ResNet

ResNet(residual network) relies on residual connections (shortcuts) that allow the input of a layer to be added directly to its output, ensuring the network learns residual mappings instead of direct mappings. This helps in training deeper networks by mitigating the vanishing gradient problem.

This ResNet architecture consist of:

1. Initial Convolution and Pooling:
   - 7×7 Convolution with 64 filters and a stride of 2 (output: 64 feature maps).
   - Batch Normalization (BN) and ReLU activation are applied to normalize the output and introduce non-linearity.
   - 3×3 Max Pooling with stride 2 reduces the spatial dimensions of the feature maps (output size: reduced spatial dimension with 64 channels).
2. Residual Block 1:
   - The first residual block contains two layers.
   - Each layer has a combination of:
     - 1×1 Convolution: This operation reduces the number of feature channels.
     - 3×3 Convolution: This extracts features from the image, like edges or patterns.
   - The output of this residual block is the addition of the shortcut connection (identity mapping) and the output of the convolutions, allowing the model to preserve information from earlier layers.
   - The output channels increase with each residual module, and the number of feature maps grows as the layers stack.
3. Transition to Next Residual Blocks (stages):

   After the first residual block, the model proceeds to the next stage, where each stage contains multiple residual blocks. The stride is set to (2, 2) in the first residual block of each stage to reduce spatial dimensions (downsampling), while the stride is (1, 1) for subsequent residual blocks within the same stage.
4. Residual Block 2 (Stage 1):
   - Similar to the first block, each residual module in this stage has convolutions (1×1 and 3×3) but with a higher number of feature maps.

- This stage typically contains multiple residual modules (you have 2 residual blocks in this stage).
- The residual connections ensure that the features from the earlier layers are reused to help prevent performance degradation.

5. Residual Block 3 (Stage 2):

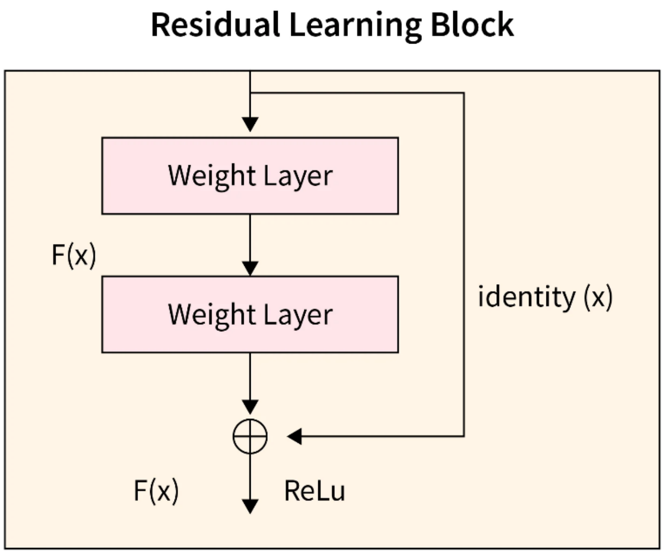
- As you progress through more stages, each block adds more complexity with an increasing number of filters (e.g., filters could be 32, 64, 128, etc.).
- This stage might have 2 residual modules, each designed to learn more complex features as the network deepens.
- Output channels increase, and the network learns richer features.

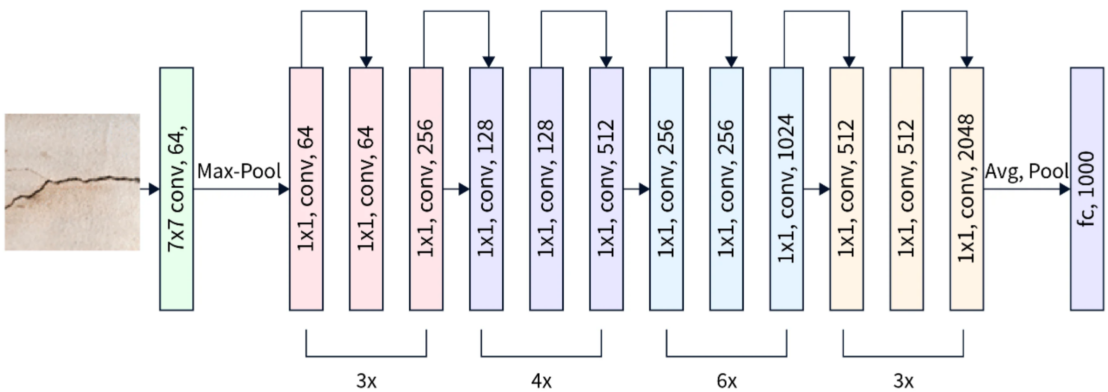
6. Residual Block 4 (Stage 3):

- This stage continues the residual pattern. The layers get deeper and more sophisticated as the output channels increase, refining the network's learned features.
- The deeper layers in the residual network focus on capturing even more abstract and complex features of the input data.

7. Final Layer and Classification:

- After all the residual blocks, the output is passed through Batch Normalization followed by ReLU activation.
- Global Average Pooling is applied to reduce the spatial dimensions to a single vector (summarizing the entire feature map).
- The final output is passed through a Fully Connected Layer (Dense), followed by softmax activation to obtain class probabilities for classification.





Reference:

https://www.researchgate.net/publication/346212393_Image_classification_based_on_RESNET

https://www.scaler.com/topics/residual-networks-resnet-deep-learning/

DenseNet

connects each layer to every other layer within a dense block, ensuring each layer receives inputs from all preceding layers and passes its output to subsequent layers. This dense connectivity promotes extensive information flow and feature reuse.

DenseNet-121 consists of 121 layers and follows the configuration of four dense blocks interspersed with three transition layers.

1. Initial Convolution and Pooling:
   7×7 Convolution with 64 filters, stride 2 (output: 64 feature maps).
   Batch Normalization (BN) and ReLU activation.
   3×3 Max Pooling with stride 2 (reduces spatial dimensions).

2. Dense Block 1:
   6 layers (each performing 1×1 and 3×3 convolutions).
   Growth Rate: k = 32 (each layer adds 32 feature maps).
   Output channels: 64 + 6 × 32 = 256 channels.

3. Transition Layer 1:
   1×1 Convolution (reduces channels).
   2×2 Average Pooling with stride 2 (downsampling).

4. Dense Block 2:
   12 layers (each adding 32 feature maps).
   Output channels: 128 + 12 × 32 = 512 channels.

5. Transition Layer 2:
   1×1 Convolution and 2×2 Average Pooling (downsampling).

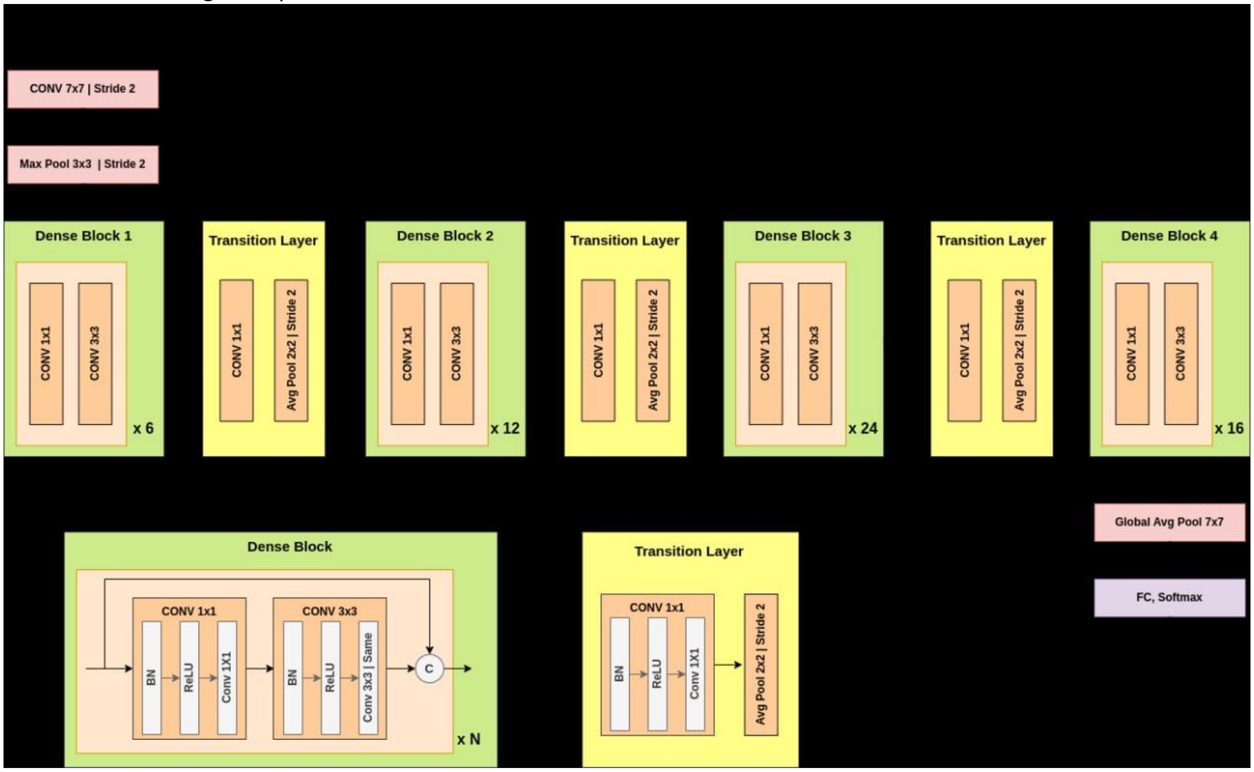
6. Dense Block 3:
   24 layers (each adding 32 feature maps).
   Output channels: 256 + 24 × 32 = 1024 channels.

7. Transition Layer 3:
   1×1 Convolution and 2×2 Average Pooling (downsampling).

8. Dense Block 4:
   16 layers (each adding 32 feature maps).
   Output channels: 512 + 16 × 32 = 1024 channels.

9. Classification Layer:
   Global Average Pooling (reduces feature maps to a single vector).

Fully Connected (Dense) Layer with output size depending on the task (e.g., 1000 classes for ImageNet).



Reference:

https://iq.opengenus.org/architecture-of-densenet121/

https://arxiv.org/abs/1608.06993

Xception

The Inception module divides the task of spatial and cross-channel correlation using filters of different sizes (1×1, 3×3, 5×5) in parallel, The Xception model refines the Inception idea by aggressively separating spatial and cross-channel processing. This gave it the name Extreme Inception Model.

The Three Parts of Xception Architecture
We divide the entire Xception architecture into three main parts: the entry flow, the middle flow, and the exit flow, with skip connections around the 36 layers.

Entry Flow :

The input image is 299×299 pixels with 3 channels (RGB).

A 3×3 convolution layer is used with 32 filters and a stride of 2×2. This reduces the image size and extracts low-level features. To introduce non-linearity, the ReLU activation function is applied.

It is followed by another 3×3 convolution layer with 64 filters and ReLU.

After the initial low-level feature extraction, the modified depthwise separable convolution layer is applied, along with the 1×1 convolution layer. Max pooling (3×3 with stride=2) reduces the size of the feature map.

Middle Flow :

This block is repeated eight times.

Each repetition consists of:

Depthwise separable convolution with 728 filters and a 3×3 kernel.

ReLU activation.

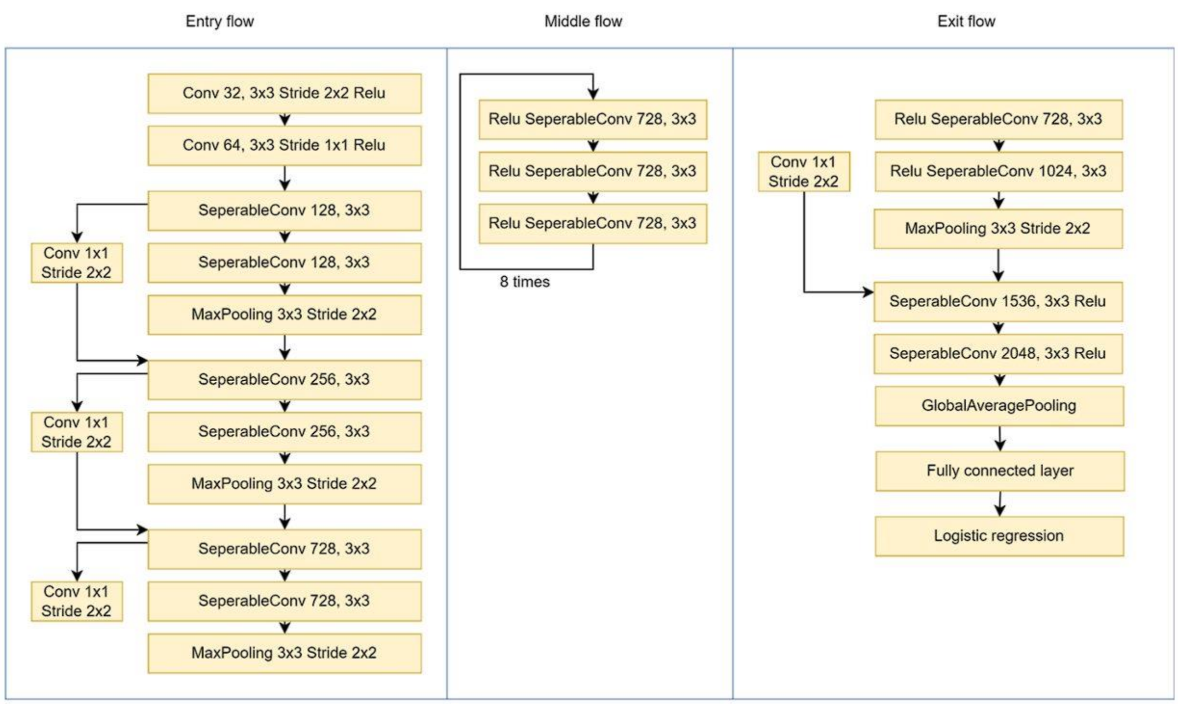
By repeating it eight times, the middle flow progressively extracts higher-level features from the image.

Exit Flow :

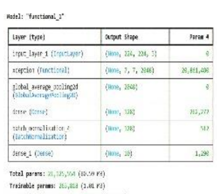
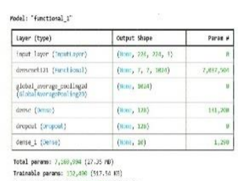
Separable convolution with 728, 1024, 1536, and 2048 filters, all with 3×3 kernel further extracts complex features.

Global Average Pooling summarizes the entire feature maps into a single vector.

Finally, at the end, a fully connected layer with logistic regression classifies the images.



https://viso.ai/deep-learning/xception-model/

<table>
<tr><th></th><th>xception</th><th>densenet</th><th>resnet</th></tr>
<tr><td></td><td>xception</td><td>densenet</td><td>resnet</td></tr>
<tr><td>train_accuracy</td><td>0.8092</td><td>0.7472</td><td>0.5998</td></tr>
<tr><td>val_accuracy</td><td>0.7283</td><td>0.7309</td><td>0.5690</td></tr>
<tr><td>Test Accuracy</td><td>72.91%</td><td>73.84%</td><td>56%</td></tr>
<tr><td>loss</td><td>0.5396</td><td>0.7191</td><td>1.2308</td></tr>
<tr><td>val_loss</td><td>0.8303</td><td>0.7692</td><td>1.3238</td></tr>
<tr><td>Classfication Report</td><td>precision recall f1-score support
abstract_painting 0.83 0.87 0.85 695
cityscape 0.70 0.65 0.67 639
genre_painting 0.60 0.60 0.60 1486
illustration 0.48 0.52 0.50 264
landscape 0.81 0.83 0.82 1850
nude_painting 0.56 0.53 0.55 268
portrait 0.81 0.81 0.81 1963
religious_painting 0.70 0.68 0.69 912
sketch_and_study 0.62 0.62 0.62 550
still_life 0.78 0.77 0.77 389
accuracy 0.73 9016
macro avg 0.69 0.69 0.69 9016
weighted avg 0.73 0.73 0.73 9016</td><td>precision recall f1-score support
abstract_painting 0.87 0.86 0.86 696
cityscape 0.73 0.63 0.68 642
genre_painting 0.60 0.57 0.59 1483
illustration 0.47 0.55 0.51 264
landscape 0.80 0.87 0.84 1846
nude_painting 0.67 0.50 0.57 269
portrait 0.79 0.84 0.81 1964
religious_painting 0.72 0.71 0.71 910
sketch_and_study 0.65 0.60 0.62 551
still_life 0.80 0.78 0.79 389
accuracy 0.74 9014
macro avg 0.71 0.69 0.70 9014
weighted avg 0.74 0.74 0.74 9014</td><td>precision recall f1-score support
0 0.67 0.69 0.68 695
1 0.41 0.69 0.52 639
2 0.49 0.26 0.34 1486
3 0.28 0.63 0.39 264
4 0.79 0.57 0.66 1850
5 0.59 0.13 0.21 268
6 0.64 0.79 0.71 1963
7 0.46 0.54 0.50 912
8 0.47 0.59 0.52 550
9 0.55 0.33 0.42 389
accuracy 0.56 9016
macro avg 0.54 0.52 0.49 9016
weighted avg 0.59 0.56 0.55 9016</td></tr>
<tr><td>Model</td><td></td><td></td><td></td></tr>
<tr><td>pros</td><td></td><td></td><td></td></tr>
</table>

Models	Pros	Cons
models	**pros**	**cons**
ResNet	1. Prevents Vanishing Gradients: Skip connections make training deep networks easier. 2. Enables Deep Networks: Can train models with hundreds of layers. 3. Improved Accuracy: Performs well on image classification tasks. 4. Efficient Training: Easier to train compared to traditional deep networks. 5. Great for Transfer Learning: Pretrained models are widely used.	1. High Computational Cost: Deeper networks require more memory and time. 2. Diminishing Returns: Adding more layers doesn't always improve performance. 3. Risk of Overfitting: Can overfit on small datasets. 4. Inference Latency: Slower for real-time applications.
Xception	1. Efficient: Uses depthwise separable convolutions, which reduce computational complexity. 2. High Accuracy: Outperforms traditional CNNs on tasks like image classification. 3. Fewer Parameters: More efficient in terms of parameters compared to similar architectures. 4. Better Feature Extraction: Handles spatial and channel-wise features effectively.	1. Computationally Heavy: Still requires significant computational resources. 2. Slower Inference: Depthwise separable convolutions can slow down inference speed. 3. Data Hungry: Can overfit on smaller datasets and needs more data for good performance. 4. Complex to Implement: Harder to understand and implement than simpler CNN architectures.

DenseNet-121	1. Improved Feature Utilization: Each layer receives input from all previous layers, leading to better feature reuse. 2. Fewer Parameters: Despite being deep, it requires fewer parameters than traditional CNNs due to efficient feature reuse. 3. Reduced Vanishing Gradient Problem: Dense connections improve gradient flow and facilitate the training of very deep networks. 4. Strong Performance: Achieves high accuracy on various image classification tasks.	1. Memory Intensive: The dense connections increase memory usage, especially for large models. 2. Slower Training: Due to the dense connections and large feature maps, training can be slower compared to other architectures like ResNet. 3. Inference Latency: Dense networks may have slower inference times due to the larger number of feature maps and connections. 4. Complex Architecture: The design is more complex and can be harder to implement and optimize.

The **3** architectures each have distinct advantages on the **Wikiart** dataset:

- **DenseNet121**: Works well with large datasets like Wiki because it efficiently reuses features, allowing faster training and better generalization with fewer parameters.
- **ResNet**: Excellent for large, diverse datasets like Wiki due to its deep architecture and residual connections, which prevent performance degradation.
- **Xception**: Excellent for complex image data and high classification accuracy with efficient computation.

But the architecture that fits well with our task and dataset is the **DenseNet121** model, achieving a **73.84% accuracy** on the test data, highlighting its strong performance and suitability for this task because it efficiently reuses features, allowing faster training and better generalization with fewer parameters.