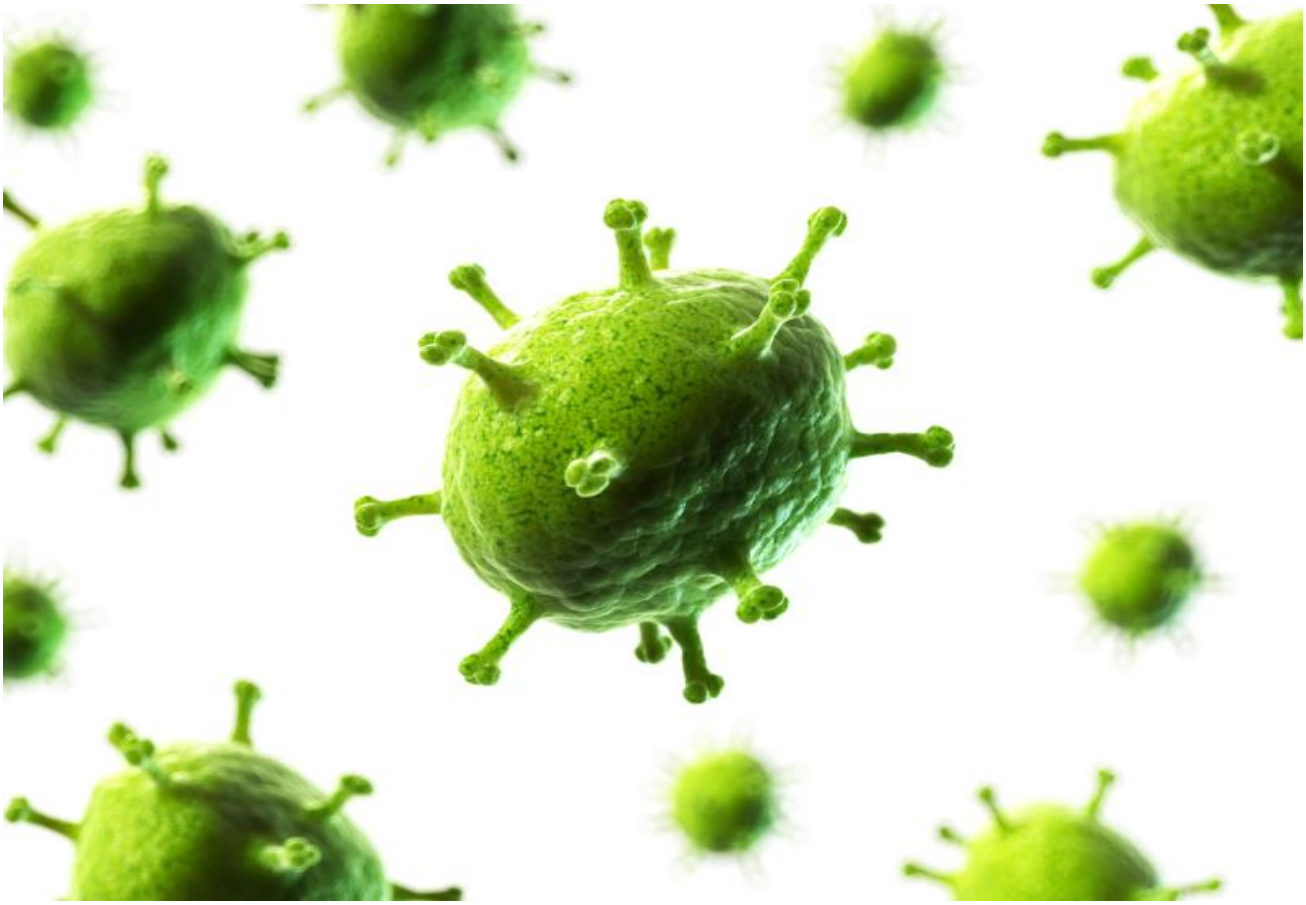


Neural Network & Deep Learning project

Virus Classification Model Documentation

Overview

This project performs virus classification using deep learning models, leveraging both custom-built ResNet-50 architecture from scratch and pre-trained models (Xception and DenseNet121) for transfer learning. The main objective is to classify images into different categories based on virus type, using a dataset split into training, validation, and test sets. The model is trained, evaluated, and saved for later use.

Libraries Used

TensorFlow/Keras: The core deep learning framework for building and training the models.

- **layers:** Provides layer types for neural networks.
- **models:** Facilitates the creation of deep learning models.
- **optimizers:** Provides various optimization algorithms (e.g., Adam).

applications: Includes pre-trained models such as Xception and DenseNet121.

ImageDataGenerator: Used for image preprocessing and augmentation.

NumPy: Provides support for numerical operations, especially when handling arrays.

Seaborn/Matplotlib: Used for visualizing results such as confusion matrices.

scikit-learn: Used for generating classification reports and confusion matrices to evaluate model performance.

Parameters

IMG_HEIGHT, IMG_WIDTH: Target dimensions (128x128) for resizing input images.

BATCH_SIZE: The number of samples to process per batch (64).

EPOCHS: Number of times the entire dataset will pass through the model during training (20).

LEARNING_RATE: Learning rate for the optimizer (0.001).

Data Loading and Augmentation

ImageDataGenerator is used for:

Rescaling: Scaling image pixel values between 0 and 1.

Augmentation: Random transformations (rotation, shift, horizontal flip) to improve model generalization.

Validation Split: Split dataset into training and validation sets (80% training, 20% validation).

The following data generators are created:

train_generator: For loading training images.

val_generator: For loading validation images.

test_generator: For loading test images, without shuffling for accurate evaluation.

ResNet-50 Model (Custom)

Description

- ResNet50 (introduced by He et al. in *"Deep Residual Learning for Image Recognition"*, CVPR 2016) is based on residual learning.

- Key feature: Skip connections (residual blocks) solve the vanishing gradient problem by adding shortcuts that skip over some layers.
- ResNet50 has 50 layers consisting of:
    - Convolutional layers, batch normalization, and shortcut connections.

Explanation:

Bottleneck Block: This function defines a single block of the ResNet-50 architecture, using 1x1 and 3x3 convolutions with batch normalization and ReLU activations. It includes skip connections to help with gradient flow.

build_resnet50: Constructs the entire ResNet-50 model with multiple bottleneck blocks, starting from an initial convolution layer and proceeding through layers with increasing filter sizes. The final output layer is a softmax layer that classifies into num_classes categories.

Advantages of ResNet-50

1.Solves Vanishing Gradient Problem:

- ResNet-50 uses skip connections, allowing gradients to flow directly through the network during backpropagation, which

alleviates the vanishing gradient problem common in deep networks.

2. **Deep Architecture:**
    - With 50 layers, ResNet-50 is deep enough to capture complex patterns and hierarchical features in data, making it suitable for tasks like image classification, object detection, and more.
3. **Residual Learning:**
    - Instead of learning a direct mapping, ResNet-50 models residuals (differences between inputs and outputs), making training faster and more accurate.
4. **Pre-Trained Availability:**
    - Pre-trained ResNet-50 models are widely available (trained on ImageNet), enabling effective transfer learning for a variety of tasks with limited data.
5. **Efficiency in Parameters:**
    - Despite being deep, ResNet-50 uses bottleneck blocks, which reduce computational complexity and memory

requirements by combining 1x1 convolutions with 3x3 convolutions.

6. **Generalization:**
   - The model generalizes well to unseen data due to its ability to learn hierarchical features and reduce overfitting.
7. **Versatility:**
   - ResNet-50 is a flexible architecture that can be adapted to many tasks (e.g., image segmentation, object recognition, and medical imaging) by modifying the output layers.

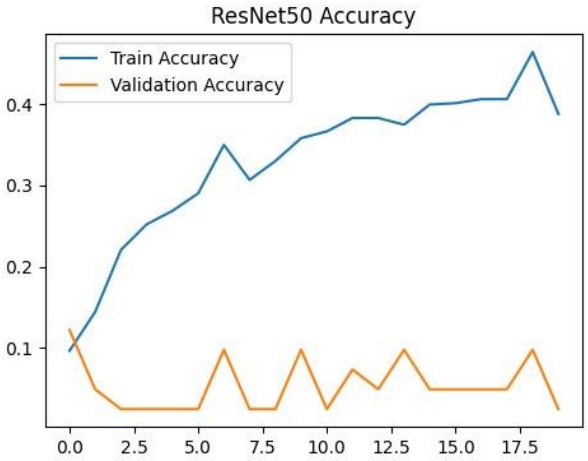
Disadvantages of ResNet-50

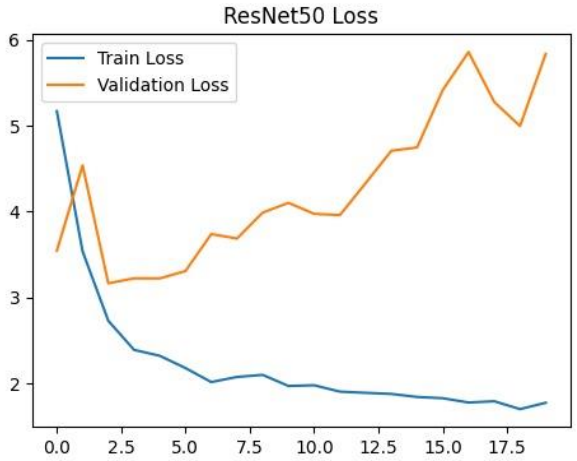
1. **High Computational Cost:**
   - ResNet-50 requires significant computational resources (GPU/TPU) for both training and inference due to its depth and number of parameters (~25.5 million).
2. **Training Time:**
   - The model is relatively slow to train compared to shallower architectures due to its depth.

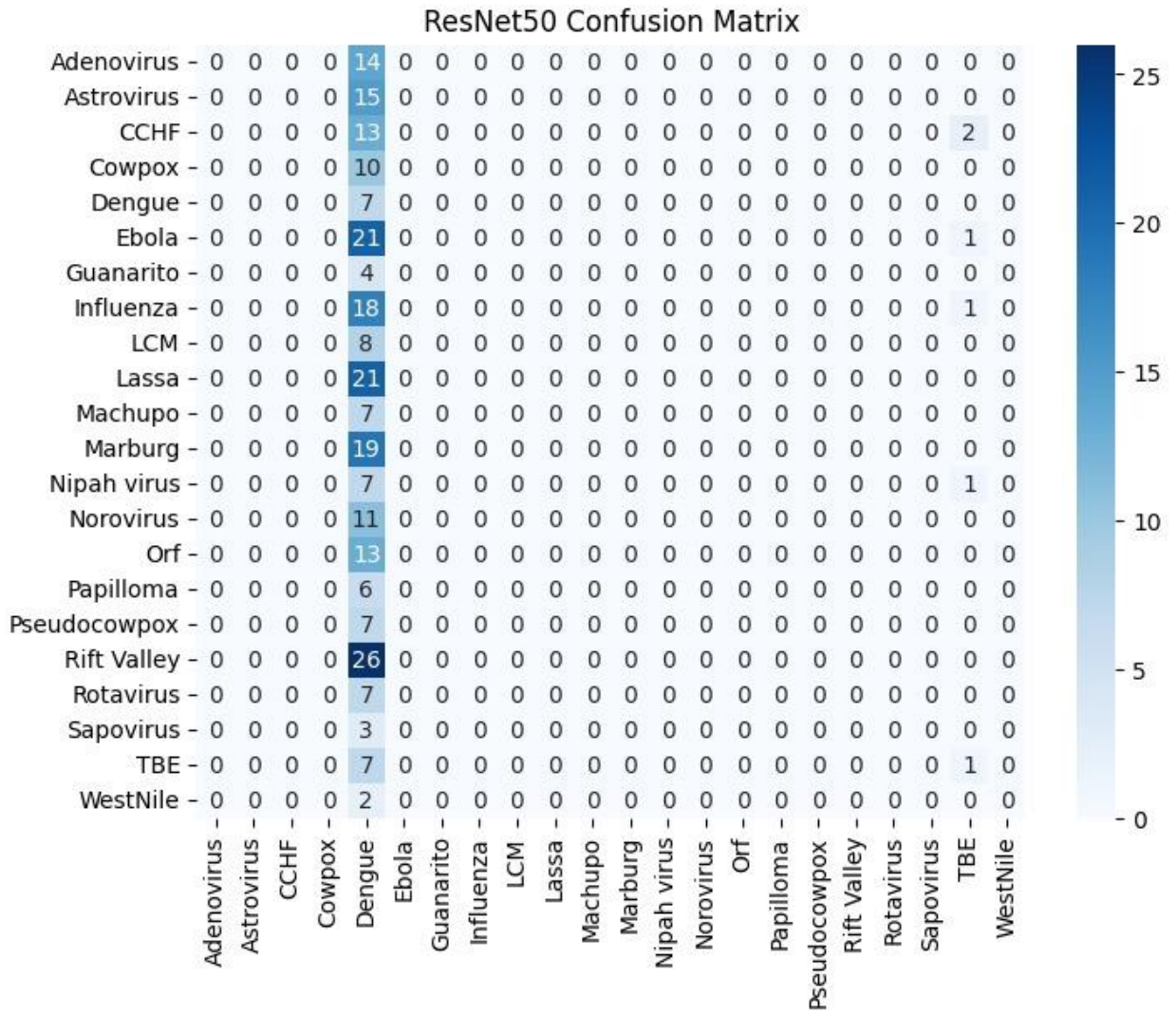
3. **Overfitting on Small Datasets:**
   - If used without proper regularization or data augmentation, ResNet-50 can overfit when trained on small or imbalanced datasets.
4. **Complexity:**
   - The architecture is more complex than simpler models like VGG, making it harder to debug or modify for custom applications.
5. **Inefficiency for Small Datasets:**
   - The depth of ResNet-50 might be unnecessary for tasks with limited data or simple classification problems, where smaller models could perform comparably with less computational cost.
6. **Memory Intensive:**
   - The model requires significant memory for storing intermediate feature maps and gradients, making it less suitable for systems with limited memory.
7. **Gradient Instability in Very Deep Versions:**
   - Although ResNet-50 handles the vanishing gradient issue well, extremely deep

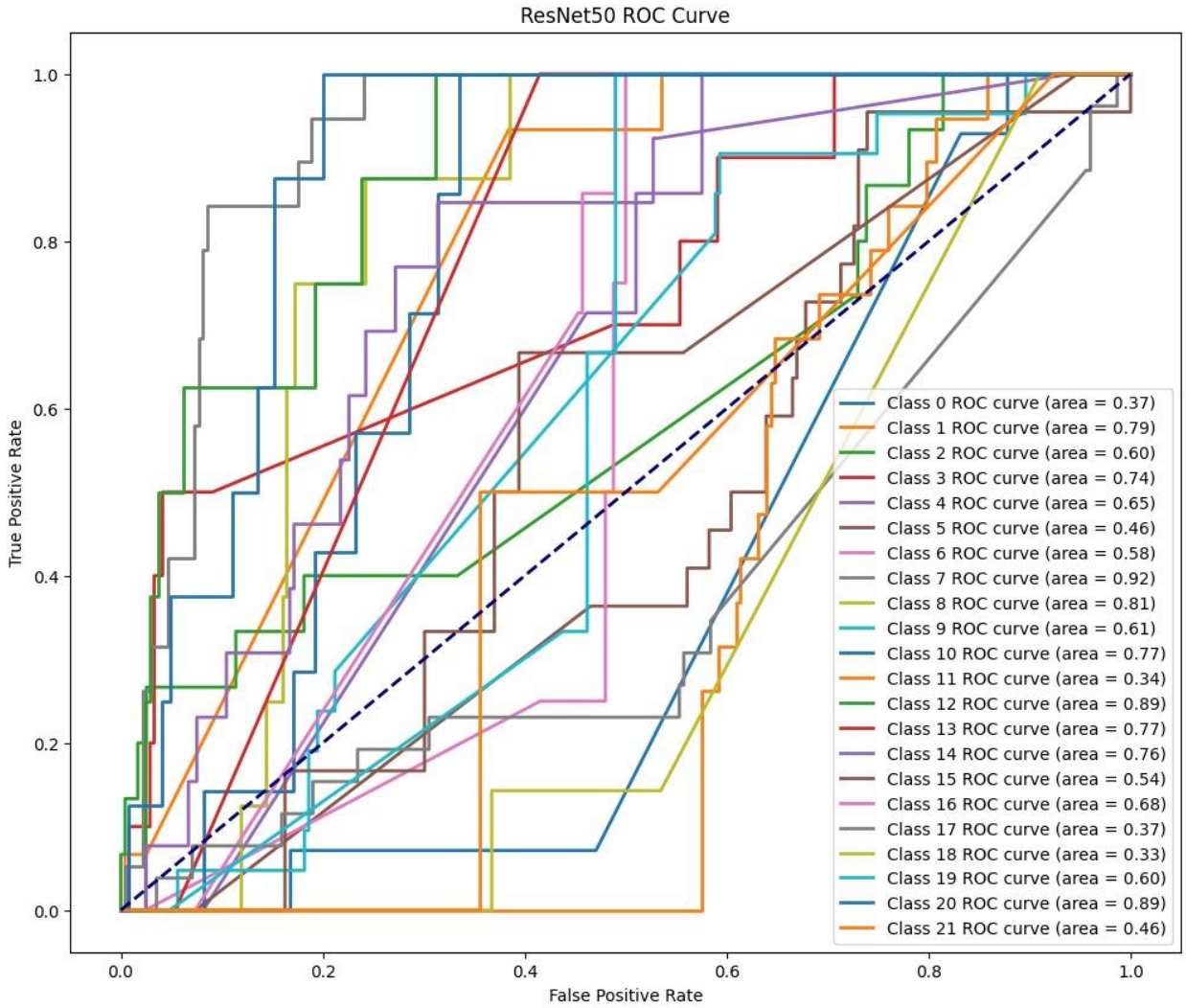
versions of ResNet (e.g., ResNet-152) may encounter gradient instability without additional techniques like adaptive gradient clipping.

Graphs:






ResNet50 Confusion Matrix
Adenovirus 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Astrovirus 0 0 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CCHF 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
Cowpox 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Dengue 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Ebola 0 0 0 0 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
Guanarito 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Influenza 0 0 0 0 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
LCM 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Lassa 0 0 0 0 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Machupo 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Marburg 0 0 0 0 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Nipah virus 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
Norovirus 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Orf 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Papilloma 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Pseudocowpox 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Rift Valley 0 0 0 0 26 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Rotavirus 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Sapovirus 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TBE 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
WestNile 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Adenovirus
Astrovirus
CCHF
Cowpox
Dengue
Ebola
Guanarito
Influenza
LCM
Lassa
Machupo
Marburg
Nipah virus
Norovirus
Orf
Papilloma
Pseudocowpox
Rift Valley
Rotavirus
Sapovirus
TBE
WestNile
25
20
15
10
5
0



Paper link:

https://ar5iv.org/html/1512.03385

Model Compilation and Training:

The model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical crossentropy loss (since the problem is a multi-class classification task).

The model is trained using the train_generator and validated on val_generator for 20 epochs.

The trained model is saved as resnet50_virus_classification.h5.

Transfer Learning Models

Two pre-trained models, Xception and DenseNet121, are used for transfer learning:

Xception:

Description:

- The Xception architecture (introduced by François Chollet in *"Xception: Deep Learning with Depthwise Separable Convolutions"*, 2017) is based on the Inception design but replaces traditional convolutions with depthwise separable convolutions. This reduces computational cost while maintaining accuracy.
- **Key features:**
  - Depthwise separable convolutions: Factorize a convolution into depthwise (per-channel) and pointwise (1x1) convolutions.

- Fewer parameters compared to standard convolutions.
- More efficient representation learning.

Explanation:

Pre-trained weights from ImageNet are loaded.

The base model is frozen (not trained further), and custom layers are added on top, including a Global Average Pooling layer, Dense layer (256 units), Dropout (0.5), and the final output layer for classification.

This model is compiled, trained, and saved as xception_virus_classification.h5.

Advantages of Xception

1. **Efficient Use of Parameters:**
   - Xception improves parameter efficiency by replacing standard convolutions with depthwise separable convolutions, significantly reducing the number of trainable parameters while maintaining performance.
2. **Improved Feature Learning:**

- Depthwise separable convolutions enable the model to learn spatial (depthwise) and channel-wise (pointwise) features separately, leading to better feature representation.

3. **High Accuracy:**
   - Xception outperforms traditional convolutional neural networks like VGG and Inception on large-scale datasets, such as ImageNet, due to its optimized architecture.
4. **Transfer Learning Potential:**
   - Xception performs well in transfer learning tasks, as its pre-trained weights (trained on ImageNet) can be fine-tuned for diverse applications like medical imaging or object detection.
5. **Computational Efficiency:**
   - Although deep, the use of depthwise separable convolutions reduces the overall computational cost compared to traditional convolutions.
6. **Scalable:**

- The architecture can scale effectively with larger datasets and more complex tasks by increasing layers without a significant computational trade-off.

7. **State-of-the-Art Architecture:**
   - Xception builds upon and generalizes the Inception architecture by completely factorizing the convolution process, resulting in superior performance in many benchmarks.

Disadvantages of Xception

1. **High Computational Cost for Small Datasets:**
   - While efficient for large-scale datasets, Xception may be computationally overkill for small or simple datasets where less complex models like MobileNet or ResNet-18 can suffice.
2. **Complex Architecture:**
   - The separation of depthwise and pointwise convolutions makes the architecture harder to understand and implement compared to simpler models like VGG.
3. **Resource Intensive:**

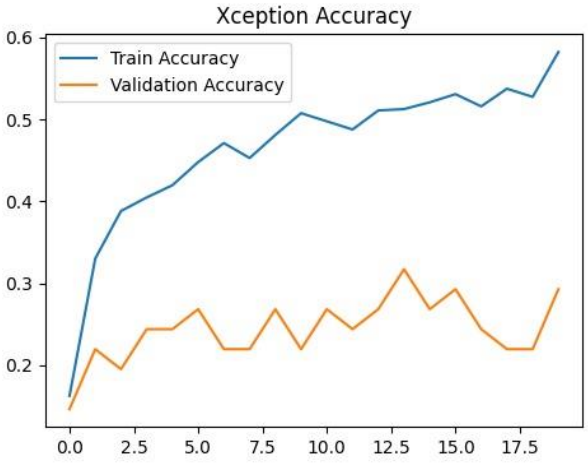
- Despite computational optimizations, Xception requires significant GPU/TPU resources for efficient training and inference, especially for high-resolution images.

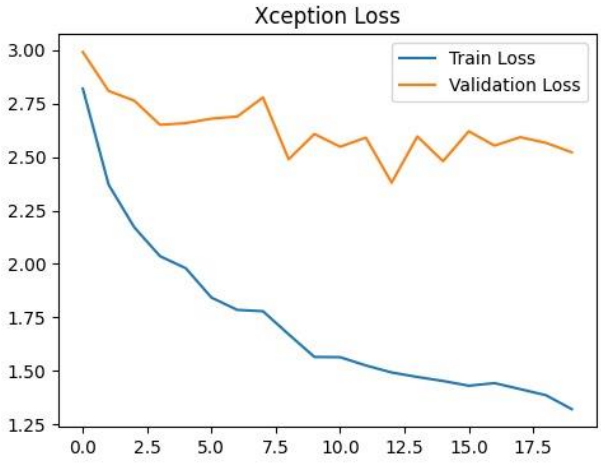
4. **Overfitting Risk:**
   - When used on small datasets, Xception can overfit due to its depth and large number of parameters unless proper regularization and augmentation techniques are applied.
5. **Requires Large Training Data:**
   - Xception's full potential is realized on large and diverse datasets. On smaller datasets, simpler architectures might yield similar performance with less computational cost.
6. **Training Stability:**
   - Deeper architectures like Xception can suffer from unstable training without appropriate learning rate schedulers or initialization techniques.
7. **Memory Usage:**
   - Due to its depth and number of operations, Xception can be memory-intensive, making

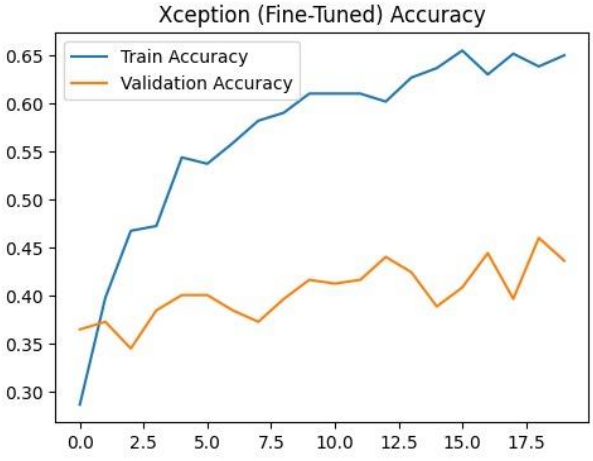
it less suitable for devices with limited resources, like edge devices or mobile phones.

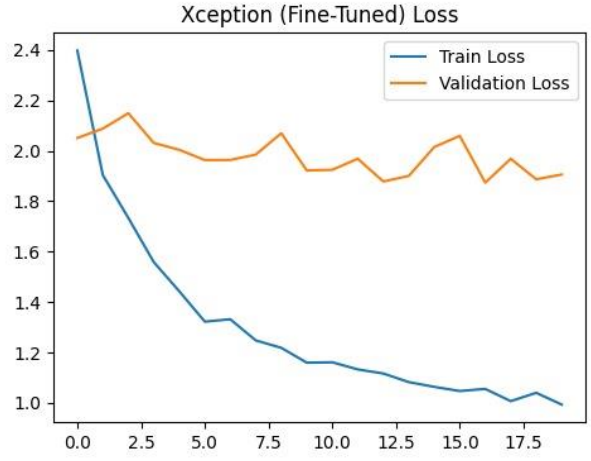
Graphs

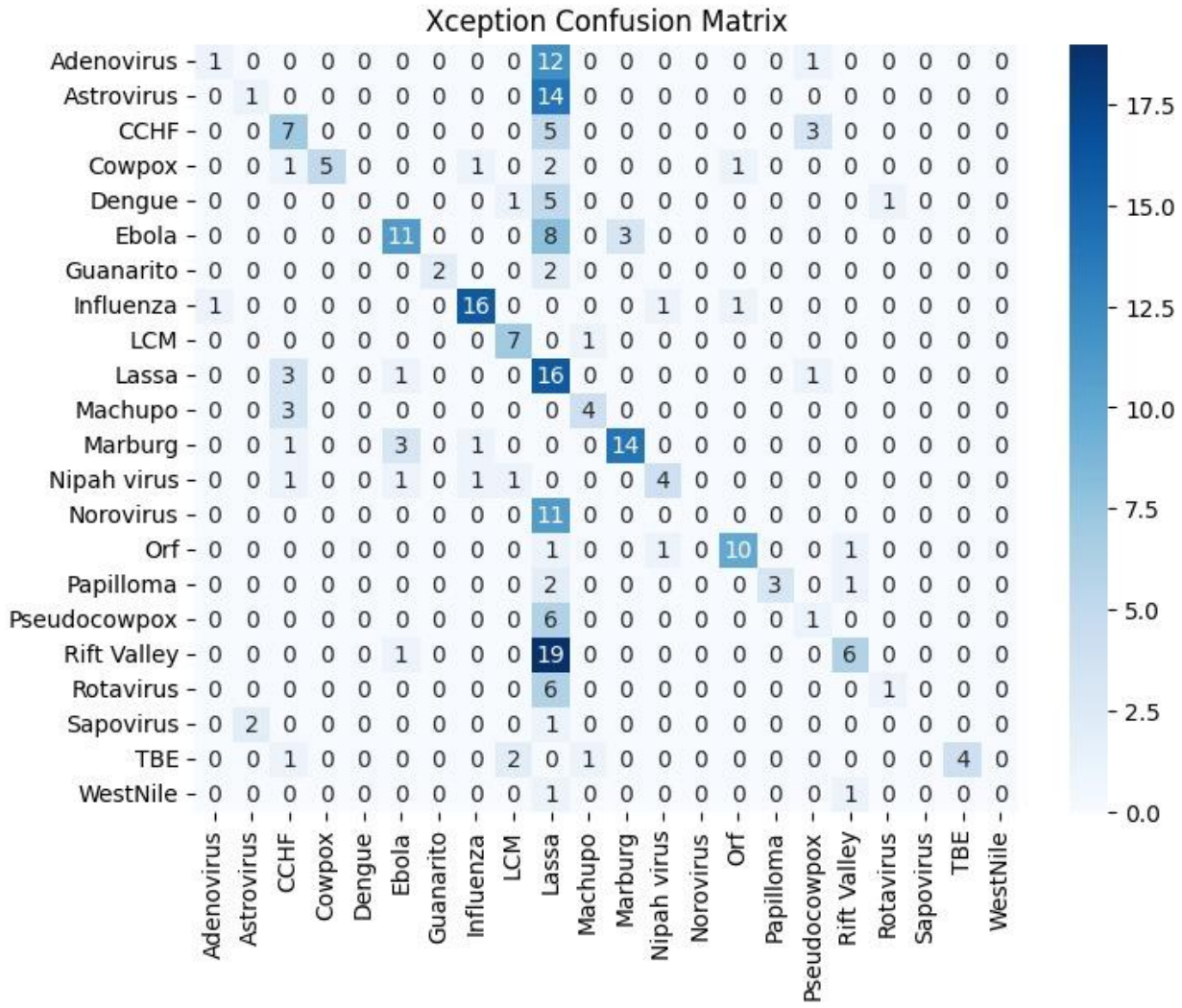
- **Accuracy and Loss Graphs:**
  - The training and validation accuracy/loss were plotted across epochs.
  - These show model performance improvement during transfer learning and fine-tuning.

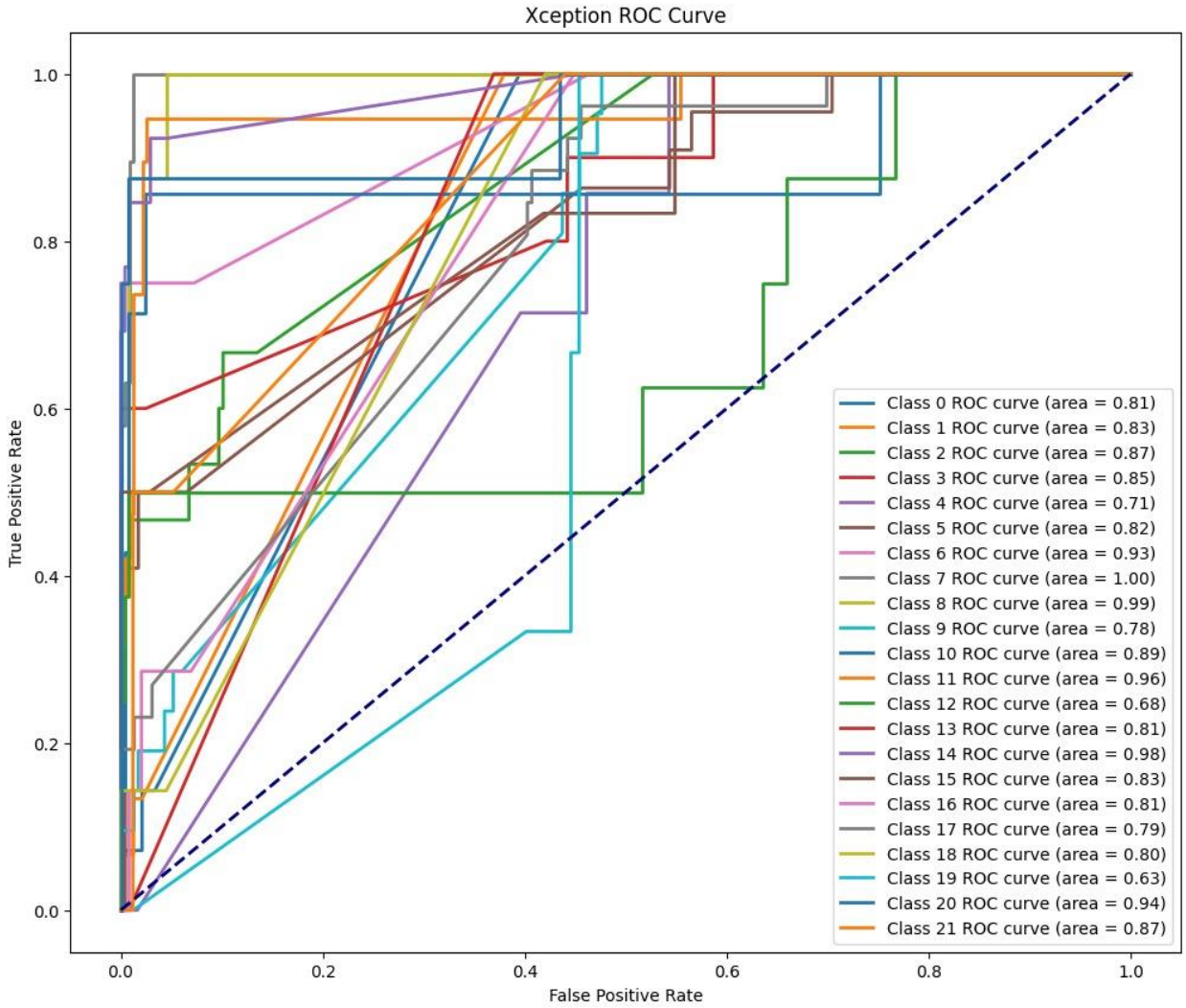









Xception Confusion Matrix
Adenovirus 1 0 0 0 0 0 0 0 0 12 0 0 0 0 0 0 1 0 0 0 0 0
Astrovirus 0 1 0 0 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0
CCHF 0 0 7 0 0 0 0 0 0 5 0 0 0 0 0 0 3 0 0 0 0 0
Cowpox 0 0 1 5 0 0 0 1 0 2 0 0 0 0 1 0 0 0 0 0 0 0
Dengue 0 0 0 0 0 0 0 0 1 5 0 0 0 0 0 0 0 0 1 0 0 0
Ebola 0 0 0 0 0 11 0 0 0 8 0 3 0 0 0 0 0 0 0 0 0 0
Guanarito 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0
Influenza 1 0 0 0 0 0 0 16 0 0 0 0 1 0 1 0 0 0 0 0 0 0
LCM 0 0 0 0 0 0 0 0 7 0 1 0 0 0 0 0 0 0 0 0 0 0
Lassa 0 0 3 0 0 1 0 0 0 16 0 0 0 0 0 0 1 0 0 0 0 0
Machupo 0 0 3 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0
Marburg 0 0 1 0 0 3 0 1 0 0 0 14 0 0 0 0 0 0 0 0 0 0
Nipah virus 0 0 1 0 0 1 0 1 1 0 0 0 4 0 0 0 0 0 0 0 0 0
Norovirus 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0
Orf 0 0 0 0 0 0 0 0 0 1 0 0 1 0 10 0 0 1 0 0 0 0
Papilloma 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 3 0 1 0 0 0 0
Pseudocowpox 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 1 0 0 0 0 0
Rift Valley 0 0 0 0 0 1 0 0 0 19 0 0 0 0 0 0 0 6 0 0 0 0
Rotavirus 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 1 0 0 0
Sapovirus 0 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
TBE 0 0 1 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 4 0
WestNile 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
Adenovirus
Astrovirus
CCHF
Cowpox
Dengue
Ebola
Guanarito
Influenza
LCM
Lassa
Machupo
Marburg
Nipah virus
Norovirus
Orf
Papilloma
Pseudocowpox
Rift Valley
Rotavirus
Sapovirus
TBE
WestNile
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0



Paper link:

https://www.proquest.com/openview/496560430a0455d0e25a4a0bb9a89f02/1?pq-origsite=gscholar&cbl=5444811

DenseNet121:

Description:

- DenseNet (Dense Convolutional Network) was introduced by Huang et al. in *"Densely Connected Convolutional Networks"*, CVPR 2017.
- Key idea: Dense connections between layers ensure feature reuse, leading to better gradient flow and fewer parameters.
- DenseNet121 consists of:
    - Dense blocks: Each layer connects to all previous layers.
    - Efficient feature propagation and reduced vanishing gradients.

Explanation:

Similar to Xception, the DenseNet121 model is used with pre-trained ImageNet weights. Custom layers are appended for classification, and the model is trained and saved as densenet121_virus_classification.h5.

Advantages of DenseNet121

1. **Efficient Parameter Usage:**
   - DenseNet121 connects each layer to every other layer in a feedforward fashion, allowing feature reuse. This drastically reduces the number of parameters compared to architectures like ResNet while maintaining similar or better performance.
2. **Feature Reuse:**
   - Each layer has access to all the feature maps from preceding layers, enabling effective feature propagation and reuse. This mitigates the problem of vanishing gradients and improves learning.
3. **Mitigates Vanishing Gradient Problem:**
   - The dense connectivity ensures efficient gradient flow throughout the network, making it easier to train deeper models.
4. **Compact Model Size:**
   - DenseNet121 has fewer parameters compared to other deep networks with similar depth and performance, making it

memory-efficient and suitable for resource-constrained environments.

5. **Improved Learning Efficiency:**
   - By concatenating features instead of summing them (as in ResNet), DenseNet121 preserves information learned by previous layers, resulting in better feature representation.
6. **State-of-the-Art Performance:**
   - DenseNet121 achieves excellent performance on benchmark datasets like ImageNet with relatively fewer parameters and computational cost.
7. **Regularization Effect:**
   - The dense connections act as implicit regularization, reducing overfitting and making DenseNet121 generalize well even on smaller datasets.
8. **Pre-trained Weights Availability:**
   - DenseNet121 is available with pre-trained weights on ImageNet, making it suitable for transfer learning and reducing the training time for new tasks.

9. **Fewer Hyperparameters to Tune:**
    - The architecture is less sensitive to hyperparameter changes, making it easier to train and adapt to various tasks.

Disadvantages of DenseNet121

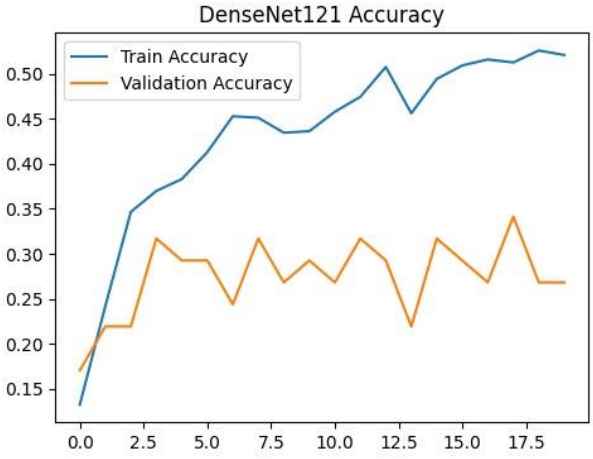
1. **High Computational Cost:**
    - Although compact in size, DenseNet121 requires significant computational resources for training due to the dense connectivity and frequent concatenation operations.
2. **Increased Memory Usage During Training:**
    - The concatenation of feature maps leads to higher memory usage during training, as all previous feature maps need to be stored and passed to subsequent layers.
3. **Inference Speed:**
    - While efficient in terms of parameters, the dense connectivity can slow down inference due to the need to compute and store multiple intermediate feature maps.
4. **Overhead in Feature Concatenation:**

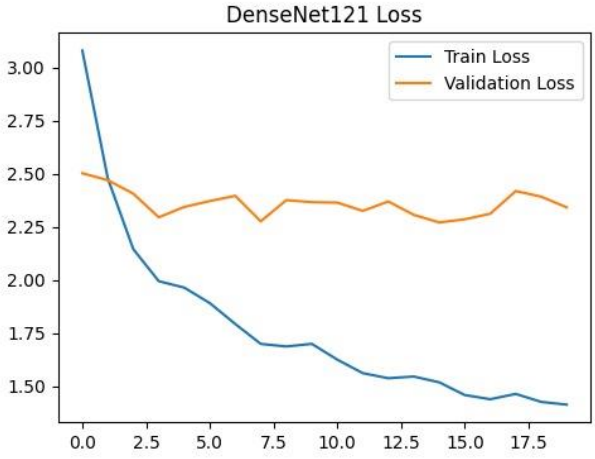
- The dense connections result in a larger number of feature maps in the later layers, which can lead to computational bottlenecks in GPUs with limited memory.

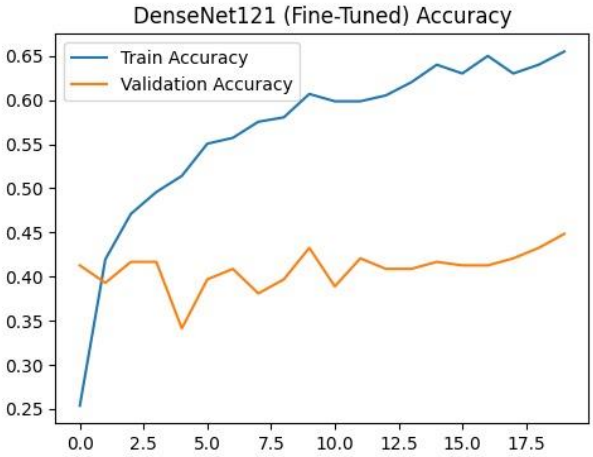
5. **Complexity in Implementation:**
   - The architecture's design, involving dense block connections, can make it slightly more challenging to implement and debug compared to simpler models like ResNet.
6. **Not Suitable for All Tasks:**
   - DenseNet121 might not always outperform simpler architectures on small or less complex datasets, where the dense connections might not provide significant advantages.
7. **Scalability Issues:**
   - Extending the network depth significantly increases the computational and memory requirements, making it less scalable for extremely deep architectures compared to simpler designs.

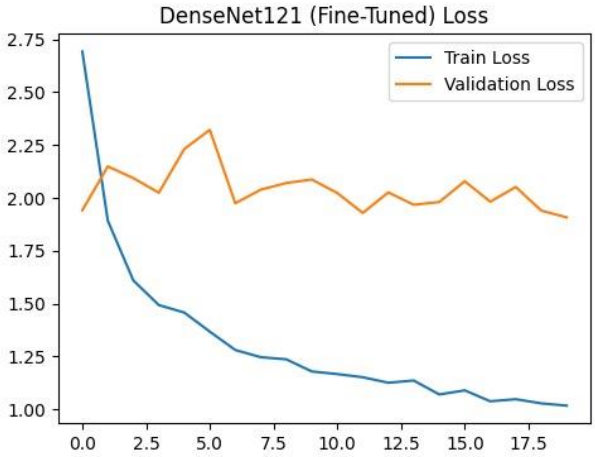
Graphs

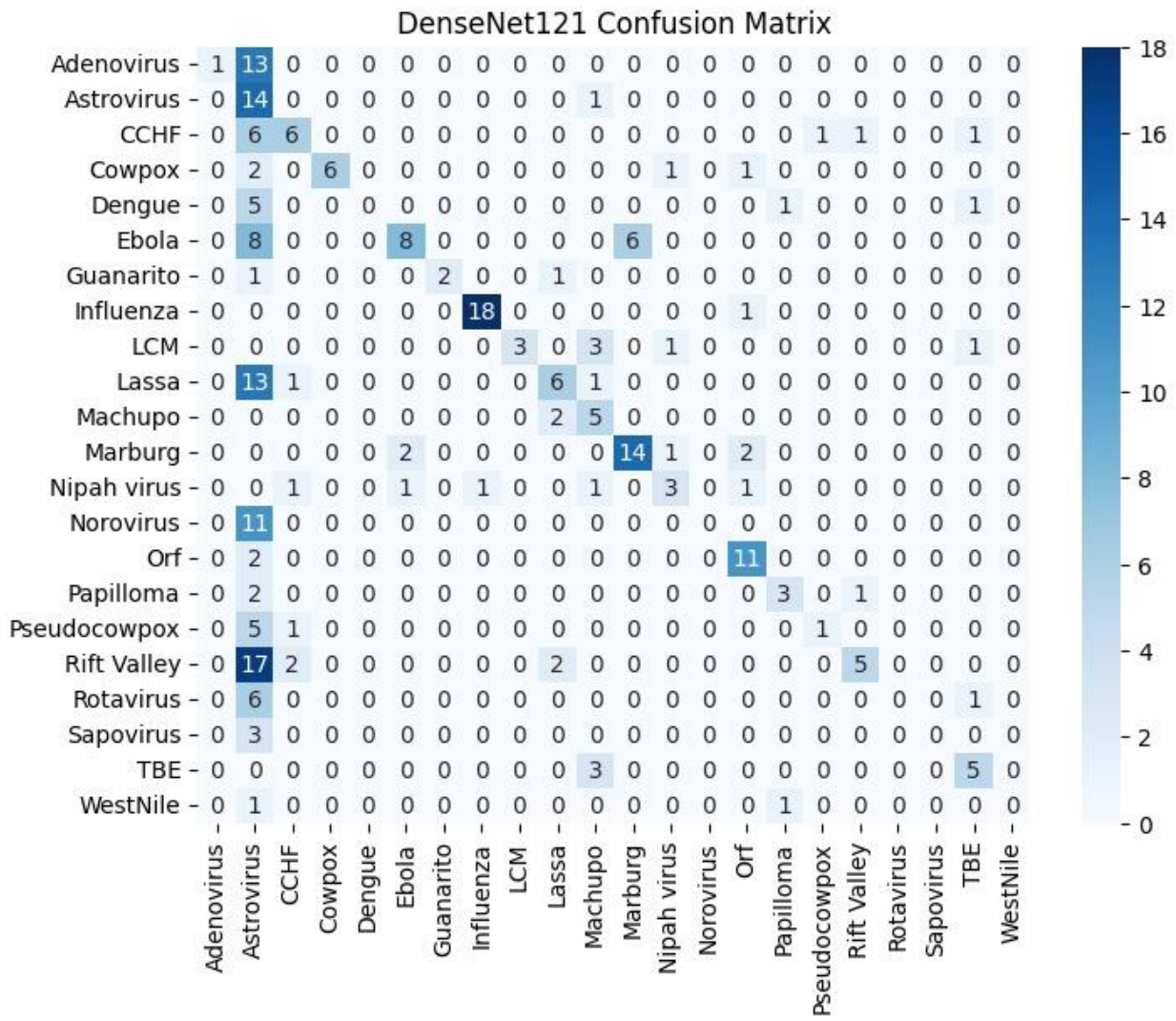
- Plots demonstrate the model's accuracy and loss over epochs during training and fine-tuning.

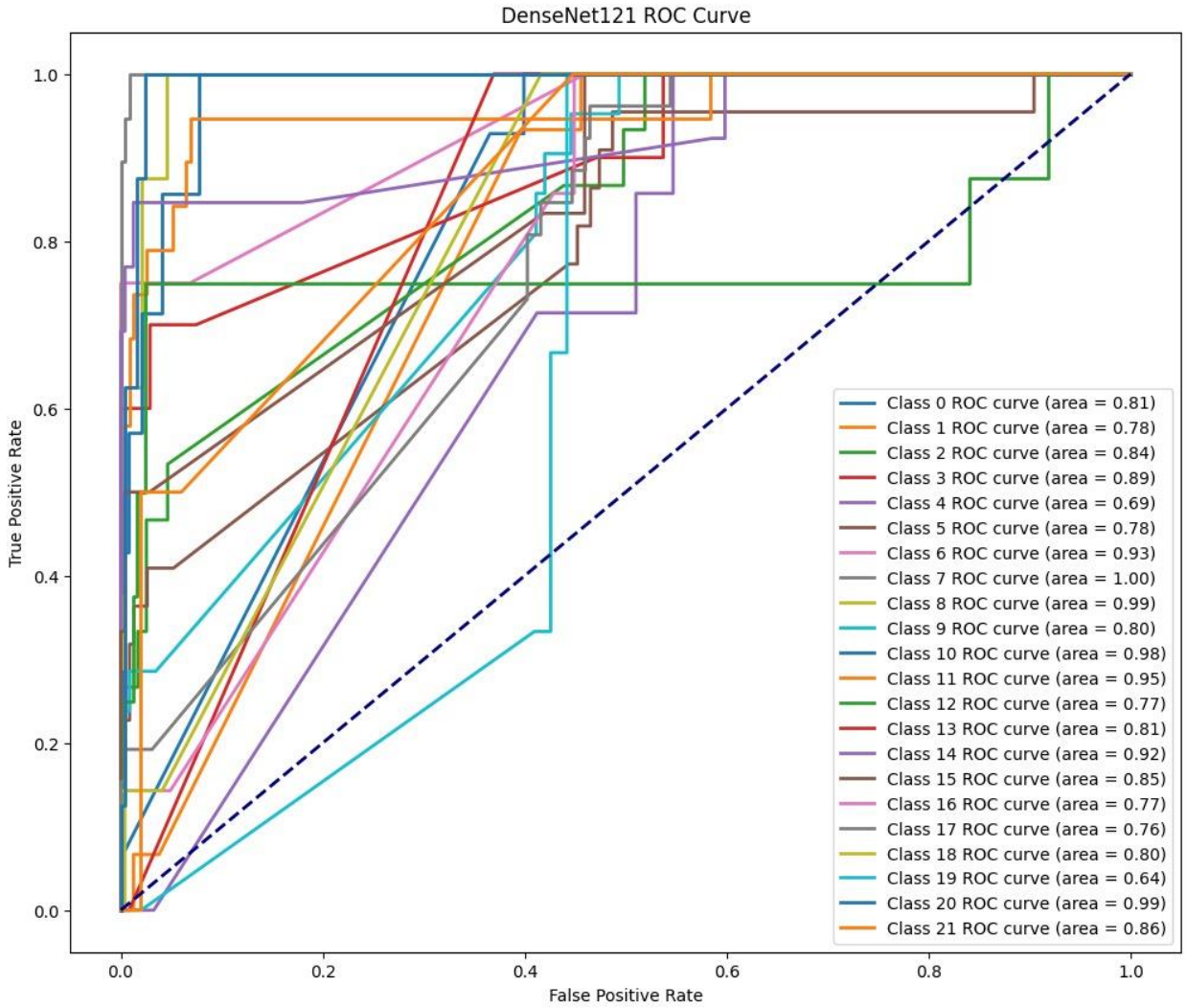









DenseNet121 Confusion Matrix
Adenovirus 1 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Astrovirus 0 14 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
CCHF 0 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0
Cowpox 0 2 0 6 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
Dengue 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
Ebola 0 8 0 0 0 8 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0
Guanarito 0 1 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Influenza 0 0 0 0 0 0 0 18 0 0 0 0 0 0 1 0 0 0 0 0 0 0
LCM 0 0 0 0 0 0 0 0 3 0 3 0 1 0 0 0 0 0 0 0 1 0
Lassa 0 13 1 0 0 0 0 0 0 6 1 0 0 0 0 0 0 0 0 0 0 0
Machupo 0 0 0 0 0 0 0 0 0 2 5 0 0 0 0 0 0 0 0 0 0 0
Marburg 0 0 0 0 0 2 0 0 0 0 0 14 1 0 2 0 0 0 0 0 0 0
Nipah virus 0 0 1 0 0 1 0 1 0 0 1 0 3 0 1 0 0 0 0 0 0 0
Norovirus 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Orf 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0
Papilloma 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 1 0 0 0
Pseudocowpox 0 5 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
Rift Valley 0 17 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 5 0 0 0
Rotavirus 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
Sapovirus 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TBE 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 5 0
WestNile 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
Adenovirus
Astrovirus
CCHF
Cowpox
Dengue
Ebola
Guanarito
Influenza
LCM
Lassa
Machupo
Marburg
Nipah virus
Norovirus
Orf
Papilloma
Pseudocowpox
Rift Valley
Rotavirus
Sapovirus
TBE
WestNile
18
16
14
12
10
8
6
4
2
0



Paper link:

https://link.springer.com/content/pdf/10.1007/s42979-021-00782-7.pdf

Model Evaluation

evaluate_model: This function evaluates the trained models on the test set using the classification_report from scikit-learn to generate precision, recall, and F1-score metrics for each class. The confusion_matrix is also plotted using Seaborn for visualizing classification errors.

The function takes:

The trained model.

The test dataset.

The model name (for logging and visualization).

The evaluation is performed for all three models:

ResNet-50

Xception

DenseNet121

Training and Model Saving

The models are trained using the fit method, utilizing the train_generator and val_generator for training and validation respectively.

After training, each model is saved in a .h5 file format, allowing for future inference or further fine-tuning.

Visualizations

Confusion Matrix: A heatmap is generated to visualize the model's performance in terms of true positives, false positives, true negatives, and false negatives for each class.

Seaborn’s heatmap function is used to create the matrix, and matplotlib is used for plotting.

Limitations and Potential Improvements

Data Augmentation: Currently, basic augmentation (rotation, shifting, flipping) is used. You can explore more advanced augmentation techniques such as random cropping or color adjustments.

Model Tuning: The architecture can be further optimized by fine-tuning hyperparameters like learning rate, batch size, or adding dropout layers to prevent overfitting.

Real-World Data: This setup assumes you have a well-labeled and balanced dataset. Ensure that

the dataset is cleaned and balanced to avoid model biases.

Important rules for computing visualization:

TP (True Positive): The number of instances that were correctly classified as positive.

TN (True Negative): The number of instances that were correctly classified as negative.

FP (False Positive): The number of instances that were incorrectly classified as positive.

FN (False Negative): The number of instances that were incorrectly classified as negative.

1. Accuracy:

   Accuracy = (TP + TN) / (TP + TN + FP + FN)

2. Recall:

   Recall = TP/ (TP + FN)

3. precision:

   Precision = TP/ (TP+ FP)

4. F1_Score:

   F1_score = 2 * (Precision * Recall) /
   (Precision + Recall)

Comparison in brief:

Architecture	Pros	cons	Best Suited For	Suitability for Dataset
Xception	good performance, efficient inference	More complex implementation	Image classification on resource-constrained devices	**Potentially Good:** If efficiency and inference speed are critical, Xception could be a strong contender.
DenseNet	Feature reuse, reduced parameters	Increased memory consumption during training	Image classification and segmentation	**Potentially Good:** If the dataset is large and complex, DenseNet's feature reuse might be beneficial.
ResNet50	simple architecture, widely used	May not always achieve state-of-the-art results	General-purpose image-related tasks, smaller datasets	**Potentially Good:** A good starting point due to its simplicity and widespread use. Could be a good baseline.

Conclusion

This project demonstrates how to use custom-built and pre-trained deep learning models for image classification tasks. By leveraging transfer learning, the Xception and DenseNet121 models are trained on a relatively small dataset, while the custom ResNet-50 model helps understand the mechanics of building a deep learning model from scratch. The models are evaluated using key performance metrics, and confusion matrices are visualized for improved interpretation of model performance

References:

1. Xception: Chollet, F. (2017). *Xception: Deep Learning with Depthwise Separable Convolutions*. CVPR.
2. DenseNet: Huang, G., et al. (2017). *Densely Connected Convolutional Networks*. CVPR.
3. ResNet50: He, K., et al. (2016). *Deep Residual Learning for Image Recognition*. CVPR.