient Use of Depthwise Separable Convolutions**: Xception utilizes depthwise separable convolutions, which split the convolution into two steps (spatial filtering and channel-wise filtering), making it computationally efficient compared to traditional convolutions.

- **High Performance**: Xception achieves state-of-the-art performance on tasks like image classification and object detection, especially when pretrained on large datasets like ImageNet.

- **Flexibility**: It can be fine-tuned on custom datasets for a variety of tasks, making it versatile for many computer vision applications.

- **Improved Training**: The architecture allows for faster convergence during training due to better utilization of model capacity and fewer parameters.

## Disadvantages of Xception

- **Large Model Size**: Xception has a large number of parameters, making the model heavy, which can lead to memory and storage constraints, particularly for deployment on resource-limited devices.
- **Slower Inference Time**: While efficient in terms of computation, Xception might still have slower inference times compared to lightweight models like MobileNet, especially when working with high-resolution inputs.
- **Requires Significant Compute Resources**: Training Xception from scratch requires substantial computational power and time, making it less suitable for environments with limited resources.
- **Not Optimized for Very Small Datasets**: Xception may overfit on smaller datasets if not fine-tuned properly, as its architecture is more suited to large-scale datasets.
- **Complex Architecture**: The model's architecture is more complex compared to simpler models like VGG or ResNet, which might make it harder to understand and debug for beginners.

## Comparison between the three architectures

| Architectures | Pros | cons |
|---|---|---|
| **ResNet** | - Solves vanishing gradient problem with skip connections.<br>- Well-suited for very deep networks.<br>- Works well on a variety of datasets | - Might suffer from overfitting on small datasets.<br>- Requires large computational resources for deeper models. |
| **DenseNet** | - Efficient with fewer parameters, thanks to dense connections.<br>- Superior gradient flow and feature reuse.<br>- Performs well on small datasets. | - Computationally expensive due to the number of connections<br>- Longer training times compared to other architectures. |
| **Xception** | -Highly efficient with depthwiseseparable convolutions.<br>- Excellent performance on medium-to large scale datasets.<br>- Fewer parameters, leading to faster inference times | - More complex architecture than ResNet.<br>- Can be computationally expensive on certain devices. |

## Which is the Best Architecture for Gesture Recognition?

- **ResNet:**

might be the best option if you're working with a very large dataset for gesture recognition. It can handle deeper networks and has established success in image classification tasks. However, if the dataset is smaller, there's a risk of overfitting.

- **Xception:**

offers a great balance of performance and efficiency. For real-time applications like gesture recognition, where computational efficiency is key, Xception would likely be an excellent choice due to its use of separable convolutions and reduced parameter count.

- **DenseNet:**

would be the best choice if the dataset is smaller and the focus is on improving feature reuse and gradient flow. However, the architecture is computationally more expensive, andmight be slower for real-time applications.

**Conclusion: For gesture classification, Xception is likely the best choice due to its efficient design and strong performance. However, for smaller datasets, DenseNet could be a strong contender if the computational cost is manageable.**