**Virus project Documentation**

|  |  |
| --- | --- |
| ID | الاسم |
| 20211040 | يارا عماد درويش يوسف |
| 20211039 | ولاء اسامه عبدالقادر مصطفي |
| 20211028 | هادي محمد عبدالهادي |
| 20210076 | احمد عبد النبي عبد المعطي عبد العزيز |
| 20210464 | شهد احمد محمد مصطفي |
| 20210794 | محمد صلاح الدين عبد السميع عبد الجواد |

**Project Overview**

**Project title**: Virus classification

**Objective**: in this project, we train our dataset on 3 models to make classification for Viruses and then we compared the evaluation of the 3 models

**Data preprocessing**: resizing, normalizing(rescale)

**Models**: Resnet , Densnet, Xception

**Dataset Description**

**Dataset Details**:

* 3 classes:train (14 class, 10109 mg) ,validation (14 class, 2891 mg) , test (14 class, 1451img)
* Train classes visualization
* A graph showing a number of virus types

  Description automatically generatedA colorful pie chart with text

  Description automatically generated

**Steps of preprocessing**

1. load the dataset
2. data splitted already (train,test , validation)
3. resizing all the images
   1. we resize pictures to fit each model
      1. Xception (299,299)
      2. Denesnet (224,224)
      3. Resnet (128,128)
4. normalizing all the images
5. we use the augmented train split

**Project setup**

**Programming Language**: Python

**Frameworks**:

* TensorFlow
* Keras

**Libraries**:

* **NumPy**: For numerical operations.
* **Pandas**: For data manipulation and analysis (e.g., dataset handling).
* **Matplotlib** / Seaborn: For data visualization.
* **OpenCV**: For image processing.
* **Scikit**-**learn**: For machine learning utilities.

**Data Augmentation Tool**:

Keras ImageDataGenerator: Used for image data augmentation during training.

**Development Environment**:

Google Colab: Used as the primary environment for coding, data analysis, and model training.

**Models**

**ResNet**

ResNet (Residual Networks) revolutionized the deep learning field when its authors introduced it in their 2015 paper. The architecture made a significant impact by addressing the problem of training very deep networks, where earlier models faced degradation in performance as the number of layers increased. The core idea behind ResNet is the introduction of **residual connections**, or shortcut connections, which allow the model to learn identity mappings, making it easier to train deeper networks.

The key advantage of ResNet is its ability to train extremely deep models **without encountering vanishing or exploding gradient problems**, enabling architectures with hundreds or even thousands of layers. As a result, ResNet became a fundamental building block in many state-of-the-art deep learning models.

ResNet has several popular versions, such as **ResNet-18**, **ResNet-34**, **ResNet-50**, and others, each varying in depth and complexity. **The ResNet-34 model, with 34 layers, was chosen** for this implementation because it provides a good balance between complexity and performance, making it suitable for training from scratch while still achieving high accuracy.

**Architecture Breakdown**

The ResNet-34 model consists of several key components:

1. **Initial Convolutional Layer**:
2. The model starts with a standard convolutional layer designed to capture broad spatial features from the input images.
3. Filter Size: The convolution uses a large kernel size (typically 7x7), which helps detect coarse patterns like edges and textures across a wide receptive field.
4. Stride: A stride of 2 is used to reduce the spatial dimensions of the input, helping to make subsequent computations more efficient.
5. Batch Normalization: This layer normalizes the output of the convolution, stabilizing the learning process and accelerating convergence.
6. Activation Function: A ReLU (Rectified Linear Unit) activation is applied to introduce non-linearity, ensuring the model can learn complex patterns
7. **Residual Blocks**:

The model contains **4 stages** of residual blocks, where each stage consists of **2 residual blocks**:

* Stage 1: 64 filters, stride 1
* Stage 2: 128 filters, stride 2
* Stage 3: 256 filters, stride 2
* Stage 4: 512 filters, stride 2

Shortcut connections are added when dimensions mismatch, using Conv2D with kernel size 1x1

1. **Strides and Filters**:

Residual Blocks and their Parameters

The architecture includes 4 stages of residual blocks, each with a specified number of filters and stride values:

-Stage 1:Filters: 64

Stride: 1 (No down-sampling)

-Stage 2:Filters: 128

Stride: 2 (Down-sampling to reduce spatial dimensions)

-Stage 3:Filters: 256

Stride: 2 (Further down-sampling)

-Stage 4:Filters: 512

Stride: 2 (Final down-sampling to reduce spatial dimensions)

**Purpose of Strides and Filters**

Strides:

Stride = 1: Helps maintain spatial dimensions as much as possible for deeper feature learning.

Stride = 2: Reduces the spatial dimensions by half, helping to extract higher-level features and reduce computational complexity.

Filters:

Increasing the number of filters in each stage allows the network to capture more complex hierarchical features as depth increases.

Helps in progressively extracting finer and more abstract features, leading to better performance in complex tasks like image classification.

1. **Fully Connected Layer**: After passing through the residual blocks, the model ends with a global average pooling layer, which reduces the feature map to a single vector. The final block is a fully connected layer that maps the output vector to the desired number of classes.

Operations:

1- A Dense layer applies a linear transformation to map features to class probabilities.

2- A softmax activation ensures the output is a valid probability distribution

By choosing ResNet-34, we aimed to create a network that is deep enough to capture complex features while being computationally feasible to train from scratch. The combination of residual connections, convolutional layers with specific strides, and 3x3 filters ensures that the model performs well even on more challenging tasks.

The Architecture:

A screenshot of a computer

Description automatically generated

**Training Process**

The training process for the ResNet-34 model involved several key components, including the selection of appropriate hyperparameters and addressing challenges **and the time took to train was 1 and half hour** using TBU. Below, we outline each of these aspects:

**Hyperparameters**

1. **Input Shape**:  
   Images were resized to **128x128** pixels with **3 channels (RGB)**.
2. **Number of Classes**:  
   The model classifies into **14 distinct classes**.
3. **Optimizer**:  
   The model uses **SGD (Stochastic Gradient Descent)** with:
4. Learning Rate Scheduler: **Cosine Decay**
   1. **Initial Learning Rate**: 0.1
   2. **Decay Steps**: 1000
   3. **Alpha**: 0.01 (minimum learning rate)
5. **Momentum**: 0.9
6. **Loss Function**:  
   **Sparse Categorical Crossentropy**, which is ideal for classification tasks where labels are provided as integers.
7. **Metrics**:  
   The model tracks **accuracy** during training and validation

**Training Challenges**

During the training process, several challenges were encountered:

**Overfitting**

**Description:**

The model showed high accuracy on the training set, but the validation accuracy plateaued, indicating overfitting. This suggests that the model was learning to memorize the training data rather than generalizing effectively to unseen data.

**Reason:**

The dataset might be simple, lacking diverse color patterns or complex structures.

The ResNet architecture, while effective, may be overly complex for the given dataset, leading to overfitting.

Data augmentation was applied, but the improvements were minimal.

**Solution:**

Data augmentation techniques were used (e.g., image rotations, flips) to increase diversity and reduce overfitting.

Regularization methods like dropout or reducing model complexity were considered but did not yield significant improvement

**Pros:**

1. Addresses Vanishing Gradient Problem:

ResNet's skip (residual) connections help gradients flow more effectively through the network during backpropagation, allowing deeper networks to converge.

1. Supports Very Deep Architectures:

The residual learning framework allows networks with hundreds or even thousands of layers to be trained efficiently, leading to better feature representation.

1. Improved Accuracy:

ResNet consistently achieves state-of-the-art performance on various tasks like image classification (e.g., ImageNet), object detection, and more.

1. Transfer Learning Efficiency:

ResNet pre-trained models are widely used for transfer learning, providing a solid base for many downstream tasks.

1. Modularity and Flexibility:

ResNet can be easily extended to architectures like ResNeXt, SE-ResNet, or combined with attention mechanisms, making it adaptable to a variety of tasks

**Cons:**

1. **High Computational Cost and Memory Usage:**

Although more efficient than plain deep architectures, ResNet models (especially deeper ones like ResNet-152 or ResNet-101) require significant computational power and memory for both training and inference.

1. **Risk of Overfitting with Small Datasets:**

Very deep architectures like ResNet can overfit small datasets without proper regularization (e.g., dropout, data augmentation).

1. **Requires Large Datasets for Training:**

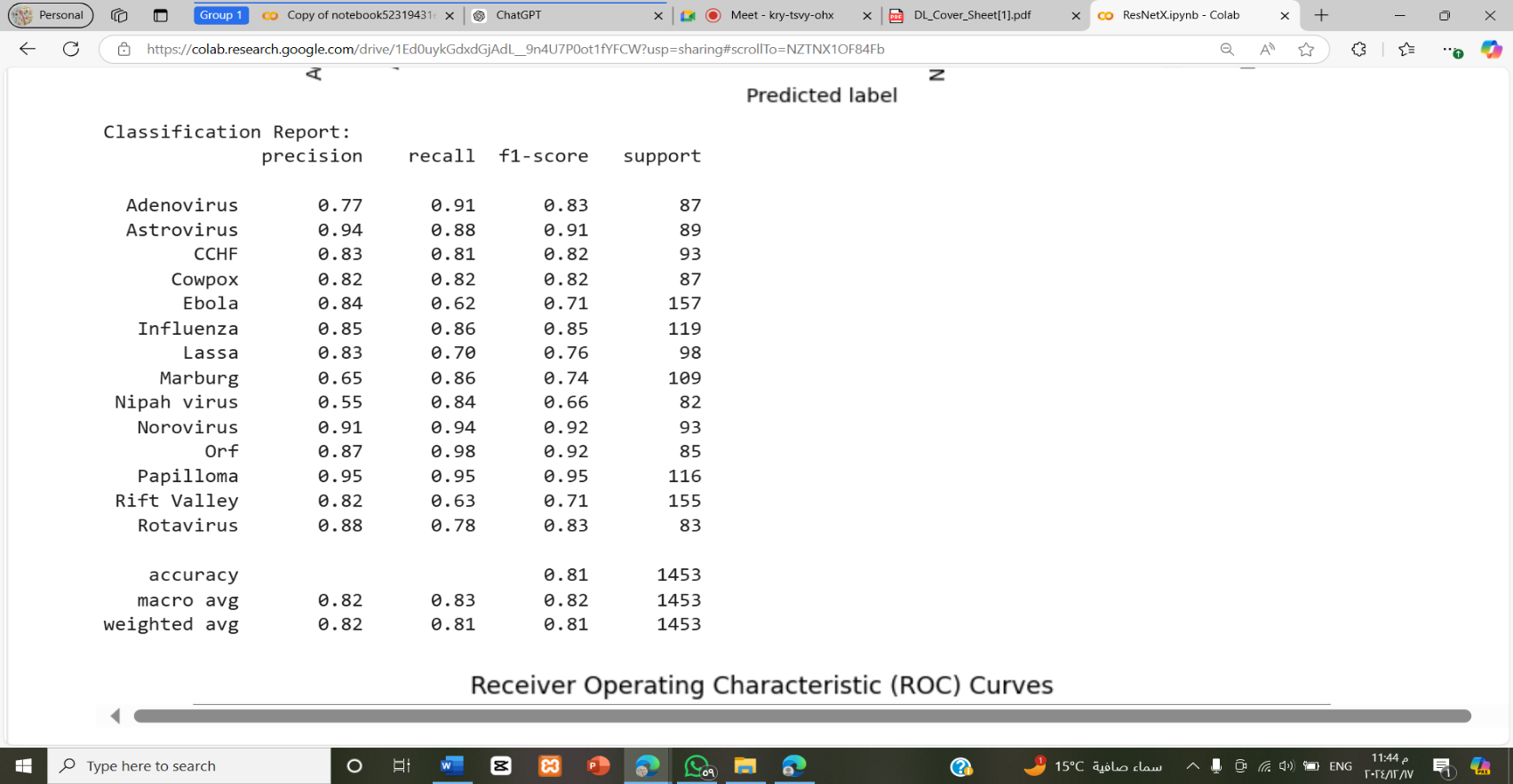
Training ResNet from scratch demands a large amount of labeled data to achieve good performance. Transfer learning mitigates this but is not always applicable.

1. **Diminishing Returns with Depth:**

Beyond a certain depth (e.g., 1000+ layers), the performance gains become marginal, even with residual connections.

1. **Optimization Challenges:**

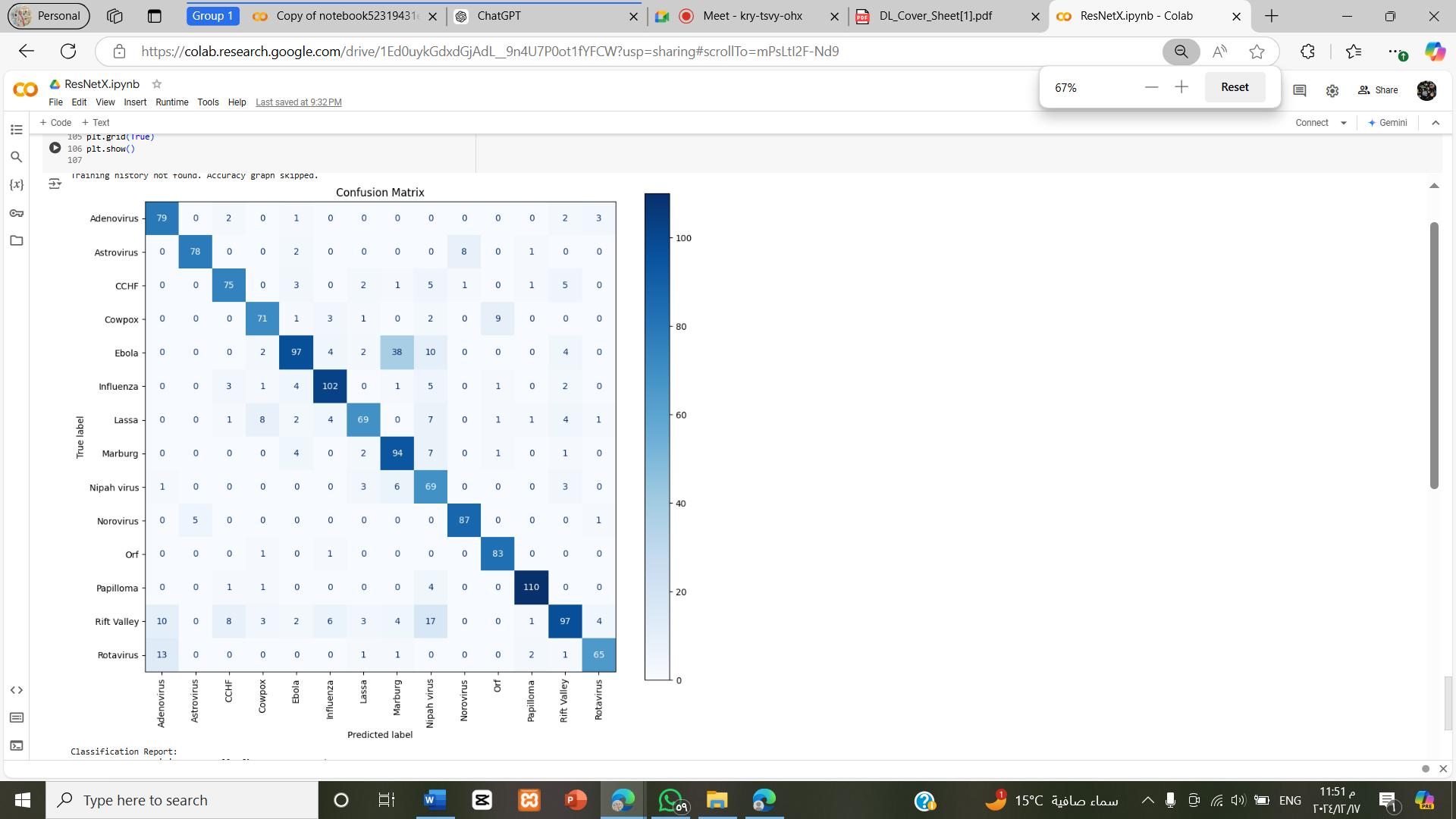
Though ResNet addresses the vanishing gradient problem, training very deep networks can still be sensitive to hyperparameters, learning rate schedules, and initialization.

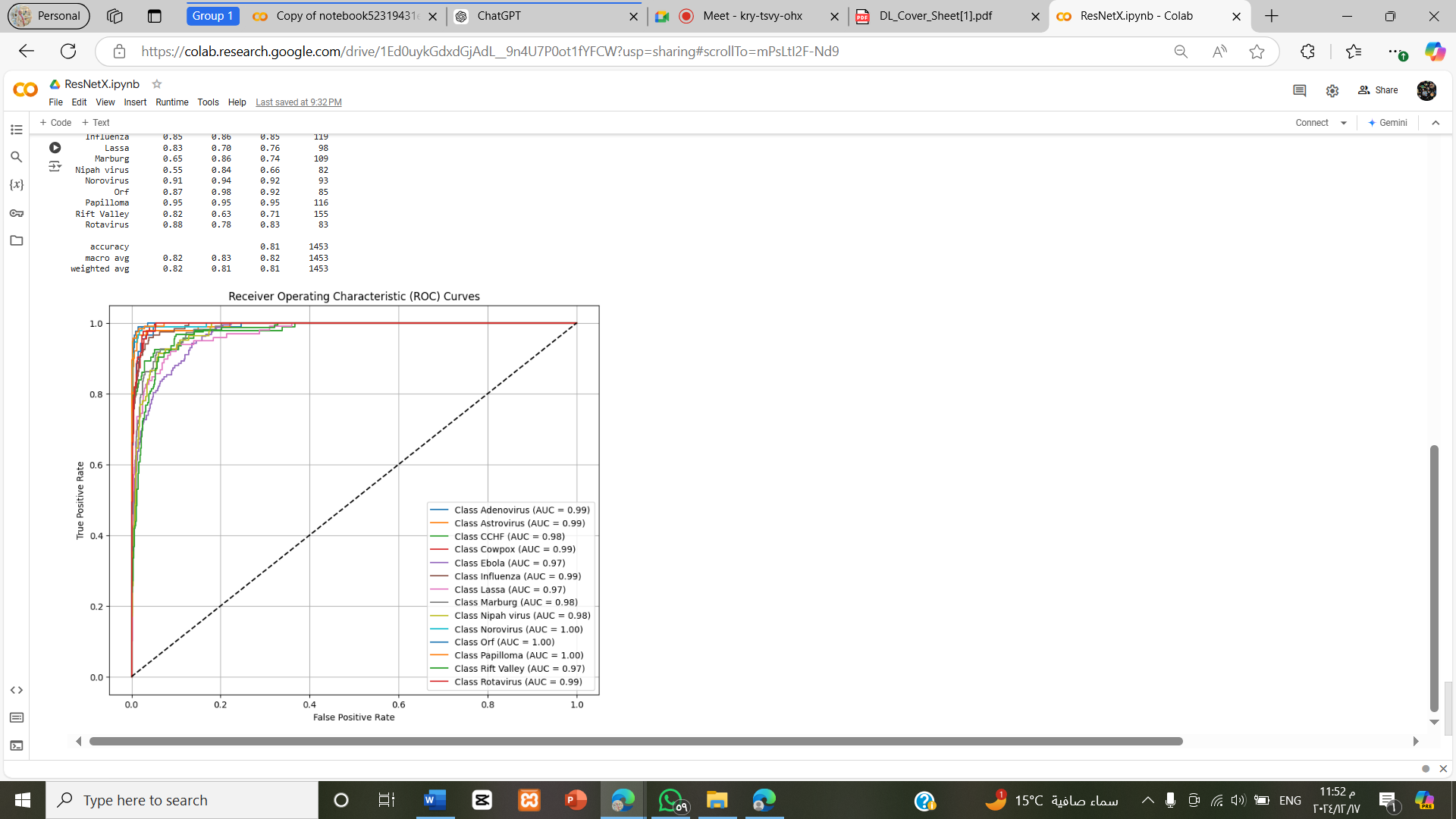
****

**Evaluation  
accuracy and classification Report**

**-Test Loss: 69%**

**-Test Accuracy: 78%**

****

**ROC AUC**

**2-XceptionModel**

**Introduction**

This model is designed to perform image classification of virus images using **TensorFlow** and the **Xception model**. It includes steps for:

* Dataset loading and preprocessing,
* Building a classification model using the Xception architecture,
* Training the model, and
* Evaluating performance.

The notebook is executed on **Google Colab** with integration to Google Drive for dataset management

**Architecture**

1. **Depthwise Separable Convolutions**: Xception leverages depthwise separable convolutions, which decompose a standard convolution into:
   * A depthwise convolution: Applies a single filter per input channel.
   * A pointwise convolution: Combines the outputs of depthwise convolutions.
2. **Linear Stack**: Unlike Inception, Xception arranges layers in a straightforward sequential structure, simplifying implementation.
3. **Global Average Pooling**: Removes fully connected layers and replaces them with a global average pooling layer for better generalization and reduced overfitting.

**Pros**

* **Efficient Computations**: Depthwise separable convolutions significantly reduce the computational cost compared to traditional convolutions.
* **High Performance**: Demonstrates strong results across various classification and detection tasks.
* **Simplified Design**: Replaces the complex Inception modules with a clean, modular architecture.

**Cons**

* **Sensitive to Hyperparameters**: Performance heavily depends on proper initialization and tuning. Common hyperparameters to consider include the learning rate, weight decay, batch size, and the number of epochs. Additionally, optimizer selection (e.g., Adam or SGD) and dropout rate may require fine-tuning to achieve optimal results.

**Data Preprocessing with ImageDataGenerator**

**Purpose: Use ImageDataGenerator for data augmentation and image rescaling**

**A screenshot of a computer

Description automatically generated**

**Build Custom Model**

**Add custom layers on top of the base model for classification**

A screenshot of a computer

Description automatically generated

**GlobalAveragePooling2D: Reduces feature maps to a single vector.**

**Dropout(0.5): Prevents overfitting.**

**Dense(4, activation='softmax'): Final layer with softmax activation for multi-class classification.**

**Model Training**

-Configure the model for training

**A screenshot of a computer

Description automatically generated**

-Fit the model to the training data

**A screenshot of a computer

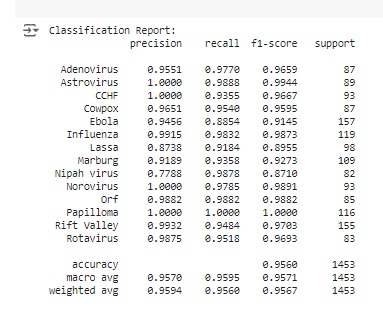
Description automatically generated**

**Plot Training History**

**-**Visualize accuracy and loss over epochs

A screenshot of a computer

Description automatically generated

* **Training Complexity**: Requires careful data preprocessing and augmentation for optimal results.
* **Less Robust for Small Datasets**: May overfit when used on small datasets without proper regularization.
* **Evaluation  
  accuracy and classification Report**

Test:95.60%

Validation:96.54%

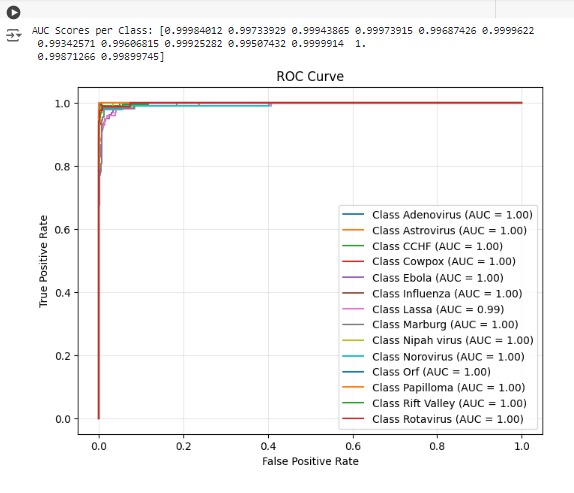
A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated

A graph with numbers and a number of classes

Description automatically generated with medium confidence

ROC AUC



**3-DenseNet Model**

**Overview**

DenseNet (Densely Connected Convolutional Network) is a neural network architecture introduced by Huang et al. in 2017. DenseNet has shown notable advantages in various domains, particularly in medical imaging, where its efficient feature reuse and strong gradient flow have been leveraged for tasks such as disease diagnosis and segmentation. It is also well-suited for other applications requiring detailed pattern recognition, such as remote sensing and biometrics. It

connects each layer to every other layer in a dense block, promoting feature reuse and efficient gradient flow.

**Architecture**

1. **Dense Connections**: Each layer receives the feature maps of all preceding layers as input, ensuring maximum information flow. This feature is particularly beneficial for tasks with limited data or high-dimensional inputs, as it allows the model to efficiently utilize features from all layers and improves generalization.
2. **Dense Blocks**: Groups of densely connected layers where each layer outputs a fixed number of feature maps (“growth rate”).
3. **Transition Layers**: Reduces dimensionality using convolutions and pooling layers, controlling model complexity.
4. **Global Average Pooling**: Like Xception, it replaces fully connected layers with global average pooling.

**Types of DenseNet**

DenseNet comes in several variants based on the number of layers and complexity:

1. **DenseNet-121**: A lightweight version with 121 layers, suitable for tasks requiring a balance between accuracy and computational efficiency.
2. **DenseNet-169**: Deeper than DenseNet-121, with 169 layers, providing improved performance but requiring more resources.
3. **DenseNet-201**: Offers 201 layers, designed for tasks where higher accuracy is prioritized over efficiency.
4. **DenseNet-264**: The deepest standard version with 264 layers, delivering top-tier accuracy but demanding significant computational and memory resources.

**Pros**

* **Efficient Feature Usage**: Reuse of features allows DenseNet to achieve better performance with fewer parameters.
* **Improved Gradient Flow**: Dense connections mitigate the vanishing gradient problem.
* **Compact Model**: Reduces the risk of overfitting due to efficient parameter usage.

**Cons**

* **High Memory Usage**: Dense connections require storing feature maps from all previous layers, increasing memory demand. To mitigate this, strategies such as gradient checkpointing, mixed precision training, or reducing the batch size can be employed to optimize memory usage during training.
* **Slower Training**: Dense connectivity increases computation time during training.
* **Not Suitable for Very Large Datasets**: The computational cost may become prohibitive for extremely large datasets.

**Model Training:**

**Configure the model for training**

A screenshot of a computer

Description automatically generated

-Fit the model to the training data

A screenshot of a computer

Description automatically generated

**Plot Training History**

**-**Visualize accuracy and loss over epochs

A screenshot of a computer

Description automatically generated

**Data Preprocessing with ImageDataGenerator** A screenshot of a computer

Description automatically generated

**Model Building**

**A screenshot of a computer

Description automatically generated**

weights='imagenet': Uses pre-trained weights.

include\_top=False: Excludes the fully connected layers for customization.

Freezes the base model to retain pre-trained features

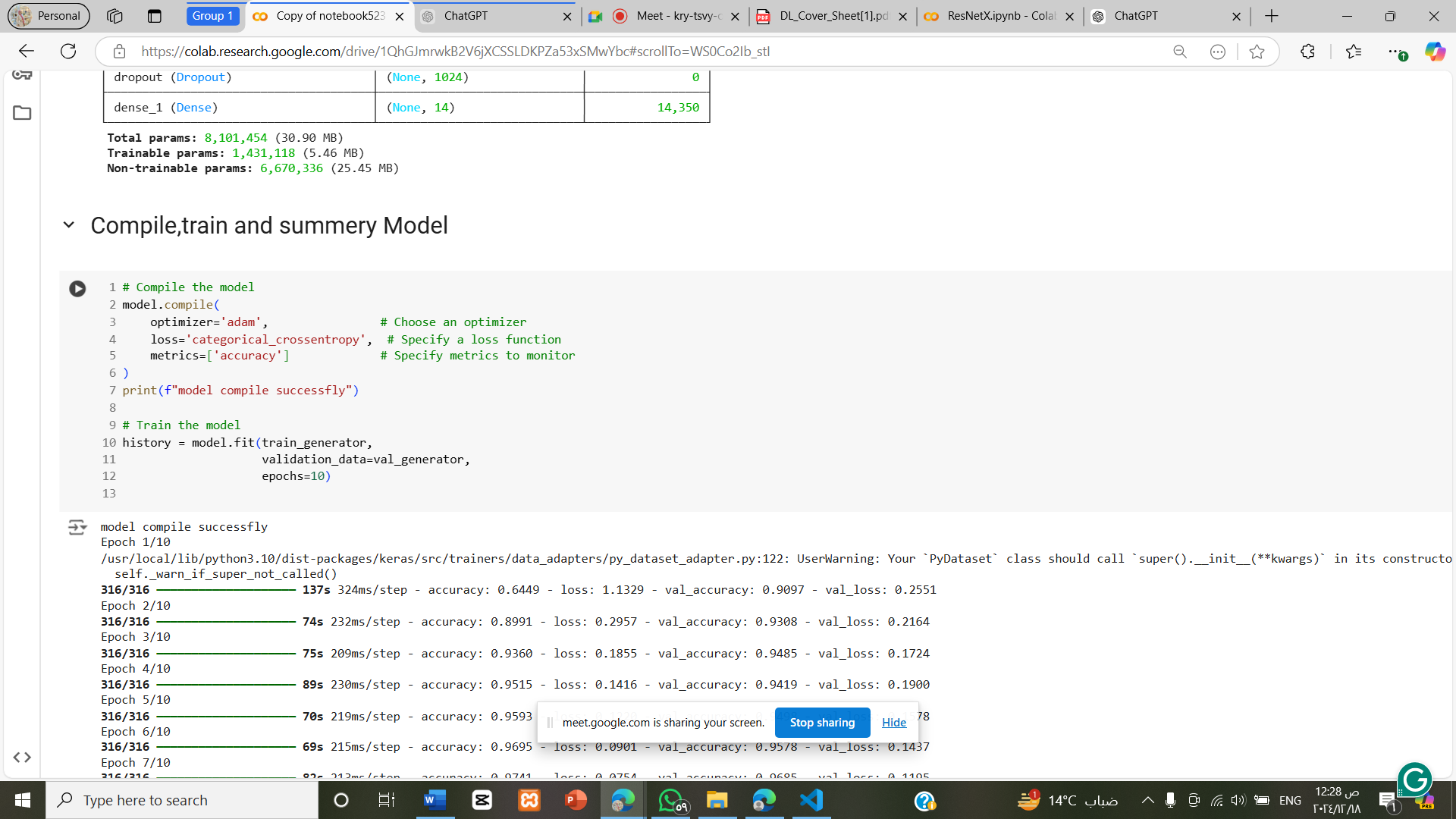
GlobalAveragePooling2D: Reduces feature maps to a single vector.

Dropout (0.5): Prevents overfitting.

Dense (4, activation='SoftMax'): Final layer with SoftMax activation for multi-class classification

Unfreeze the last 20 layers

**Compile Model**

****

with 10 epochs for fit model

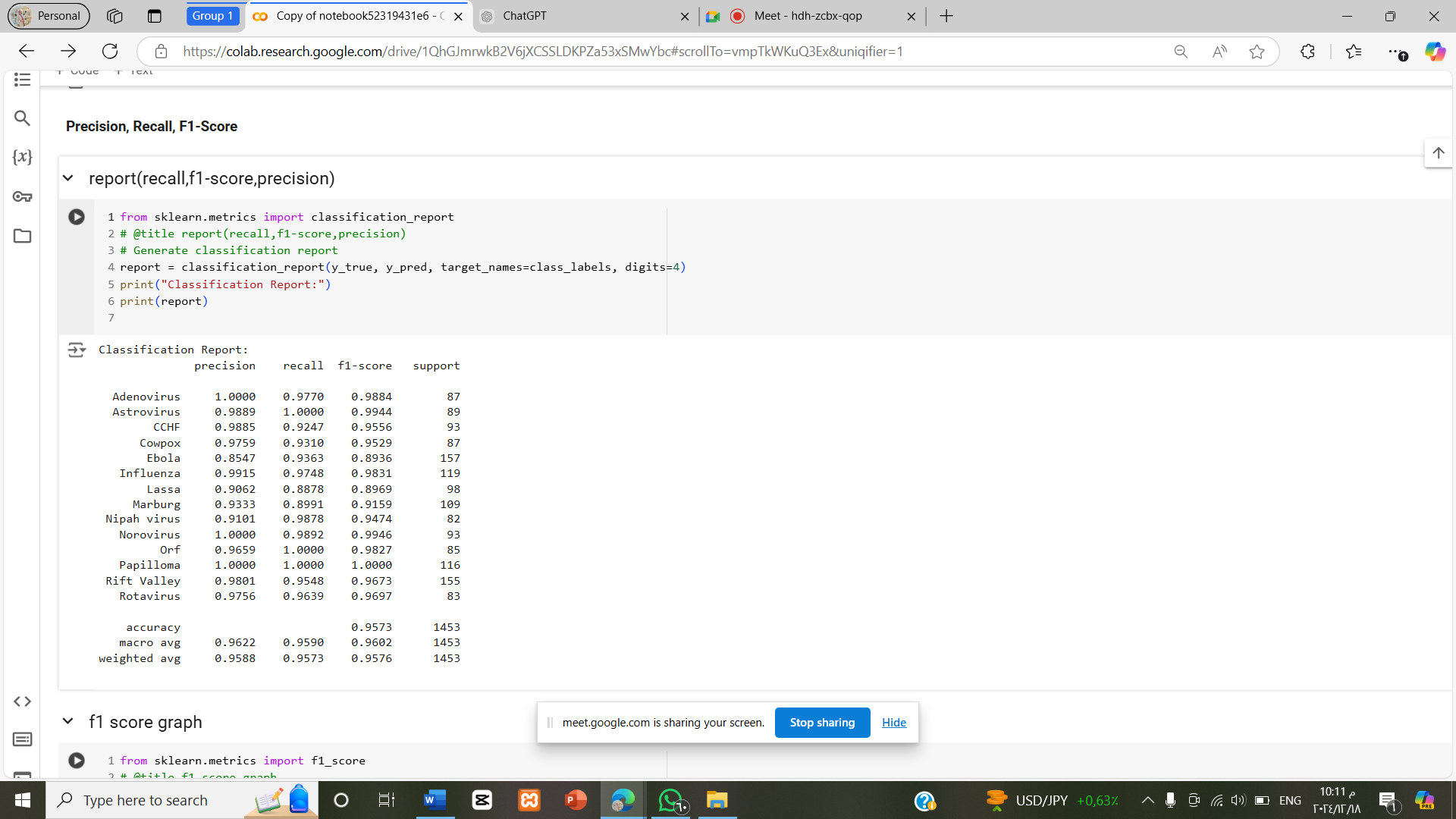
**Evaluation  
accuracy and classification Report,Graphics**

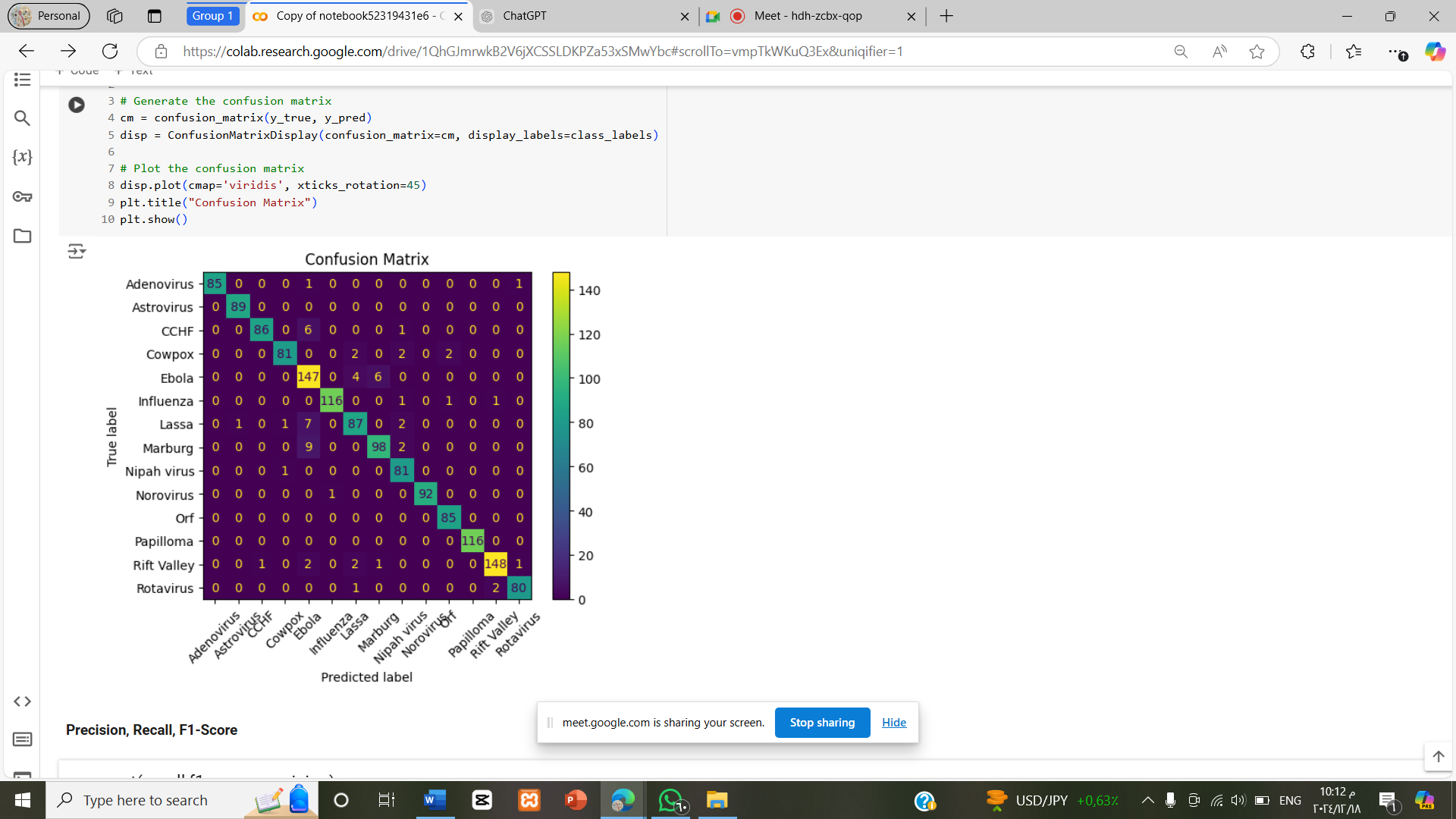
**Test acc: 95.73%**

**Validation acc:97.13%**

A screenshot of a computer

Description automatically generated





Roc curve:A screenshot of a computer

Description automatically generated

**Papers links**

**Resnet18 Paper Ref https**:[**https://paperswithcode.com/paper/resnet18-model-with-sequential-layer-for**](https://paperswithcode.com/paper/resnet18-model-with-sequential-layer-for)

**Densnet121 Paper Reference:** [**https://paperswithcode.com/lib/torchvision/densenet**](https://paperswithcode.com/lib/torchvision/densenet)

**Xception Paper:** [**https://paperswithcode.com/model/xception?variant=xception-1**](https://paperswithcode.com/model/xception?variant=xception-1)

**Comparison Of 3 Models:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **DenseNet** | **Xception** | **ResNet** |
| **Test Accuracy** | 95.73% | 95.60% | 78% |
| **Validation Acc** | 97.13% | 96.54% | Lower than DenseNet and Xception |
| **Computational Efficiency** | Moderate | High (lightweight) | Low |
| **Scalability** | High | Moderate | High (for very deep networks) |
| **Ease of Implementation** | Moderate | Easy | Moderate |
| |  | | --- | | **Task Flexibility** | | High | Moderate | Moderate |
| |  | | --- | | **Dataset Suitability** | | Small-to-Medium | Large | Large |
| **pros** | 1. **Performance:** Highest test accuracy (95.73%) and macro F1-score (96.02%). 2. **Efficient Feature Propagation:** Dense connections reuse learned features, improving gradient flow. 3. **Compact Architecture:** Achieves high performance with fewer parameters compared to ResNet. 4. **Task Flexibility:** Versatile for small-to-medium image datasets and medical diagnostics. 5. **Generalization:** High generalization due to efficient feature usage | 1. **Computational Efficiency:** Lightweight due to depthwise separable convolutions, reducing computational overhead. 2. **Performance:** Test accuracy (95.60%) close to DenseNet, with a high macro F1-score. 3. **Flexibility:** Ideal for image classification, object detection, and mobile applications. 4. **Ease of Training:** Efficient and faster to train than DenseNet or ResNet. | 1. **Residual Connections:** Solves vanishing gradient issues, enabling deep network training. 2. **Transfer Learning:** Pretrained ResNet models are widely available and highly effective for transfer learning. 3. **Scalability:** Suitable for scaling up to very deep architectures (e.g., ResNet-152 |
| **cons** | 1. **Memory Intensive:** Dense connections require higher memory, which can limit scalability to very deep networks. 2. **Complex Implementation:** Dense connectivity introduces additional implementation challenges | 1. **Sensitivity:** Requires careful hyperparameter tuning for optimal performance. 2. **Less Suitable for Smaller Datasets:** Performance may drop on tasks with limited data. 3. **Limited Use Cases:** Optimized mainly for image-related tasks; not as flexible as DenseNet for other domains | 1. **Performance Drop:** Significantly lower accuracy (78%) compared to DenseNet and Xception. 2. **Overfitting Risk:** Prone to overfitting on smaller datasets. 3. **Computational Inefficiency:** Deeper architectures require substantial computational resources. 4.  **Suboptimal Gradient Flow:** Despite residual connections, struggles with very complex datasets |