Project Description Document

Overview about the project: **Classify medical images, such as X-rays to detect diseases like pneumonia.**

Dataset used: **Chest X-ray Dataset.**

Found 5216 images belonging to 2 classes [NORMAL] & [PNEUMONIA] for train.

Found 624 images belonging to 2 classes [NORMAL] & [PNEUMONIA] for test.

MODEL 1: [ResNet Implemented from scratch]:

Implementation Details

Feature Extraction Phase

The feature extraction phase leverages convolutional and residual blocks to learn and encode features from the input medical X-ray images. The layers progressively extract features of increasing complexity, moving from low-level features like edges to high-level features such as patterns indicative of Normal or Pneumonia classes.

Number of Features Extracted:

- The number of features extracted at each stage depends on the filter dimensions used in the convolutional and residual blocks. At the final feature map level, **2048 features** are extracted before global average pooling.
- patterns learned from the data in the context of the X-ray dataset, the extracted features could represent:
- Texture patterns
- Edges and boundaries
- Anomalies and regions of interest (e.g., areas indicative of Pneumonia)

Blocks Used in the Model

The ResNet model implements **four stages** of blocks (Convolutional and Identity), where each stage learns features at increasing levels of abstraction. Below are the details:

1. **Initial Convolutional Layer:**
   - **Filters**: 64
   - **Kernel Size**: (7, 7)
   - **Strides**: (2, 2)
   - **Output Features**: 64 channels, downsampled spatially using a 3×3 max pooling layer.

2. **Residual Blocks Stages:**
    - **Stage 1**:
        - **Block Type**: 1 Convolutional Block + 2 Identity Blocks
        - **Filters**: (64, 256)
        - **Output Features**: 256 channels
        - **Purpose**: Extract basic features like edges and corners.
    - **Stage 2**:
        - **Block Type**: 1 Convolutional Block + 3 Identity Blocks
        - **Filters**: (128, 512)
        - **Output Features**: 512 channels
        - **Purpose**: Learn intermediate patterns like shapes or clusters.
    - **Stage 3**:
        - **Block Type**: 1 Convolutional Block + 2 Identity Blocks
        - **Filters**: (256, 1024)
        - **Output Features**: 1024 channels
        - **Purpose**: Capture complex and abstract patterns.
    - **Stage 4**:
        - **Block Type**: 1 Convolutional Block + 2 Identity Blocks
        - **Filters**: (512, 2048)
        - **Output Features**: 2048 channels
        - **Purpose**: Capture very high-level features related to abnormalities in the X-ray images.

3. **Classification Head:**
    - **Global Average Pooling**: Reduces the 2048-channel feature map to a 1D vector of 2048 features.
    - **Fully Connected Layer**:
        - 1024 units with ReLU activation and dropout.
    - **Output Layer**: 1 unit with sigmoid activation (for binary classification).

Skip Connections:

- **Purpose**: Address the vanishing gradient problem by adding residual connections.
- **Details**: These connections skip one or more layers, allowing the model to learn identity mappings, which makes the optimization easier.

Summary of Blocks

Stage	Block Type	Filters	Output Features	Number of Blocks
Initial	Conv Layer + Pool	(7x7, 64 filters)	64	1
Stage 1	Conv Block + ID	(64, 256)	256	1 Conv + 2 ID
Stage 2	Conv Block + ID	(128, 512)	512	1 Conv + 3 ID
Stage 3	Conv Block + ID	(256, 1024)	1024	1 Conv + 2 ID
Stage 4	Conv Block + ID	(512, 2048)	2048	1 Conv + 2 ID

ResNet Architecture.

Refrence[https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8]

Preprocessing Steps

1- Rescaling

What's Done ?

- All pixel values are scaled to the range [0, 1] by dividing by 255.
- Applied using rescale = 1./255 in the ImageDataGenerator.

Why ?

Rescaling normalizes the input data, ensuring faster convergence during training and preventing numerical instabilities.

2- Data Augmentation

What's Done ?

Data augmentation is applied only to the training dataset to artificially increase its size and diversity. This reduces overfitting and improves generalization.

Augmentation Techniques:

- **Rotation Range:** Small rotations up to 10° to simulate slight variations in image orientation.
- **Width and Height Shifts:** Random translations up to 10% of the image dimensions to simulate spatial displacement.
- **Horizontal Flipping:** Random horizontal flipping of images to account for mirrored anatomical structures.
- **Fill Mode:** nearest is used to fill in pixels when transformations create empty areas.

Why ?

To expose the model to varied versions of the same data, improving robustness to real-world variations.

Encourages the model to learn features invariant to transformations.

3- Class Balancing

What's Done ?

- compute_class_weight is used to calculate class weights for the imbalanced dataset.
- Class weights are applied during training to penalize misclassification of minority classes.

Why ?

Medical datasets often have class imbalances (e.g., more Pneumonia cases than Normal), leading to biased predictions.
Ensures fair learning for all classes.

4- Input Image Dimensions

What's Done ?

- All images are resized to (128, 128) during data loading..

Why ?

Uniform input size is required for deep learning models.
Smaller dimensions reduce computational load while preserving enough detail for classification.
Reduces training time and memory usage.
Ensures compatibility with the ResNet model architecture.

Primary disadvantage for this dimensions size is minor loss of detail, especially in high-resolution images.

5- Shuffling

What's Done ?

- Shuffling is applied to training batches but disabled for test data to ensure label order is maintained during evaluation.

Why ?

- Prevents overfitting by ensuring that the model doesn't memorize the sequence of training data.
- Maintains the integrity of evaluation metrics for the test dataset.

Callback Mechanisms

The model uses several callbacks to enhance training efficiency and avoid overfitting.

EarlyStopping:

What is done?

- Monitors validation loss and stops training when no improvement is observed for 5 epochs.

- **Pros:** Saves training time, prevents overfitting.

- **Cons:** May stop training prematurely if the model needs more time to converge.

ModelCheckpoint:

What is done?

- Saves the model with the best validation loss during training.
- **Pros:** Retains the best-performing model.
- **Cons:** Additional disk space usage.

ReduceLROnPlateau:

What is done?

- Reduces the learning rate by a factor of 0.5 if validation loss plateaus for 3 epochs.

- **Pros:** Helps escape local minima; improves convergence.
- **Cons:** Increases training time slightly due to slower learning.

Loss Function

Binary Crossentropy:

- Chosen because it is well-suited for binary classification tasks.
- **Why?** Maximizes the log probability of correct predictions.

Optimizer

- **Adam Optimizer with Learning Rate of 1e-4:**
  - Adaptive learning rate optimization algorithm.
  - Combines the benefits of RMSProp and Momentum optimizers.
- **Pros:**
  - Automatically adjusts learning rates.
  - Requires less tuning compared to SGD.
- **Cons:**
  - Slightly more memory-intensive.

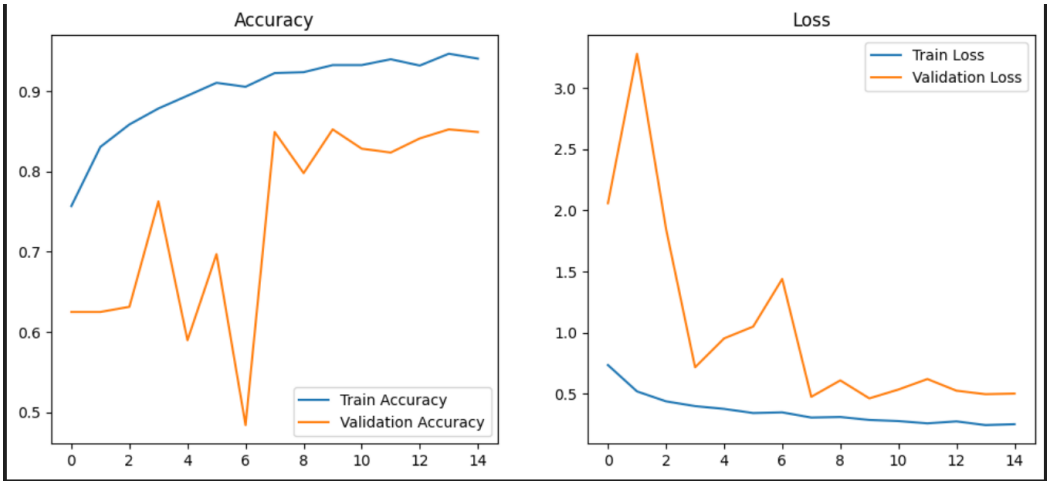
Evaluation Metrics

- **Accuracy:**
  - Measures overall correctness of predictions.
- **Confusion Matrix:**
  - Provides insights into class-wise performance.
- **Precision, Recall, F1-Score:**
  - Precision highlights false positives, recall emphasizes false negatives, and F1-score balances both.
- **ROC-AUC:**
  - Evaluates the model's ability to distinguish between classes at varying thresholds.

Additional Considerations

1. **Regularization:**
    - L2 regularization is applied to convolutional and dense layers to prevent overfitting.
    - Dropout with a rate of 0.5 is used in the dense layer to further regularize the model.
2. **Training Epochs:**
    - Set to 15 with early stopping to avoid overfitting.
3. **Batch Size:**
    - Fixed at 32 for both training and testing datasets.

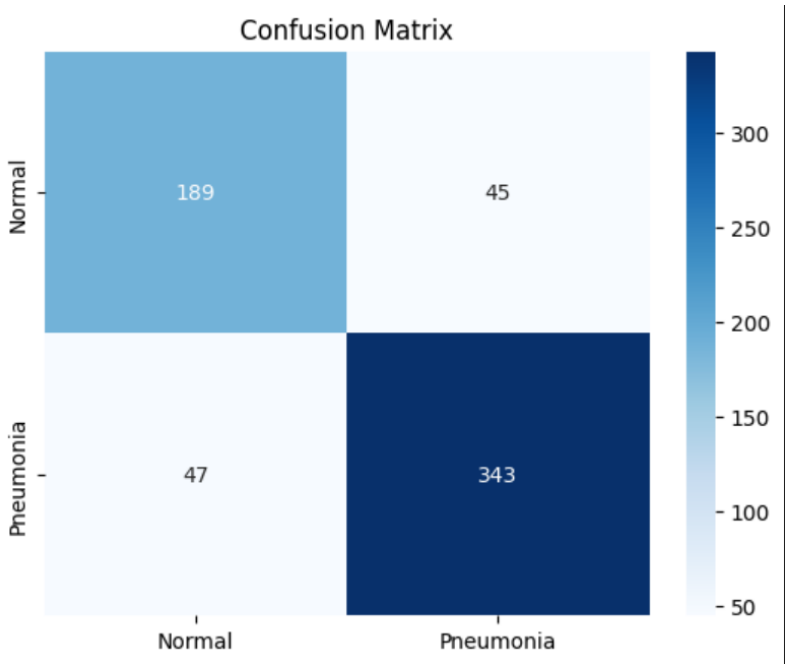
- ## Accuracies




```
- accuracy: 0.9422 - loss: 0.2436 - val_accuracy: 0.8494 - val_loss: 0.5013
```

```
Test Loss: 0.4622
Test Accuracy: 0.8526
```

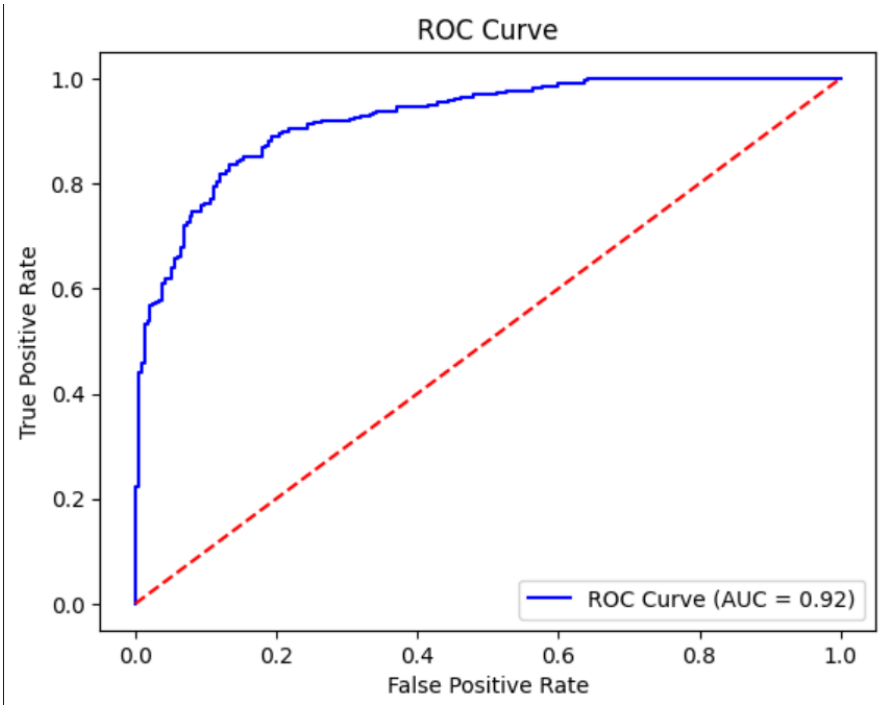
- **Confusion Matrix**



- **Precision, Recall, F1-Score:**

	precision	recall	f1-score	support
NORMAL	0.80	0.81	0.80	234
PNEUMONIA	0.88	0.88	0.88	390
accuracy			0.85	624
macro avg	0.84	0.84	0.84	624
weighted avg	0.85	0.85	0.85	624

- **ROC-AUC:**




```
AUC Score: 0.9233
```

MODEL 2: [DenseNet121 pre-trained]:

Implementation Details

Feature Extraction Phase

The feature extraction phase in **DenseNet121** utilizes **Dense Blocks** and **Transition Layers** to efficiently learn hierarchical and spatial patterns in the medical X-ray images. DenseNet uniquely connects each layer to every other subsequent layer, ensuring feature reuse and gradient flow. This architecture is particularly effective for medical imaging tasks like detecting anomalies.

Number of Features Extracted

- **DenseNet121** extracts **2048 features** at the final feature map level before the **Global Average Pooling** layer.
- These features encode critical visual patterns such as:
  - **Texture patterns**
  - **Edges and boundaries**
  - **Regions of interest indicative of Normal or Pneumonia classes**

Blocks Used in the Model

DenseNet121 comprises four Dense Blocks, each connected via Transition Layers. Here's a detailed breakdown:

1. **Initial Convolutional Layer:**
   - **Filters:** 64
   - **Kernel Size:** (7, 7)
   - **Stride:** (2, 2)
   - **Output:** Downsampled spatial dimensions followed by a **3×3 Max Pooling** layer.

2. **Dense Block 1:**
    - **Number of Layers:** 6
    - **Growth Rate:** 32
    - **Output Features:** 256
    - **Purpose:** Extract basic features like edges and corners.
3. **Transition Layer 1:**
    - Includes a **1×1 Convolutional Layer** followed by **2×2 Average Pooling** for downsampling.
4. **Dense Block 2:**
    - **Number of Layers:** 12
    - **Growth Rate:** 32
    - **Output Features:** 512
    - **Purpose:** Learn intermediate patterns like clusters or regions indicative of pneumonia.
5. **Transition Layer 2:**
    - Similar to Transition Layer 1, reduces dimensions to prevent feature map explosion.
6. **Dense Block 3:**
    - **Number of Layers:** 24
    - **Growth Rate:** 32
    - **Output Features:** 1024
    - **Purpose:** Extract more complex and abstract features.
7. **Transition Layer 3:**
    - Same downsampling as the previous transition layers.
8. **Dense Block 4:**
    - **Number of Layers:** 16
    - **Growth Rate:** 32
    - **Output Features:** 2048
    - **Purpose:** Capture high-level representations crucial for differentiating between Normal and Pneumonia classes.
9. **Classification Head:**
    - **Global Average Pooling:** Reduces 2048 features to a 1D vector.
    - **Fully Connected Layer:**
        - 1024 units with **ReLU activation** and **Dropout (rate=0.5)**.
    - **Output Layer:** 1 unit with **sigmoid activation** for binary classification.

Preprocessing Steps

1. Rescaling

- **What's Done?**
  All pixel values are normalized to the range [0, 1] by dividing by 255 using rescale=1./255.
- **Why?**
  Normalization ensures faster convergence and prevents numerical instability during training.

2. Data Augmentation

- **What's Done?**
  Augmentation is applied to the training dataset to improve generalization and reduce overfitting. Techniques include:
  - **Rotation Range:** Up to 10° to simulate slight image orientation variations.
  - **Width and Height Shifts:** Up to 10% to simulate spatial displacements.
  - **Brightness Adjustment:** Brightness variations within the range [0.9, 1.1].
  - **Zoom Range:** Random zooming up to 10%.
  - **Horizontal Flip:** To account for mirrored anatomical structures.
  - **Fill Mode:** Pixels created during transformations are filled using the nearest neighbor method.
- **Why?**
  Encourages the model to learn robust features invariant to transformations and increases dataset diversity.

3. Class Balancing

- **What's Done?**
  - Class weights are calculated to address class imbalances using the formula:

$$\text{Class Weight for Class } i = \frac{\text{Total Samples}}{2 \times \text{Number of Samples in Class } i}$$

- **Why?**
  Ensures that both classes (Normal and Pneumonia) are equally prioritized during training.

4. Input Image Dimensions

- **What's Done?**
  All images are resized to **224×224**.
- **Why?**
  DenseNet121 requires fixed input dimensions, and resizing standardizes the input while maintaining sufficient detail.

5. Shuffling

- **What's Done?**
  - Enabled for training data.
  - Disabled for test data (ensures label order for consistent evaluation).
- **Why?**
  Shuffling prevents overfitting by exposing the model to varied batches and maintains label integrity for testing.

Callback Mechanisms

1. EarlyStopping

- **What's Done?**
  Monitors validation loss, stops training if it doesn't improve for 5 consecutive epochs.
- **Pros:** Prevents overfitting and saves computational resources.
- **Cons:** May stop prematurely if patience is set too low.

2. ModelCheckpoint

- **What's Done?**
  Saves the model with the lowest validation loss during training.
- **Pros:** Retains the best model.
- **Cons:** Requires additional disk space.

3. ReduceLROnPlateau

- **What's Done?**
  Reduces the learning rate by half if validation loss stagnates for 3 epochs.
- **Pros:** Helps the optimizer escape plateaus.
- **Cons:** Slightly increases training time.

Model Hyperparameters

1. **Loss Function:**
   - Binary Crossentropy: Suitable for binary classification tasks.
2. **Optimizer:**
   - Adam Optimizer with a learning rate of **1e-4**:
     - Combines Momentum and RMSProp optimizers for adaptive learning.
3. **Batch Size:**
   - Set to **32** for both training and testing.
4. **Epochs:**
   - Maximum of **15** with early stopping to avoid overfitting.

Evaluation Metrics

1. **Accuracy:**
   Measures the overall correctness of predictions.
2. **Confusion Matrix:**
   Provides insights into class-wise performance, e.g., true positives and false negatives.
3. **Precision, Recall, F1-Score:**
   - **Precision:** Highlights false positives.
   - **Recall:** Emphasizes false negatives.
   - **F1-Score:** Balances precision and recall.
4. **ROC-AUC:**
   - Evaluates the model's ability to distinguish between classes at varying thresholds.

Additional Considerations

1. **Regularization:**
   - **Dropout:** Rate of 0.5 applied to the dense layer.
   - Prevents overfitting.
2. **Fine-Tuning:**
   - Only the last **10 layers** of DenseNet121 are trainable to adapt pretrained features to the dataset.
3. **Hardware Requirements:**
   - DenseNet121 requires a GPU for efficient training due to its dense connectivity and computational demands.

What is DenseNet?

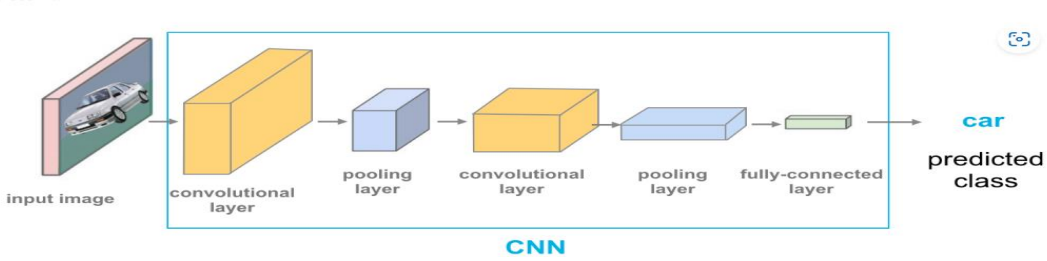
DenseNet, short for Dense Convolutional Network, is a deep learning architecture for convolutional neural networks (CNNs) introduced by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger in their paper titled "Densely Connected Convolutional Networks" published in 2017. DenseNet 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.

Refrence: [https://www.geeksforgeeks.org/densenet-explained/]

In a standard **Convolutional Neural Network**, we have an input image, that is then passed through the network to get an output predicted label in a way where the forward pass is pretty straightforward as shown in the image below:



Each convolutional layer except the first one (which takes in the input image), takes in the output of the previous convolutional layer and produces an output

feature map that is then passed to next convolutional layer. For `L` layers, there are `L` direct connections - one between each layer and its subsequent layer.

The **DenseNet** architecture is all about modifying this standard CNN architecture.

In a **DenseNet** architecture, each layer is connected to every other layer, hence the name **Densely Connected Convolutional Network**. For `L` layers, there are `L(L+1)/2` direct connections. For each layer, the feature maps of all the preceding layers are used as inputs, and its own feature maps are used as input for each subsequent layers.

This is really it, as simple as this may sound, DenseNets essentially conect every layer to every other layer. This is the main idea that is extremely powerful. The input of a layer inside **DenseNet** is the concatenation of feature maps from previous layers.

From the paper: > DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

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	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

DenseNet Architecture.

Refrence[https://amaarora.github.io/posts/2020-08-02-densenets.html]

Each architecture consists of four DenseBlocks with varying number of layers. For example, the DenseNet-121 has [6,12,24,16] layers in the four dense blocks whereas DenseNet-169 has [6, 12, 32, 32] layers.

We can see that the first part of the DenseNet architecture consists of a 7x7 stride 2 Conv Layer followed by a 3x3 stride-2 MaxPooling layer. And the fourth dense block is followed by a **Classification Layer** that accepts the feature maps of all layers of the network to perform the classification.

Also, the convolution operations inside each of the architectures are the Bottle Neck layers. What this means is that the 1x1 conv reduces the number of channels in the input and 3x3 conv performs the convolution operation on the transformed version of the input with reduced number of channels rather than the input.

Bottleneck Layers
By now, we know that each layer produces K feature maps which are then concatenated to previous feature maps. Therefore, the number of inputs are quite high especially for later layers in the network.

This has huge computational requirements and to make it more efficient, the authors decided to utilize Bottleneck layers. From the paper: > 1×1 convolution can be introduced as bottleneck layer before each 3×3 convolution to reduce the number of input feature-maps, and thus to improve computational efficiency. In our experiments, we let each 1×1 convolution produce **4k** feature-maps.

We know K refers to the growth rate, so what the authors have finalized on is for 1x1 conv to first produce 4*K feature maps and then perform 3x3 conv on these 4*k size feature maps.

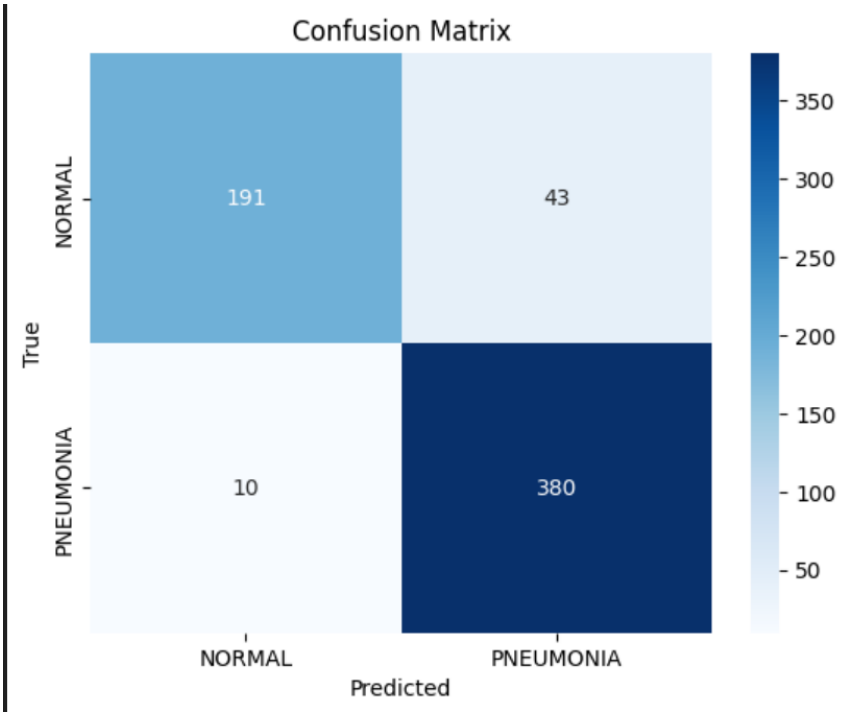
Refrence:[https://amaarora.github.io/posts/2020-08-02-densenets.html]

Accuracies

```
accuracy: 0.9794 - loss: 0.0561 - val_accuracy: 0.9263 - val_loss: 0.2686
```

```
Test Accuracy: 0.9151
```

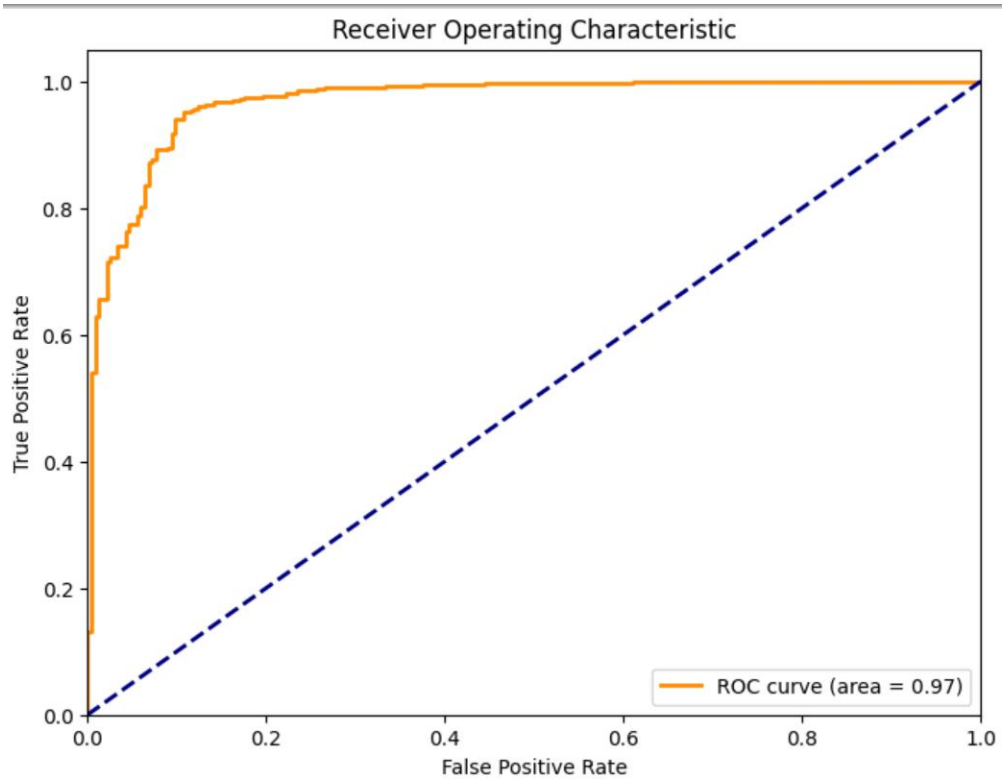
Confusion Matrix



Precision, Recall, F1-Score

	precision	recall	f1-score	support
0	0.95	0.82	0.88	234
1	0.90	0.97	0.93	390
accuracy			0.92	624
macro avg	0.92	0.90	0.91	624
weighted avg	0.92	0.92	0.91	624

ROC-AUC




```
AUC Score: 0.9687
```

MODEL 3: [Xception pretraiend]:

Implementation Details

Feature Extraction Phase

The Xception model employs **depthwise separable convolutions** for feature extraction. These convolutions perform spatial and channel-wise processing separately, allowing the model to learn more efficient feature representations. The feature extraction phase involves progressively learning hierarchical patterns from the input X-ray images, starting from low-level features (edges, textures) to high-level features (anomalies indicative of Pneumonia or Normal lungs).

Number of Features Extracted

- At the final feature map level, **2048 features** are extracted from the last convolutional block before global average pooling.
- The learned features include:
  - **Texture patterns**
  - **Boundaries and edges**
  - **Anomalous regions** relevant to medical conditions

Blocks Used in Xception

1. **Entry Flow (Initial Layers)**
   - Extracts basic image features.
   - Includes several convolutional layers and depthwise separable convolutions.
   - Downsampling occurs through strided convolutions and max pooling.
   - Filters: Start from **32** and progressively increase.
2. **Middle Flow (Depthwise Separable Convolutions)**
   - Consists of **8 identical blocks**, each containing depthwise separable convolutions.
   - Filters remain constant at **728** throughout these blocks.

- Purpose: Focuses on learning intermediate and high-level patterns, retaining spatial information.

3. **Exit Flow (Final Layers)**
   - Extracts high-level features essential for classification.
   - Ends with depthwise separable convolutions, followed by a final convolutional layer with **2048 filters**.
   - Purpose: Captures abstract patterns indicative of specific medical conditions.
4. **Global Average Pooling (GAP)**
   - Reduces the 2D feature map of size **(n, m, 2048)** to a 1D vector of **2048 features**, ensuring computational efficiency.
5. **Classification Head**
   - A fully connected (dense) layer with:
     - **1024 units** using ReLU activation and L2 regularization (to prevent overfitting).
     - **Dropout** (rate: 0.5) for regularization.
   - The output layer has **1 unit** with a sigmoid activation function for binary classification (Normal vs. Pneumonia).

Preprocessing Steps

1. **Rescaling**
   - **What's Done?**
     - All pixel values are scaled to the range [0, 1] by dividing by 255.
   - **Why?**
     - Normalizes input data, improving numerical stability and accelerating model convergence.
2. **Data Augmentation**
   - **What's Done?**
     - Applied only to the training dataset to reduce overfitting and improve robustness.
     - Techniques include:
       - **Rotation Range**: Up to 10°
       - **Width/Height Shifts**: Up to 10% of image dimensions
       - **Brightness Adjustment**: Ranges between 90% to 110% of original brightness
       - **Zoom Range**: Up to 10%

        - **Horizontal Flipping**: Simulates mirrored anatomical variations
    - **Why?**
        - Exposes the model to varied versions of the same data, making it more robust to real-world variations.

3. **Class Balancing**
    - **What's Done?**
        - Class weights computed to penalize misclassifications in imbalanced datasets.
    - **Why?**
        - Ensures the model pays equal attention to minority classes (e.g., Normal cases).
4. **Input Image Dimensions**
    - **What's Done?**
        - Resized all images to **(150, 150, 3)** to match the input shape of the Xception model.
    - **Why?**
        - Ensures uniformity across datasets and compatibility with the Xception architecture.
    - **Pros:** Reduces computational load.
    - **Cons:** Minor loss of detail for high-resolution images.
5. **Shuffling**
    - **What's Done?**
        - Training data is shuffled to prevent sequential learning.
        - Test data remains unshuffled for consistent evaluation.
    - **Why?**
        - Prevents overfitting and ensures reliable metric calculation.

Callback Mechanisms

1. **EarlyStopping**
    - **What's Done?**
        - Stops training if validation loss does not improve for 5 epochs.
    - **Pros:** Prevents overfitting.
    - **Cons:** Might terminate training prematurely.

-

2. **ModelCheckpoint**
    - **What's Done?**
        - Saves the model with the best validation loss.
    - **Pros:** Guarantees the retention of the best model during training.
    - **Cons:** Uses additional disk space.
3. **ReduceLROnPlateau**
    - **What's Done?**
        - Reduces the learning rate by half if validation loss stagnates for 3 epochs.
    - **Pros:** Helps the optimizer escape local minima.
    - **Cons:** Increases training time slightly.

Loss Function

- **Binary Crossentropy**
    - Suitable for binary classification tasks.
    - Optimizes log probabilities of correct predictions.

Optimizer

- **Adam Optimizer** with a learning rate of **1e-4**.
    - Combines benefits of momentum and adaptive learning rate methods.
- **Pros:**
    - Automatically adjusts learning rates.
    - Requires less tuning.
- **Cons:**
    - Slightly higher memory usage.

Evaluation Metrics

1. **Accuracy**: Measures the proportion of correct predictions.
2. **Confusion Matrix**: Visualizes true vs. predicted class performance.
3. **Precision, Recall, F1-Score**:
    - **Precision**: Evaluates false positives.
    - **Recall**: Evaluates false negatives.
    - **F1-Score**: Balances precision and recall.
4. **ROC-AUC**: Assesses the model's performance across various thresholds.

Additional Considerations

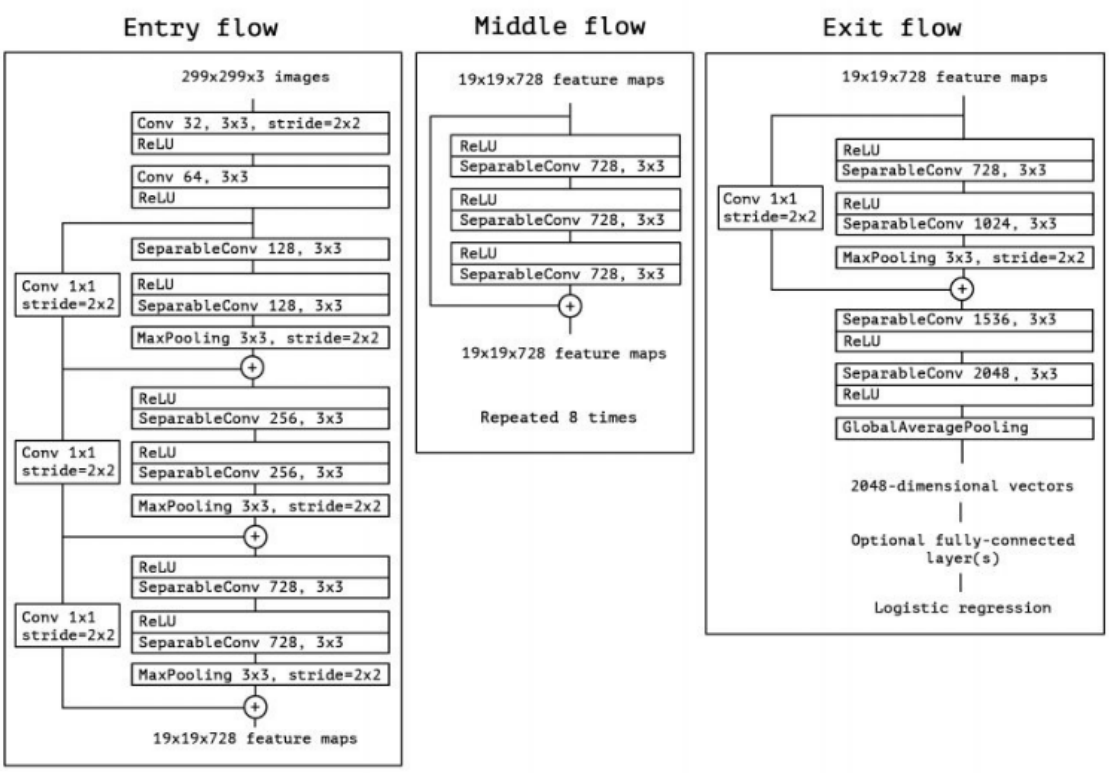
1. **Regularization**
    - L2 regularization in dense layers and dropout (rate: 0.5) mitigate overfitting.
2. **Training Epochs**
    - Limited to **15 epochs** with early stopping.
3. **Batch Size**
    - Fixed at **32** for training and testing datasets.

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 is an XCeption network?

What does it look like?

The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization.



Xception Architecture

Refrence[https://towardsdatascience.com/xception-from-scratch-using-tensorflow-even-better-than-inception-940fb231ced9]

Xception architecture has overperformed VGG-16, ResNet and Inception V3 in most classical classification challenges.

How does XCeption work?

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

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

Depthwise Separable Convolution

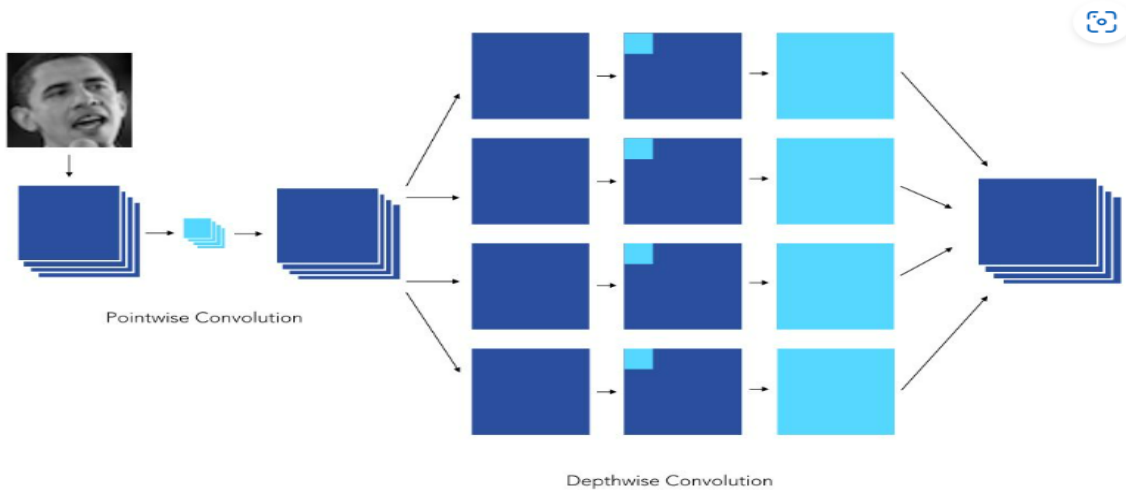
Depthwise Separable Convolutions are alternatives to classical convolutions that are supposed to be much more efficient in terms of computation time.

Implementation of the XCeption

XCeption offers an architecture that is made of Depthwise Separable Convolution blocks + Maxpooling, all linked with shortcuts as in ResNet implementations.

The specificity of XCeption is that the Depthwise Convolution is not followed by a Pointwise Convolution, but the order is reversed, as in this

example :



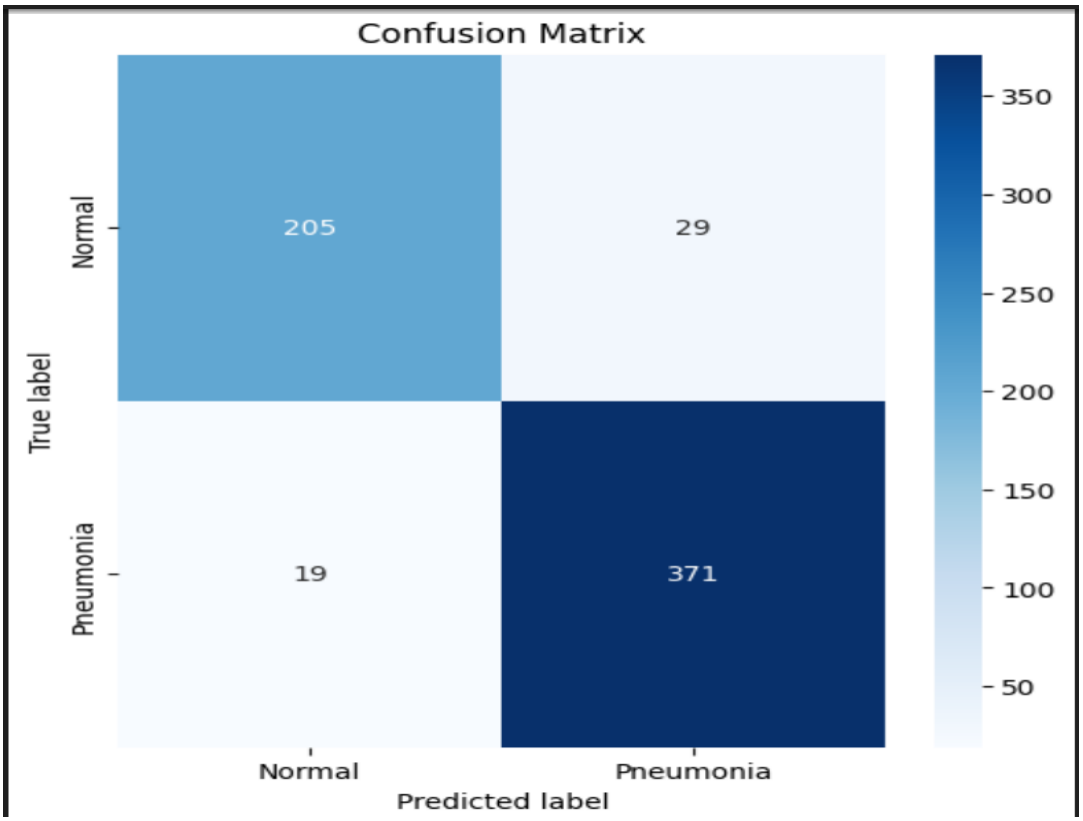
Reference [https://maelfabien.github.io/deeplearning/xception/#pointwise-convolution]

Accuracies

```
accuracy: 0.9710 - loss: 0.0758 - val_accuracy: 0.9311 - val_loss: 0.2255
```

```
Test Loss: 0.2132, Test Accuracy: 0.9231
```

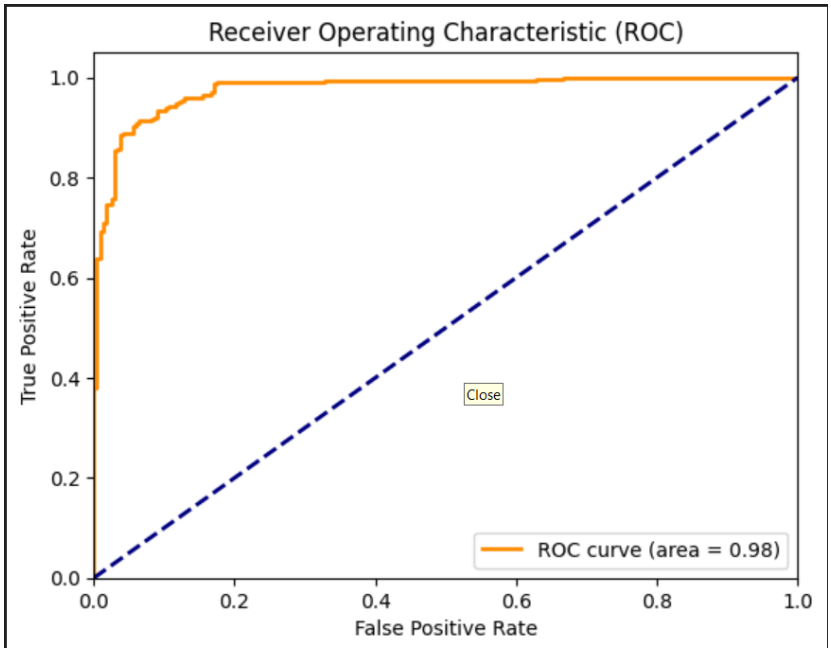
Confusion Matrix



Precision, Recall, F1-Score

	precision	recall	f1-score	support
0	0.92	0.88	0.90	234
1	0.93	0.95	0.94	390
accuracy			0.92	624
macro avg	0.92	0.91	0.92	624
weighted avg	0.92	0.92	0.92	624

ROC-AUC




```
AUC Score: 0.9765
```

ResNet Model

- **TP = 189 case was predicted correctly from 234**
- **TN = 343 case was predicted correctly from 390**
- **Total Accuracy = 85%**
- **AUC Score = 92.3%**

DenseNet121 Model

- **TP = 191 case was predicted correctly from 234**
- **TN = 380 case was predicted correctly from 390**
- **Total Accuracy = 91.5%**
- **AUC Score = 96.87%**

Xception Model

- **TP = 205 case was predicted correctly from 234**
- **TN = 371 case was predicted correctly from 390**
- **Total Accuracy = 92.3%**
- **AUC Score = 97.65%**

So from this evaluations we have achieved the highest accuracy in Xception model with accuracy 92.3%.

General comparison between ResNet, DenseNet121, and Xception:

1. ResNet (Residual Network)

- Key Feature: Introduces residual connections (skip connections) to avoid vanishing gradient issues.

Advantages:

1. **Deep Network Training**: Residual connections make it easier to train very deep networks (e.g., ResNet-50, ResNet-101).
2. **Improved Convergence**: Allows gradients to flow through the network by skipping layers, leading to faster and more stable convergence.
3. **Versatility**: Effective across a variety of tasks, including image recognition and object detection.
4. **Efficiency**: Computationally less expensive than models like DenseNet due to its sparse connectivity.

Disadvantages:

1. **Parameter Heavy**: High number of parameters compared to DenseNet, leading to higher memory usage.
2. **Redundancy**: The architecture sometimes introduces redundant layers that don't significantly improve performance.
3. **Difficulty Scaling**: Requires careful tuning for architectures beyond ResNet-152.

2. DenseNet121 (Densely Connected Convolutional Networks)

- **Key Feature**: Each layer is connected to all subsequent layers, allowing feature reuse.

Advantages:

1. **Feature Reuse**: Dense connectivity ensures efficient usage of feature maps, reducing redundancy.
2. **Reduced Parameters**: Fewer parameters compared to ResNet because of feature reuse, despite dense connectivity.
3. **Improved Gradient Flow**: Connections between all layers ensure better gradient flow during backpropagation.
4. **Compact Model**: DenseNet models are typically smaller in size compared to ResNet for similar performance.

Disadvantages:

1. **High Computational Cost**: Dense connectivity increases computational overhead and memory usage during training.
2. **Inference Latency**: Dense connections make it less efficient during inference.
3. **Scaling Issues**: Less scalable for very deep architectures due to high memory demands.

3. Xception (Extreme Inception)

- **Key Feature**: Combines depthwise separable convolutions and inception modules for efficient feature extraction.

Advantages:

1. **Efficient Computations**: Depthwise separable convolutions drastically reduce computational cost and parameter count.
2. **Performance**: Achieves high accuracy with fewer parameters compared to traditional CNNs like Inception and ResNet.
3. **Flexibility**: Adaptable to various tasks, including image classification and segmentation.
4. **Modular Design**: Easier to implement and customize due to its straightforward design.

Disadvantages:

1. **Complexity**: More complex than ResNet and DenseNet, requiring specialized implementation (e.g., depthwise separable convolutions).
2. **Hardware Dependency**: Benefits of separable convolutions are maximized on hardware optimized for such operations.
3. **Overfitting Risk**: Tends to overfit smaller datasets due to its high capacity.

Choosing the Right Model

- **ResNet**: Best for general-purpose tasks where depth is essential and computational resources are moderate.

- **DenseNet121**: Best when memory is sufficient and accuracy is critical, particularly for medium-sized datasets.

- **Xception**: Ideal for lightweight, high-performance applications or tasks optimized for modern hardware.

General Comparison Between 3 Models

Feature	ResNet	DenseNet121	Xception
Gradient Handling	Residual connections for smooth flow	Dense connections improve flow	Efficient separable convolutions
Parameter Count	High	Low	Medium
Computational Cost	Moderate	High	Low
Memory Usage	Moderate	High	Low
Accuracy	High	Very High	Very High
Training Efficiency	Easy to train	Harder due to memory demands	Moderate
Inference Speed	Fast	Slower due to dense layers	Fast

References

[https://www.researchgate.net/figure/Comparison-of-the-performances-of-DenseNet-121-and-ResNet-50-with-different-r-with-either_tbl1_339294578]

[https://medium.com/@enrico.randellini/image-classification-resnet-vs-efficientnet-vs-efficientnet-v2-vs-compact-convolutional-c205838bbf49]

[https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/]