Documentation: Deep Learning Project

Contents

1 PlantVillage Dataset

The PlantVillage dataset consists of 54303 healthy and unhealthy leaf images divided into 39 categories by species and disease. It was introduced in the 2015 paper: 'An open access repository of images on plant health to enable the development of mobile disease diagnostics'[5].

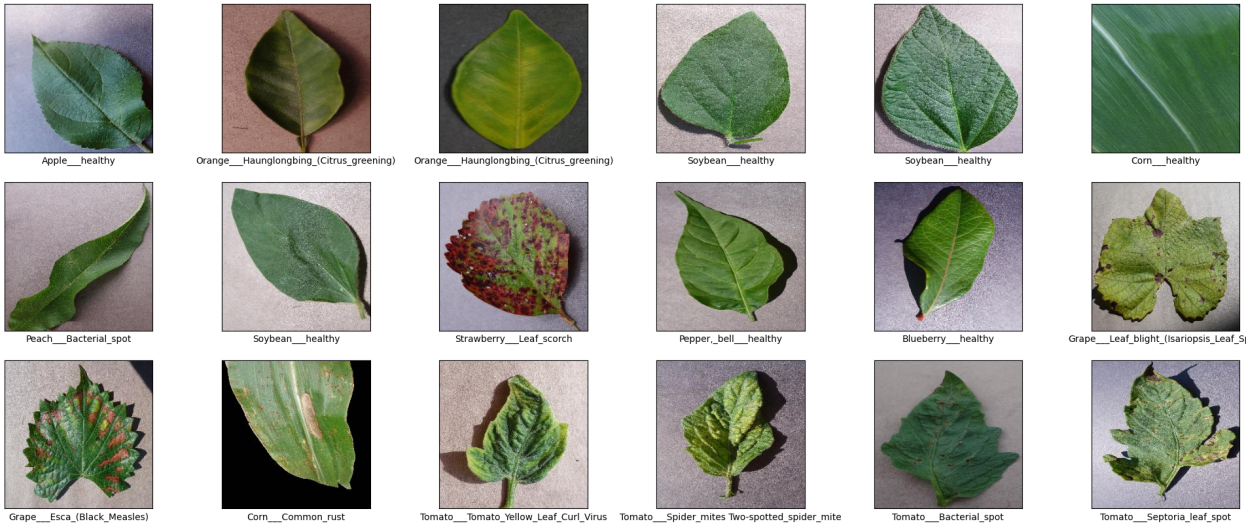

Figure 1: Sample from the plant village dataset.

1.1 Preprocessing

Resizing and Normalization

Each image $\mathbf{I} \in \mathbb{R}^{M \times N \times R}$ is resized in order to standardize the spatial dimensions of the input data: $\mathbf{I}_{\text{resized}} \in \mathbb{R}^{244 \times 244 \times 3}$.

After resizing we apply normalization to scale and center the image pixel values. Normalization is performed on each image layer c based on predefined mean μ_c and standard deviation σ_c values:

$$\mathbf{I}_{\text{norm}}(i, j, c) = \frac{\mathbf{I}_{\text{resized}}(i, j, c) - \mu_c}{\sigma_c}. \tag{1}$$

The specific normalization parameters used were:

$$\mu = [0.485, 0.456, 0.406],$$

$$\sigma = [0.229, 0.224, 0.225],$$

Thus, the normalization for each channel is explicitly given by:

$$\mathbf{I}_{\text{norm}}(i, j, 1) = \frac{\mathbf{I}_{\text{resized}}(i, j, 1) - 0.485}{0.229} \quad \text{(Red channel)}$$

$$\mathbf{I}_{\text{norm}}(i, j, 2) = \frac{\mathbf{I}_{\text{resized}}(i, j, 2) - 0.456}{0.224} \quad \text{(Green channel)}$$

$$\mathbf{I}_{\text{norm}}(i, j, 3) = \frac{\mathbf{I}_{\text{resized}}(i, j, 3) - 0.406}{0.225} \quad \text{(Blue channel)}$$

1.2 Dataset Splitting Procedure

The complete dataset consists of a collection of image file paths, with the label for a given image being included in it's path as it's parent directory (e.g. '`dataset/Apple__Apple_scab/image(101).JPG` belongs to class 'Apple - Apple scab').

We partition the dataset into a training, validation, and testing set. We distribute the data between the splits follows: `train` (70%), `val` (15%), `test` (15%). In terms of number of samples: `train` (38,813), `val` (8,317), `test` (8,318).

The dataset splitting procedure is stratified, meaning that the procedure ensures the class label distributions in each subset/split is the same. We also verify this by plotting the distributions for each split (*see Figure 2*).

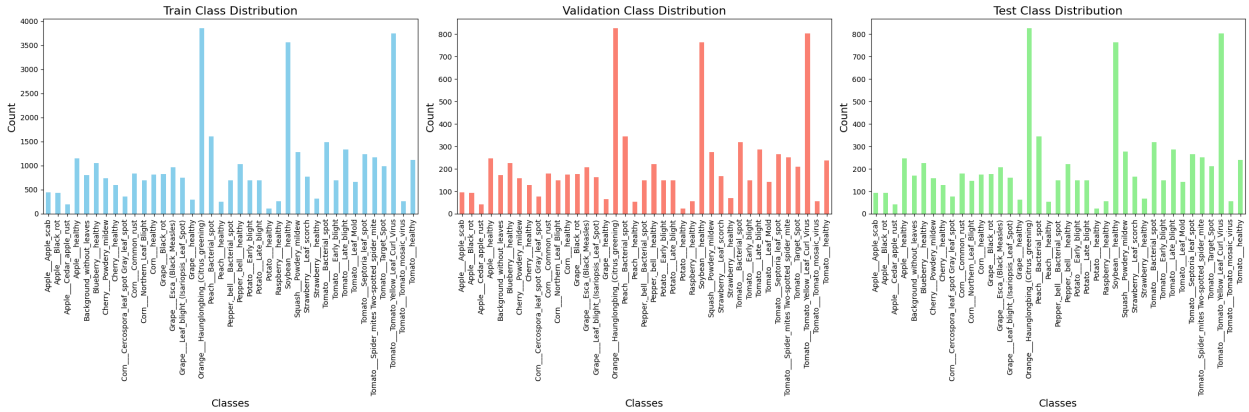

Figure 2: Plot of the ditribution of class occurences in each of the dataset splits. The distributions are almost identical, indicating a successful stratification during the splitting process.

2 DenseNets: Densely Connected Convolutional Networks

The DenseNet architecture was outlined in the 2017 paper: 'Densely Connected Convolutional Networks'[1], densely connected networks aimed to address the problem of gradient and input vanishing that is associated with deep convolutional neural networks (CNNs)

The proposed Densenet architecture echos a charactersitic present in other CNN papers released around that time (ResNets[2], Highway Networks[3], Fractal Networks[4]): *creating short/bypassing paths from early layers to later layers.*

2.1 Model Architecture

Dense Connectivity

The architecture comprises L layers, each of which implementing a non-linear transformation H_l, where l indexes the layer. H_l is a composite function of three consecutive operations*: batch normalization (BN), followed by a rectified linear unit (ReLU) and a 3×3 convolution (Conv). The output of the layer l is denoted $\mathbf{x}_l$.

In a traditional CNN the output of the l^{th} layer is connected as the input of the $l + 1^{th}$ layer:

$$\mathbf{x}_l = H_l(\mathbf{x}_{l-1} + \mathbf{x}_{l-1}). \tag{2}$$

In a DenseNet the l^{th} layer instead **receives the feature-maps of all preceding layers**, $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{l-1}$, as inputs:

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{l-1}]), \tag{3}$$

where $[\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{l-1}]$ refers to the **concatentation**** of the feature maps produced in the layers $0, \cdots, l-1$.

This novel layer connection scheme referred to as *dense connectivity* by the authors is what defines the DenseNet architecture. This connectivity aims to provide maximum *information flow* between layers in the network.

* The authors describe a variant of H_l that has 4 operations instead of 3, with a *bottleneck layer*, a 1×1 convolution, preceding the 3×3 convolution. This variant is referred to as DenseNet-B.

** The concatenation operation is preceded by a pooling operation to allow for different sized feature-maps to be concatenated. See 'Modularity' section for more details.

Growth Rate

Each function H_l produces k feature maps, it follows from this that the l^{th} layer has $k_0 + k \times (l-1)$ input feature-maps, where k_0 is the number of channels in the input layer. The hyperparameter k is referred to as the *growth rate* of the network. In simpler terms k the number of new feature maps each application of H produces, which is essentially the depth of the output feature map $\mathbf{x}$.

Modularity

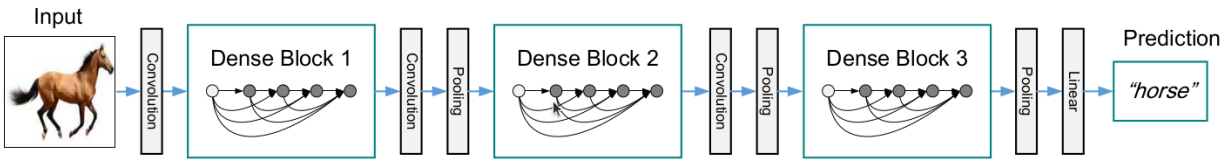

Figure 3: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

The network is divided into multiple densely connected *dense blocks*. Between each block are *transition layers*, which perform convolution and pooling. The transition layers consist of a batch normalization layer and a 1×1 convolutional layer followed by a 2×2 average pooling layer.

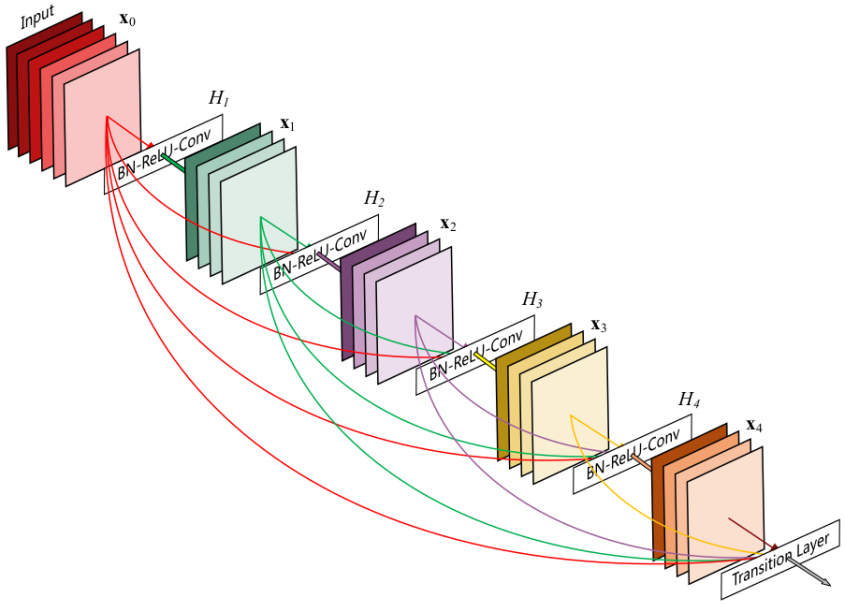

Figure 4: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

2.2 Implementations

Code and pre-trained models are available at `https://github.com/liuzhuang13/DenseNet`.

Popular deep learning libraries like Pytorch and Keras also have their own implementations of DenseNet variants.

2.3 Transfer Learning with DenseNets

Base Model

We use DenseNet-201 for our implementation. DenseNet-201 is available as part of the Keras library, and can be loaded directly with the pretrained model weights for the 'ImageNet' dataset. We load in DenseNet-201 without it's linear classifier or *top*. The DenseNet model in this implementation is frozen (it's parameters are not trained) and essentially acts as a feature extractor.

The DenseNet201 model (without classifier) has input dimensions of `(None, 224, 224, 3)`, where the `None` is a stand in for the batch size, and output dimensions of `(None, 7, 7, 1920)`.

Classifier

We receive a feature map of dimension `(None, 7, 7, 1920)` from the base model, we use global average pooling to reduce the feature map to `(None, 1920)`. This feature map is then fed into a fully connected dense layer with 39 units and softmax activation to map the features to the 39 classes in our dataset.

Overall Model

Layer (type)	Output Shape	Param #
`densenet201` (`Functional`)	(None, 7, 7, 1920)	18,321,984
`global_average_pooling2d` (`GlobalAveragePooling2D`)	(None, 1920)	0
`dense` (`Dense`)	(None, 39)	74,919

Training Procedure

The model is trained for 20 epochs, using the Adam optimizer with initial learning rate 0.00001. The loss function used is categorical cross entropy.

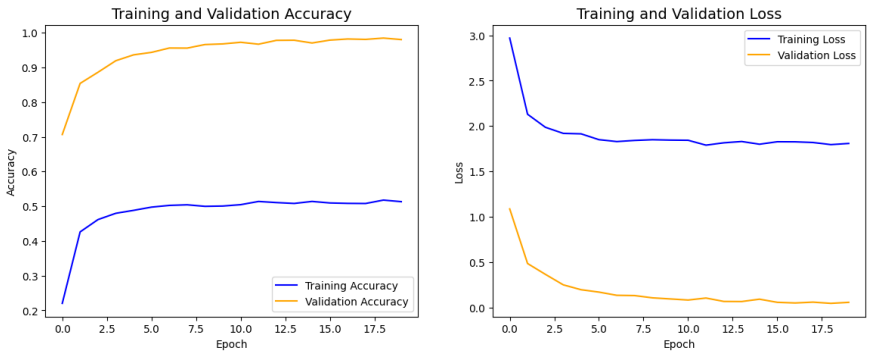

Figure 5: Training history visualization. Accuracy and loss plotted over epochs for both the training and validation subsets.

Evaluation

Table 1: Classification Report

Class	Precision	Recall	F1-Score	Support
Apple___Apple_scab	1.00	1.00	1.00	94
Apple___Black_rot	1.00	1.00	1.00	93
Apple___Cedar_apple_rust	1.00	0.98	0.99	41
Apple___healthy	0.98	1.00	0.99	247
Background_without_leaves	0.99	0.99	0.99	171
Blueberry___healthy	0.99	1.00	0.99	226
Cherry___Powdery_mildew	1.00	0.98	0.99	158
Cherry___healthy	0.98	1.00	0.99	128
Corn___Cercospora_leaf_spot Gray_leaf_spot	0.89	0.99	0.94	77
Corn___Common_rust	0.98	0.99	0.99	179
Corn___Northern_Leaf_Blight	0.99	0.90	0.95	147
Corn___healthy	0.99	1.00	1.00	174
Grape___Black_rot	0.97	1.00	0.99	177
Grape___Esca_(Black_Measles)	1.00	0.98	0.99	207
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1.00	1.00	1.00	161
Grape___healthy	1.00	1.00	1.00	63
Orange___Haunglongbing_(Citrus_greening)	1.00	0.98	0.99	826
Peach___Bacterial_spot	1.00	1.00	1.00	345
Peach___healthy	1.00	1.00	1.00	54
Pepper,_bell___Bacterial_spot	0.99	0.95	0.97	149
Pepper,_bell___healthy	0.99	0.98	0.98	222
Potato___Early_blight	0.88	1.00	0.93	150
Potato___Late_blight	0.98	0.94	0.96	150
Potato___healthy	0.77	1.00	0.87	23
Raspberry___healthy	1.00	1.00	1.00	55
Soybean___healthy	1.00	1.00	1.00	764
Squash___Powdery_mildew	1.00	1.00	1.00	276
Strawberry___Leaf_scorch	0.99	1.00	1.00	166
Strawberry___healthy	1.00	1.00	1.00	68
Tomato___Bacterial_spot	0.99	0.94	0.96	319
Tomato__Early_blight	0.99	0.91	0.94	150
Tomato__Late_blight	0.97	0.97	0.97	287
Tomato__Leaf_Mold	1.00	0.94	0.97	143
Tomato__Septoria_leaf_spot	0.95	0.97	0.96	266
Tomato__Spider_mites Two-spotted_spider_mite	0.95	0.96	0.95	252
Tomato__Target_Spot	0.87	0.95	0.90	211
Tomato__Tomato_Yellow_Leaf_Curl_Virus	1.00	0.99	0.99	804
Tomato__Tomato_mosaic_virus	1.00	1.00	1.00	56
Tomato__healthy	0.96	1.00	0.98	239
Accuracy			0.98	8318
Macro Avg	0.98	0.98	0.98	8318
Weighted Avg	0.98	0.98	0.98	8318

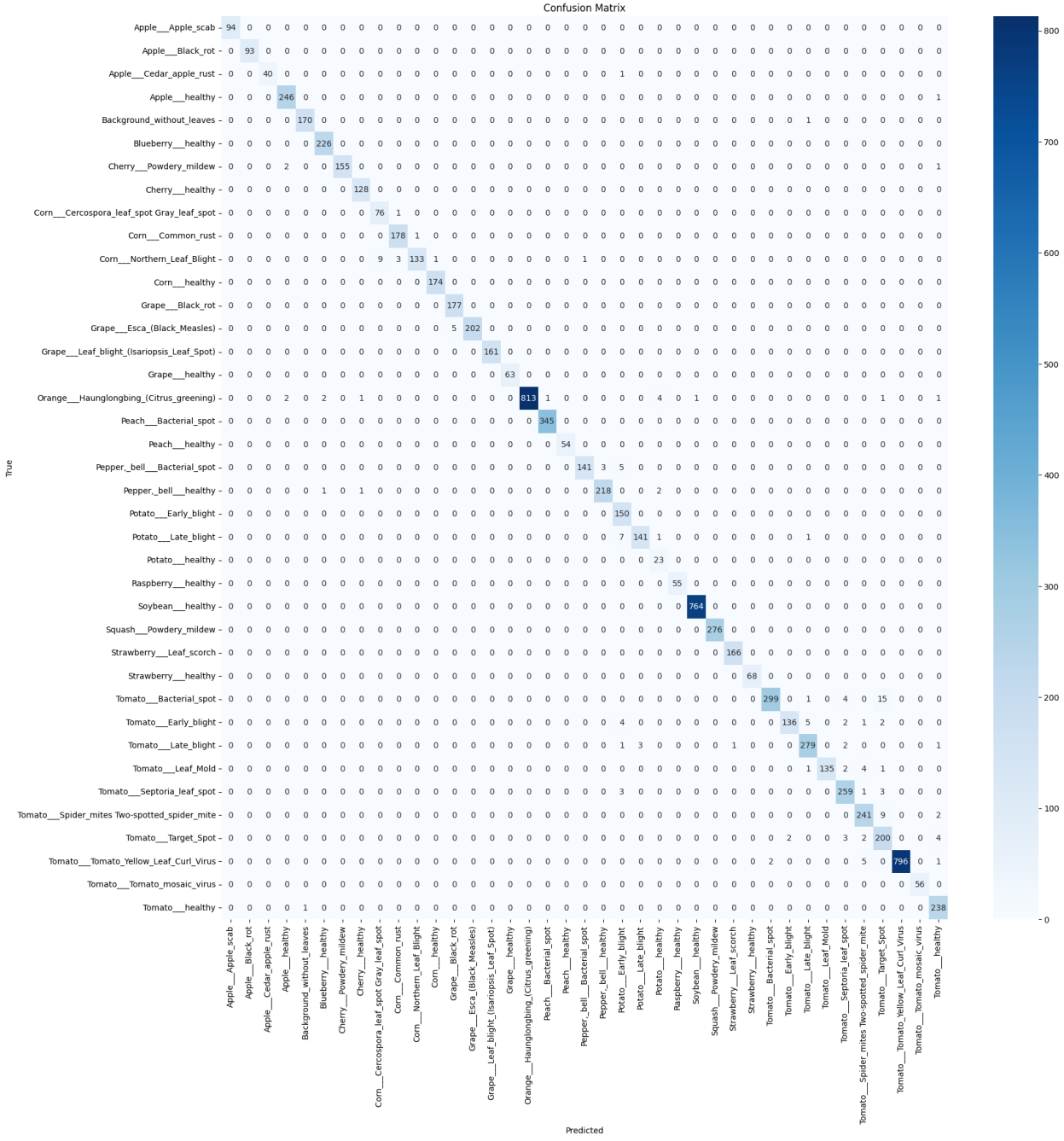

Figure 6: Confusion Matrix.

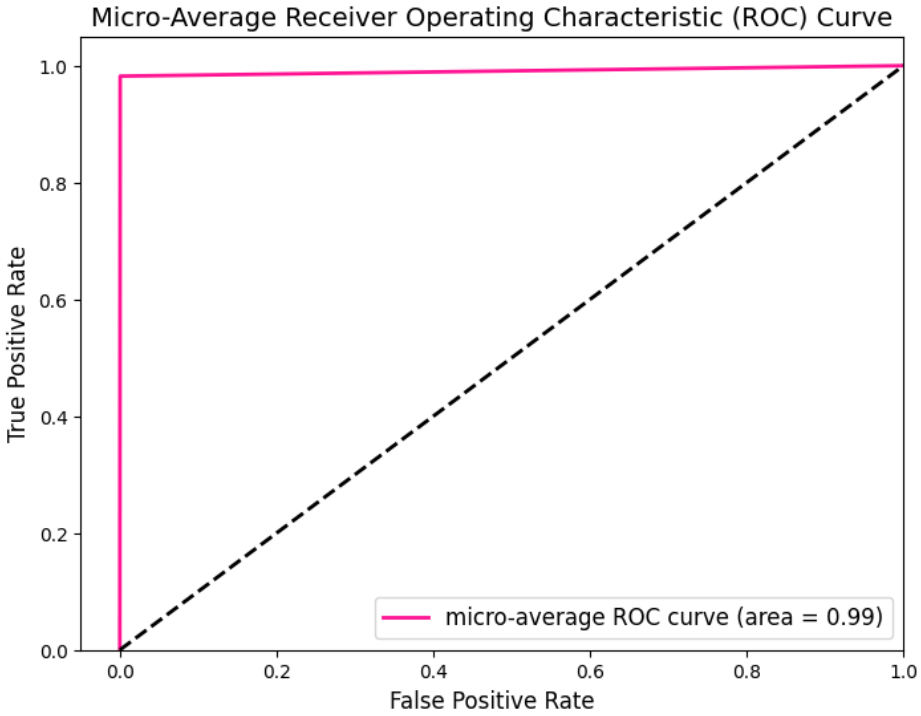

Figure 7: ROC Curve.

3 Xception: Deep Learning with Depthwise Separable Convolutions

The Xception architecture was introduced in the 2017 paper: 'Xception: Deep Learning with Depthwise Separable Convolutions' by François Chollet [1]. Xception stands for *Extreme Inception* and builds upon the foundations laid by the Inception architectures, proposing a more efficient and performant design by completely decoupling spatial and cross-channel correlations in feature maps.

3.1 Model Architecture

Depthwise Separable Convolutions

A depthwise separable convolution splits the traditional convolution operation into two steps:

1. *Depthwise Convolution*: Spatial convolution performed independently on each input channel.
2. *Pointwise Convolution*: A 1×1 convolution that projects the output of the depthwise convolution onto a new channel space.

More formally, the the depthwise convolution applies a filter $\mathbf{K}c$ of size $D_k \times D_k$ to each input channel c, producing:

$$\mathbf{Y}c = \mathbf{X}_c * \mathbf{K}c, \tag{4}$$

and the pointwise convolution applies a 1×1 filter $\mathbf{W}$ across all channels:

$$\mathbf{Z}c = \mathbf{Y} \cdot \mathbf{W}. \tag{5}$$

This design drastically reduces computational cost compared to standard convolutions, as the number of parameters and multiplications decreases significantly. For a convolutional layer with N input channels, M output channels, and kernel size $D_k \times D_k$, the number of parameters in a depthwise separable convolution is:

$$D_k^2 \cdot N + N \cdot M, \tag{6}$$

compared to the standard convolution:

$$D_k^2 \cdot N \cdot M. \tag{7}$$

Linear Residual Connections

Xception incorporates residual connections[2] for most convolutional blocks. This design helps prevent vanishing gradients and accelerates training. These connections can be described as:

$$\mathbf{F}(\mathbf{X}) + \mathbf{X}, \tag{8}$$

where $\mathbf{F}(\mathbf{X})$ represents the output of the depthwise separable convolution layers, and $\mathbf{X}$ is the input to the block. These residual connections exist for all modules except the first and last ones in the architecture.

Three-Flow Design

The architecture comprises three main flows, as depicted in Figure 8:

- **Entry Flow**: Consists of standard and depthwise separable convolutions, reducing spatial dimensions while increasing feature depth.
- **Middle Flow**: A linear stack of eight modules, each containing depthwise separable convolutions with residual connections.
- **Exit Flow**: Aggregates features via depthwise separable convolutions, followed by a global average pooling layer and a softmax classifier.

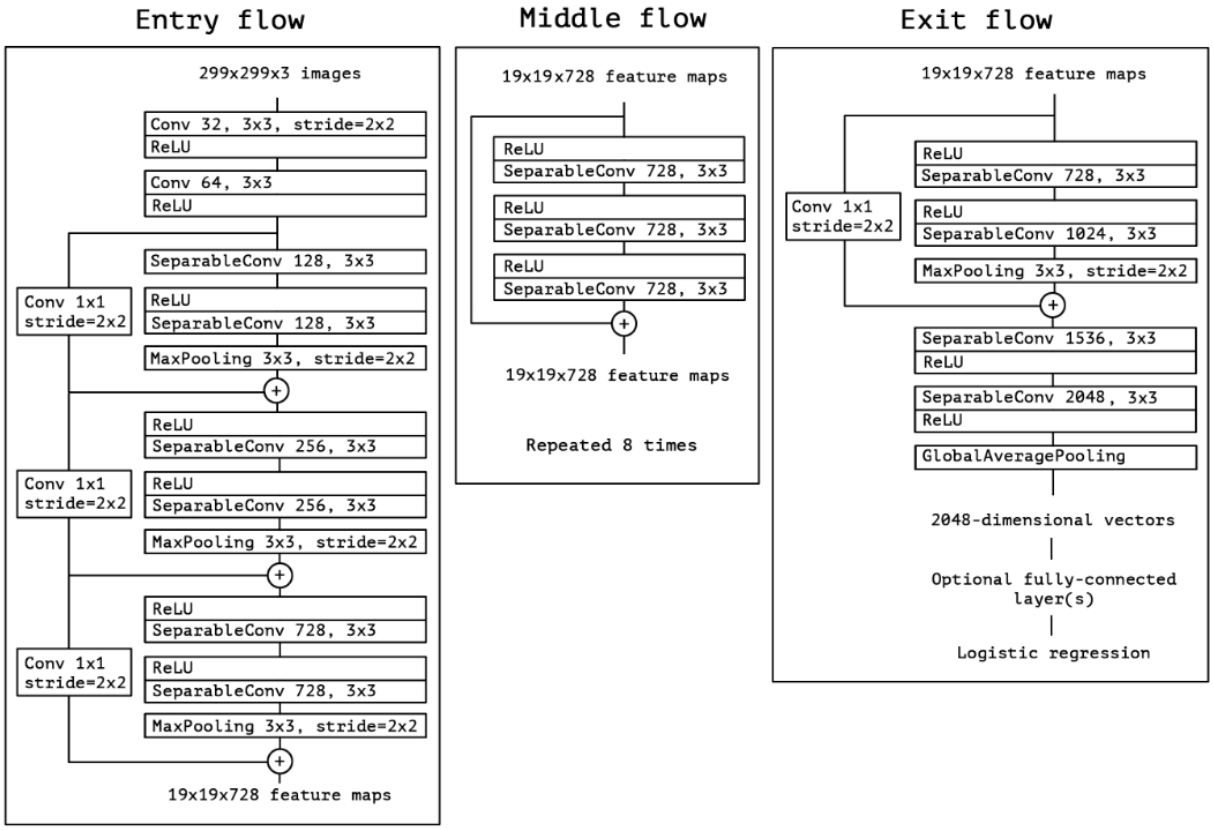

Figure 8: Overview of the Xception architecture, including Entry, Middle, and Exit flows. Each module consists of depthwise separable convolutions with batch normalization and ReLU activation. Residual connections span most modules.

3.2 Implementations

Pre-trained Xception models are available in popular libraries such as TensorFlow and Keras. These implementations are optimized for ease of use and adaptation for transfer learning. More details can be found at `https://keras.io/applications/#xception`.

3.3 Transfer Learning with Xception

Base Model

The Xception is available as part of the Keras libirary, and can be loaded in direcectly with the pretrained model weights for the 'ImageNet' dataset. We load in Xception without it's linear classifier or *top*. The Xception base model in this implementation is frozen (it's parameters are not trained) and acts only as a feature extractor.

The Xception base model has input dimensions `(None, 224, 224, 3)`, and output dimensions of `(None, 10, 10, 2048)`.

Classifier

We receive the feature map of dimension `(None, 10, 10, 2048)` from the base model, we apply global average pooling tp reduce the feature map to `(None, 2048)`. This feature map is then fed through a dense layer with 256 units and ReLU activation followed by another dense layer with 39 units and softmax activation to map the features to the 39 classes in our dataset.

Overall Model

Layer (type)	Output Shape	Param #
`input_layer_3` (`InputLayer`)	(None, 299, 299, 3)	0
`xception` (`Functional`)	(None, 10, 10, 2048)	20,861,480
`global_average_pooling2d_1` (`GlobalAveragePooling2D`)	(None, 2048)	0
`dense_1` (`Dense`)	(None, 256)	524,544
`dense_2` (`Dense`)	(None, 39)	10,023

Training Procedure

The model is trained for 5 epochs using the Adam optimizer and a categorical cross-entropy loss function.

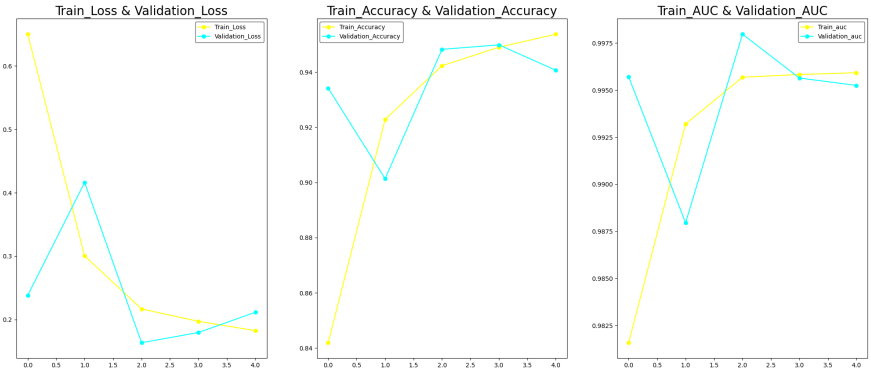

Figure 9: Training history visualization. Accuracy, loss, and AUC-ROC plotted over epochs for both the training and validation subsets.

Evaluation

Table 2: Classification Report

Class	Precision	Recall	F1-Score	Support
Apple___Apple_scab	0.97	0.94	0.95	62
Apple___Black_rot	1.00	0.89	0.94	61
Apple___Cedar_apple_rust	1.00	1.00	1.00	32
Apple___healthy	0.94	0.97	0.96	170
Background_without_leaves	0.99	0.98	0.99	113
Blueberry___healthy	0.90	1.00	0.95	148
Cherry___Powdery_mildew	1.00	0.96	0.98	113
Cherry___healthy	0.99	0.99	0.99	88
Corn___Cercospora_leaf_spot_Gray_leaf_spot	0.95	0.83	0.89	47
Corn___Common_rust	1.00	1.00	1.00	104
Corn___Northern_Leaf_Blight	0.90	0.98	0.94	97
Corn___healthy	1.00	0.99	1.00	133
Grape___Black_rot	0.63	1.00	0.78	140
Grape___Esca_(Black_Measles)	0.96	0.91	0.93	141
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1.00	0.60	0.75	106
Grape___healthy	0.97	0.92	0.95	39
Orange___Haunglongbing_(Citrus_greening)	1.00	1.00	1.00	552
Peach___Bacterial_spot	0.99	0.98	0.98	232
Peach___healthy	0.94	1.00	0.97	31
Pepper,_bell___Bacterial_spot	0.98	0.93	0.95	88
Pepper,_bell___healthy	0.98	0.99	0.99	151
Potato___Early_blight	0.98	0.97	0.98	101
Potato___Late_blight	0.91	1.00	0.95	107
Potato___healthy	1.00	0.57	0.73	14
Raspberry___healthy	1.00	0.97	0.99	36
Soybean___healthy	1.00	1.00	1.00	485
Squash___Powdery_mildew	1.00	1.00	1.00	202
Strawberry___Leaf_scorch	0.97	0.75	0.85	96
Strawberry___healthy	0.73	0.97	0.83	36
Tomato___Bacterial_spot	0.94	0.99	0.96	217
Tomato___Early_blight	0.82	0.92	0.87	85
Tomato___Late_blight	0.96	0.89	0.92	178
Tomato___Leaf_Mold	1.00	0.83	0.91	93
Tomato___Septoria_leaf_spot	0.94	0.85	0.89	181
Tomato___Spider_mites_Two-spotted_spider_mite	0.87	0.98	0.92	178
Tomato___Target_Spot	1.00	0.51	0.68	129
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1.00	0.97	0.99	547
Tomato___Tomato_mosaic_virus	0.87	0.97	0.92	35
Tomato___healthy	0.78	1.00	0.88	177
Accuracy			0.95	5545
Macro Avg	0.95	0.92	0.93	5545
Weighted Avg	0.96	0.95	0.95	5545

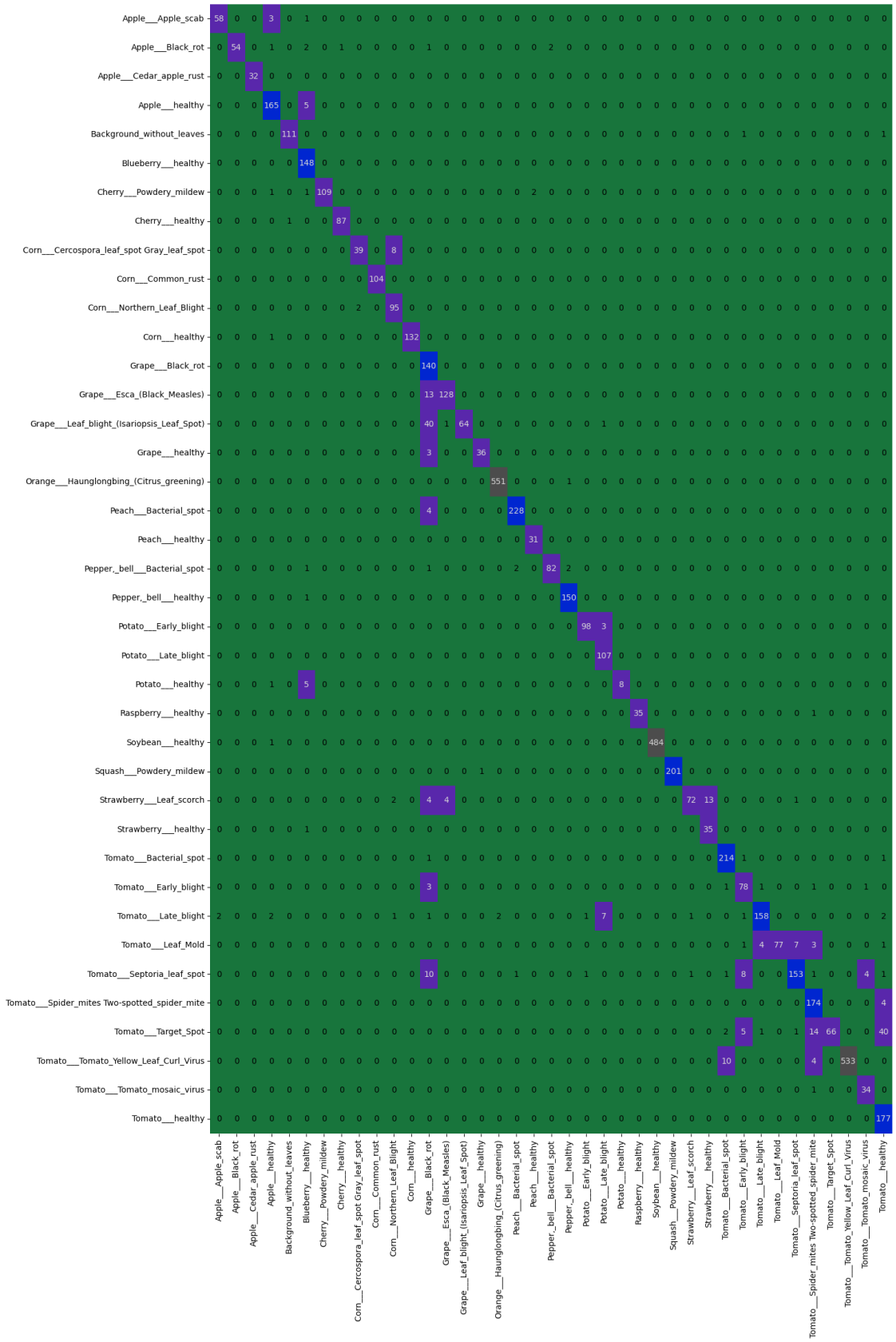

Figure 10: Confusion Matrix.

4 ResNets: Residual Neural Networks

The ResNet architecture was introduced in the 2015 paper: 'Deep Residual Learning for Image Recognition' by Kaiming He et al. [1]. ResNets were designed to address the challenges of training very deep neural networks, particularly the degradation problem, where adding more layers to a sufficiently deep model leads to higher training error.

The key innovation of ResNets lies in their use of *residual learning*, where layers learn the residual mapping $F(x) := H(x) + x$, rather than directly approximating the desired function $H(x)$. This reformulation allows deep networks to achieve better optimization and generalization performance.

4.1 Model Architecture

Residual Blocks

A fundamental building block of a ResNet is the residual block, where a shortcut connection bypasses one or more layers. Formally, the output of a residual block can be expressed as:

$$y = F(x, W_i) + x, \tag{9}$$

where x is the input, y is the output, and $F(x, W_i)$ represents the residual mapping implemented by the intermediate layers (e.g., convolutions, batch normalization, and activation functions). The addition operation ensures that the shortcut connection passes the input x directly to the output.

When the input and output dimensions differ, the shortcut connection can include a linear projection (e.g., a 1×1 convolution) to match dimensions:

$$y = F(x, W_i) + W_s x. \tag{10}$$

Bottleneck Design

For deep ResNet architectures (e.g., ResNet-50, ResNet-101, ResNet-152), the residual block uses a bottleneck design to reduce computational complexity. Each block consists of three layers:

- A 1×1 convolution reduces the input dimensions.
- A 3×3 convolution processes the reduced dimensions.
- A 1×1 convolution restores the original dimensions.

4.2 Implementations

Pre-trained ResNet models are available in libraries such as TensorFlow, PyTorch, and Keras, with variants optimized for tasks like image classification, detection, and segmentation. Code and models are accessible at `https://github.com/KaimingHe/deep-residual-networks`.

4.3 Training ResNets From Scratch

Model Definition

We define each residual block as a combination of two convolutional layers with batch normalization and ReLU activation, followed by a skip connection that adds the input of the block (identity) to the output of the second convolution*.

The overall model is constructed with an initial convolutional layer, followed by a max pooling layer and four stages of residual blocks. Each stage contains a different number of blocks, with the number of filters doubling in each subsequent stage.

The final layers consist of a global average pooling layer and a dense layer with a softmax activation function for classification.

* If the dimensions of the input and output do not match (e.g., due to stride changes), a 1×1 convolution is applied to the input to match the dimensions.

Layer (type)	**Output Shape**	**Param #**
Input Layer (`InputLayer`)	(None, 128, 128, 3)	0
Conv2D (`Conv2D`)	(None, 128, 128, 64)	1,792
Batch Normalization (`BatchNormalization`)	(None, 128, 128, 64)	256
ReLU (`ReLU`)	(None, 128, 128, 64)	0
MaxPooling2D (`MaxPooling2D`)	(None, 64, 64, 64)	0
Residual Block Stage 1 (`residual_block`)	(None, 64, 64, 64)	Multiple
Residual Block Stage 2 (`residual_block`)	(None, 32, 32, 128)	Multiple
Residual Block Stage 3 (`residual_block`)	(None, 16, 16, 256)	Multiple
Residual Block Stage 4 (`residual_block`)	(None, 8, 8, 512)	Multiple
Global Average Pooling (`GlobalAveragePooling2D`)	(None, 512)	0
Dense (`Dense`)	(None, num_classes)	num_classes $\times$ 513

Training

We train our ResNet model for 20 epochs using the Adam optimizer and a categorical cross-entropy loss function. We decay the learning rate after each epoch, from an initial 0.001, at a rate of 0.96.

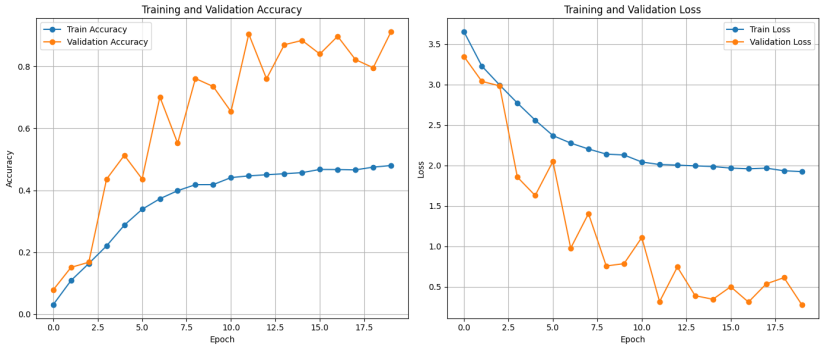

Figure 11: Training history visualization. Accuracy and loss plotted over epochs for both the training and validation subsets.

Evaluation

Table 3: Classification Report

Class	Precision	Recall	F1-Score	Support
Apple___Apple_scab	0.96	0.68	0.80	94
Apple___Black_rot	1.00	0.92	0.96	93
Apple___Cedar_apple_rust	0.95	0.98	0.96	41
Apple___healthy	0.97	0.81	0.88	247
Background_without_leaves	0.95	0.89	0.92	171
Blueberry___healthy	0.99	0.68	0.81	226
Cherry___Powdery_mildew	0.90	0.99	0.94	158
Cherry___healthy	0.94	0.96	0.95	128
Corn___Cercospora_leaf_spot Gray_leaf_spot	0.73	0.81	0.77	77
Corn___Common_rust	0.91	0.88	0.90	179
Corn___Northern_Leaf_Blight	0.84	0.93	0.88	147
Corn___healthy	0.99	0.95	0.97	174
Grape___Black_rot	0.83	0.99	0.90	177
Grape___Esca_(Black_Measles)	1.00	0.83	0.91	207
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.94	1.00	0.97	161
Grape___healthy	0.97	1.00	0.98	63
Orange___Haunglongbing_(Citrus_greening)	0.98	0.98	0.98	826
Peach___Bacterial_spot	0.91	0.92	0.91	345
Peach___healthy	0.92	0.89	0.91	54
Pepper,_bell___Bacterial_spot	0.83	0.99	0.90	149
Pepper,_bell___healthy	0.97	0.82	0.89	222
Potato___Early_blight	0.92	0.95	0.94	150
Potato___Late_blight	0.95	0.91	0.93	150
Potato___healthy	0.34	0.96	0.51	23
Raspberry___healthy	0.50	1.00	0.67	55
Soybean___healthy	0.97	0.96	0.96	764
Squash___Powdery_mildew	0.89	1.00	0.94	276
Strawberry___Leaf_scorch	0.99	0.94	0.96	166
Strawberry___healthy	0.60	0.97	0.74	68
Tomato___Bacterial_spot	0.80	0.93	0.86	319
Tomato___Early_blight	0.93	0.43	0.58	150
Tomato___Late_blight	0.90	0.77	0.83	287
Tomato___Leaf_Mold	0.84	0.93	0.88	143
Tomato___Septoria_leaf_spot	0.91	0.91	0.91	266
Tomato___Spider_mites Two-spotted_spider_mite	0.79	0.89	0.84	252
Tomato___Target_Spot	0.76	0.88	0.81	211
Tomato___Tomato_Yellow_Leaf_Curl_Virus	0.98	0.95	0.96	804
Tomato___Tomato_mosaic_virus	0.98	0.95	0.96	56
Tomato___healthy	0.94	0.91	0.92	239
Accuracy			0.91	8318
Macro Avg	0.88	0.90	0.88	8318
Weighted Avg	0.92	0.91	0.91	8318

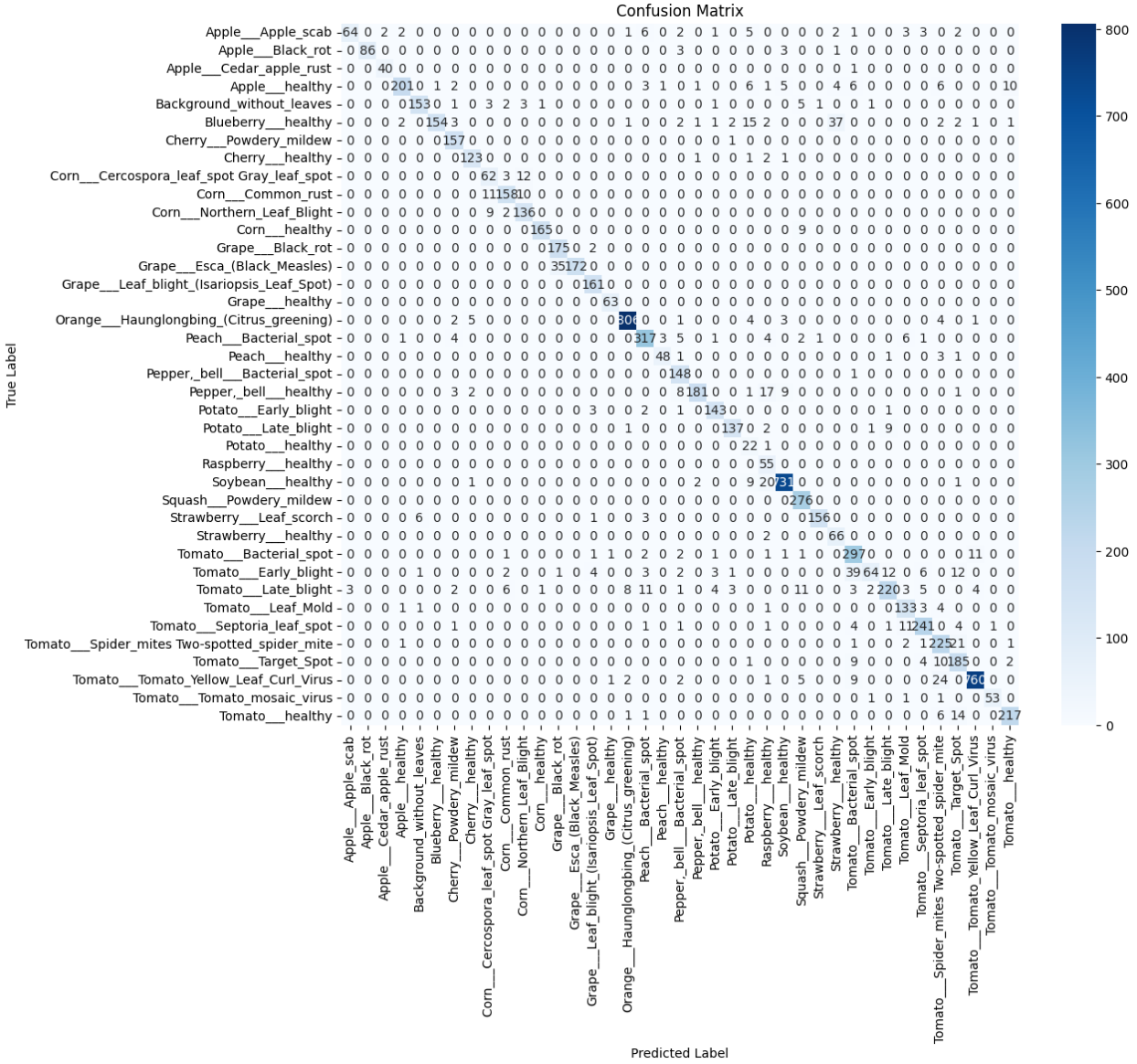

Figure 12: Confusion Matrix.

Table 4: ROC AUC Scores for Each Class

Class	AUC Score
Apple___Apple_scab	0.8402
Apple___Black_rot	0.9624
Apple___Cedar_apple_rust	0.9877
Apple___healthy	0.9064
Background_without_leaves	0.9469
Blueberry___healthy	0.8406
Cherry___Powdery_mildew	0.9957
Cherry___healthy	0.9800
Corn___Cercospora_leaf_spot Gray_leaf_spot	0.9012
Corn___Common_rust	0.9404
Corn___Northern_Leaf_Blight	0.9611
Corn___healthy	0.9740
Grape___Black_rot	0.9921
Grape___Esca_(Black_Measles)	0.9155
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.9993
Grape___healthy	0.9999
Orange___Haunglongbing_(Citrus_greening)	0.9870
Peach___Bacterial_spot	0.9574
Peach___healthy	0.9442
Pepper,_bell___Bacterial_spot	0.9947
Pepper,_bell___healthy	0.9073
Potato___Early_blight	0.9759
Potato___Late_blight	0.9562
Potato___healthy	0.9757
Raspberry___healthy	0.9967
Soybean___healthy	0.9769
Squash___Powdery_mildew	0.9979
Strawberry___Leaf_scorch	0.9698
Strawberry___healthy	0.9826
Tomato___Bacterial_spot	0.9609
Tomato___Early_blight	0.7130
Tomato___Late_blight	0.8818
Tomato___Leaf_Mold	0.9634
Tomato___Septoria_leaf_spot	0.9516
Tomato___Spider_mites Two-spotted_spider_mite	0.9427
Tomato___Target_Spot	0.9348
Tomato___Tomato_Yellow_Leaf_Curl_Virus	0.9715
Tomato___Tomato_mosaic_virus	0.9732
Tomato___healthy	0.9531
Micro-average AUC	0.9527
Macro-average AUC	0.9490

5 Comparison

Metric	DenseNet	DenseNet	Xception	ResNet
Implementation	Transfer Learning	Finetuning	Transfer Learning	From Scratch
# Trainable Params	74,919	18,167,847	534,567	11,193,639
Training Epochs	11	20	5	20
Train Accuracy	96.6%	50.9%	95.3%	47.9%
Test Accuracy	95.0%	98.0%	94.6%	91.0%
Test AUC	-	0.99	0.996	0.949
Precision	-	0.98	0.95	0.88
Recall	-	0.98	0.92	0.90
F1-Score	-	0.98	0.93	0.88
Support	-	8318	5545	8318

Table 5: Comparison Table of Model Implementations

5.1 Observations

- **DenseNet with Finetuning** achieved the highest test accuracy (98.0%) and AUC (0.99), showcasing the benefits of extensive finetuning with pretrained weights. In contrast, **DenseNet with Transfer Learning** performed well (95.0% accuracy) but with fewer trainable parameters.
- **Xception with Transfer Learning** demonstrated strong performance with a high test AUC (0.996) and competitive accuracy (94.6%), despite having significantly fewer parameters and requiring only 5 training epochs.
- **ResNet trained from scratch** lagged behind, achieving the lowest test accuracy (91.0%) and AUC (0.949), likely due to its dependence on extensive data and training epochs without pretrained weights.