# Cover Page:

Contents:

[Cover Page: 1](#_Toc164033708)

[Contents: 2](#_Toc164033709)

[Abstract: 3](#_Toc164033710)

[Acknowledgement: 4](#_Toc164033711)

[List of Figures: 5](#_Toc164033712)

[List of Tables: 6](#_Toc164033713)

[Introduction: 7](#_Toc164033714)

[Background: 7](#_Toc164033715)

[Objectives: 8](#_Toc164033716)

[Limitations: 9](#_Toc164033717)

[Literature Review: 11](#_Toc164033718)

[Methodology: 13](#_Toc164033719)

[Introduction: 13](#_Toc164033720)

[Dataset: 13](#_Toc164033721)

[Software and Hardware Specifications: 14](#_Toc164033722)

[Deep Learning Architectures: 15](#_Toc164033723)

[Deep Learning Optimizers: 24](#_Toc164033724)

[Training Specifications: 24](#_Toc164033725)

[Code implementation and explanation: 26](#_Toc164033726)

[Results: 32](#_Toc164033727)

[Introduction to Results: 32](#_Toc164033728)

[Training Progress: 33](#_Toc164033729)

[Training, Validation, and Testing Performance: 35](#_Toc164033730)

[Classification Report Analysis: 41](#_Toc164033731)

[Conclusion: 43](#_Toc164033732)

[Future Directions: 43](#_Toc164033733)

[Summary of Results: 44](#_Toc164033734)

[Significance of the Study: 46](#_Toc164033735)

[References 48](#_Toc164033736)

# Abstract:

Recently, various state-of-the-art deep learning (DL) algorithms have been used to classify plant diseases on publicly available/author-generated datasets. In this research, a deep learning-based comparative analysis was proposed for plant disease classification. First, a comparative analysis of well-known DCNN architectures was proposed. The research found a convolutional neural network (CNN) a best Second, which attempted to improve the performance of the realized model by training on various deep learning optimizers have been developed. The comparison between different DCNNs was based on performance metrics such as validation accuracy/loss, F1 score, and number of required epochs All the selected DCNN architectures were trained on the PlantVillage dataset with 26 different diseases in 14 plant species. Keras with a TensorFlow backend was used to train the deep learning algorithm. It is concluded that the Xception architecture trained with Adam optimizer obtained the highest verification accuracy and F1-score of 99.87% and 0.9978, respectively, better than the previous methods and indicating the novelty of the work Therefore, the method proposed in this study can be applied in other agricultural applications to identify its explicit classification.

# Acknowledgement:

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Bappaditya Mandal, for his continuous support and guidance throughout my BSc project. This project would not have been possible without his direction and constant assistance.

Secondly, I would like to thank Ash leake for his technical support and for providing timely resource arrangement.

Finally, I would like to thank my family for continually believing in me. They have always encouraged me never to give up in pursuing my goals. Without their love and presence, the successes as well as the failures would have been meaningless. (Huang, 2020)

# List of Figures:

[Figure 1.EfficientNetB3-Architecture 18](#_Toc163843315)

[Figure 2.MobileNetV1-Architecture 19](#_Toc163843316)

[Figure 3.ResNet50-Architecture 20](#_Toc163843317)

[Figure 4.VGG16-Architecture 21](#_Toc163843318)

[Figure 5.Xception-Architecture 22](#_Toc163843319)

[Figure 6. The methodology of this research. 24](#_Toc163843320)

[Figure 7. Some of the plant diseases from the PlantVillage dataset. 24](#_Toc163843321)

[Figure 8. Data Splitting - Code Snippet 27](#_Toc163843322)

[Figure 9. Data Augmentation - Code Snippet 29](#_Toc163843323)

[Figure 10. Custom Callback - Code Snippet 32](#_Toc163843324)

# List of Tables:

[Table 1.Architecture Metrics 22](#_Toc163922470)

[Table 2. Hyperparameters of the deep learning optimizers 25](#_Toc163922471)

[Table 3.Training Time Comparison 34](#_Toc163922472)

[Table 4. Training Metrics 36](#_Toc163922473)

[Table 5.Validation Metrics 38](#_Toc163922474)

[Table 6. Testing Metrics 40](#_Toc163922475)

[Table 7.Classification Report 42](#_Toc163922476)

# Introduction:

## Background:

Agriculture is considered to play an important role in sustaining human life and ensuring food security around the world. With the global population projected to reach 9 billion by 2050, there is increasing pressure to increase agricultural productivity and reduce challenges such as plant diseases. Plant diseases have economic and environmental impacts, leading to crop losses, reduced yields, and increased use of pesticides.

Traditional approaches to diagnosis and management are often relied upon, which can be time-consuming, subjective, and limited in feasibility. Modern technology combinations, especially computer vision and machine learning, are seen as promising solutions to address these challenges. By using big data and advanced algorithms, automated systems for rapid, accurate diagnosis and targeted therapy in plants can be developed.

The availability of datasets such as the Plant Village dataset has changed the landscape of plant disease research. The Plant Village dataset contains a high-quality collection of images showing a variety of plant diseases including fungi, viruses, and pathogens as well as samples of healthy plants. These labelled images serve as valuable training data for training and evaluation of machine learning models, especially convolutional neural networks (CNNs) (Hughes & Salathe, 2015).

The use of pre-existing CNN architectures such as ResNet (He, et al., 2015), VGG (Simonyan & Zisserman, 2014), and Xception (Chollet, 2016) is seen to provide a structured approach to modelling, allowing researchers to focus on optimizing and generalizing model performance. It also gives flexibility and improves convergence speed during training.

## Objectives:

This study aims to evaluate and compare the performance of various pre-existing convolutional neural network (CNN) architectures for accurate plant disease classification using the Plant Village dataset. The objective of the study is to aid insights in the selection of optimal models to optimize agricultural applications. To achieve this objective, the following objectives will be pursued.

First, several CNN algorithms, including ResNet, VGG, and Inception etc, will be trained on the Plant Village dataset. Standardized preprocessing techniques and hyperparameters will be used to remove variability during the training process. The trained models will then be evaluated and compared based on their classification accuracy, robustness, and computational efficiency.

Second, various optimization algorithms such as Adamax, Adadelta, and RMSProp will be used to evaluate their impact on model convergence performance and trade-off analysis between convergence speed and accuracy. The results obtained by each optimizer will be analysed to understand their effectiveness in training CNN models for classifying plant diseases.

Third, model performance will be evaluated using standard metrics including accuracy, precision, recall, and F1-score. The generation and analysis of confusion matrices will provide insight into the ability of different models to correctly identify different plant diseases and will further contribute to more detailed studies of classification efficiencies.

Finally, the study aims to identify and compare the best-performing CNN designs and optimization algorithms based on established evaluation metrics and criteria. This requires intensive research to understand the strengths and limitations of each model in a real-world agricultural setting.

## Limitations:

This research project has several limitations that define the context and the boundaries of the study. First, the reliance on plant village dataset for model training and analysis poses limitations on dataset representativeness. Although the list covers a wide range of plant diseases, it may not include all possible variations and environmental factors encountered in actual agricultural conditions. These limitations require caution when attributing outcomes are moving into broader agricultural contexts or diversity contexts.

Second, computer hardware plays an important role in the size and complexity of the experiments conducted in this study. Limited technical capacity may prevent analysis of large datasets, complex model architectures, or detailed hyperparameter tuning, which may affect the depth of analysis and optimization achieved.

Furthermore, the choice of the overall findings are specific choices related to CNN architectures, optimization algorithms, dataset preprocessing techniques, although these choices are made deliberately to maintain the accuracy of the experiment, they can hinder limit the applicability of the results to other models.

Finally, external factors such as changes in disease prevalence, changing environmental factors, or extracurricular technological advances may affect the relevance and consumption of research findings in the long run.

# Literature Review:

One of the key features of DCNNs is their ability to learn hierarchical representations of data at different levels of abstraction. This hierarchical learning allows DCNNs to capture complex patterns and relationships within the data, making them extremely effective for tasks such as image classification. The foundation of DCNNs was laid by (LeCun, et al., 1998) during the development of LeNet-5. The success of DCNNs in image classification can be attributed to their ability to automatically learn and adapt features during the training process, making them versatile and well-suited for a wide range of tasks. In addition to their effectiveness in image classification, DCNNs have also been successfully applied to a variety of other computer vision tasks, such as object detection, segmentation, and image recognition. One important development was the unveiling of AlexNet by (Krizhevsky, et al., 2012), who won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a wide margin, showcasing the power of deep CNNs. Many architectures followed AlexNet, each improving on the advantages of the previous models while resolving their drawbacks. These architectures have proved to be robust and capable of handling large datasets with high levels of accuracy, making them a valuable tool for researchers and practitioners in the field of computer vision. Furthermore, the evolution of DCNN architectures has led to the development of more sophisticated models, such as ResNet (He, et al., 2015), Xception (Chollet, 2016), and VGG-16 (Simonyan & Zisserman, 2014), which have further improved the performance of image classification systems. These more advanced architectures incorporate techniques such as skip connections, batch normalization, and bottleneck layers to enhance the learning capabilities of the network and reduce issues such as vanishing gradients. Overall, the advancements in CNNs have significantly impacted the field of computer vision, providing researchers and practitioners with powerful tools for a wide range of image analysis tasks. The continued development and refinement of these architectures will likely lead to even greater breakthroughs in the future, further cementing CNNs as a foundational technology in the field of computer vision.

Xception, introduced by (Chollet, 2016), set a new standard in CNN architecture by incorporating depth wise separable convolutions, which significantly reduced the number of parameters compared to traditional convolutional layers. This led to improved efficiency and scalability, making it possible to deploy deeper networks without suffering from issues such as vanishing gradients. ResNet, developed by (He, et al., 2015), introduced the concept of residual connections, where input from one layer is added to the outputs of subsequent layers. This design helped address the problem of vanishing gradients in very deep networks, allowing for the training of CNNs with hundreds of layers. This breakthrough enabled further improvements in accuracy and performance on tasks such as image recognition and classification. VGG, proposed by (Simonyan & Zisserman, 2014), also made significant contributions to CNN architecture by introducing the concept of using small 3x3 convolutional filters successively, leading to deeper networks with smaller-sized kernels. This approach helped in capturing more intricate features in images while maintaining a simple and uniform architecture, thereby improving the overall accuracy of the model.

These advancements in CNN architecture have paved the way for the development of even more sophisticated neural networks that continue to push the boundaries of what is possible in tasks such as image recognition, object detection, and natural language processing. As researchers and engineers strive for higher levels of accuracy, efficiency, and scalability, it is likely that we will see further innovations in CNN architecture that will continue to drive progress in the field of deep learning. Furthermore, research in the field of CNN architectures has also expanded beyond just image analysis. Applications such as video processing, natural language processing, and speech recognition have also seen significant advancements using CNNs. For example, the use of CNNs in video action recognition has shown promising results in accurately identifying and classifying complex human expression in photos and videos (Mollahosseini, et al., 2016). In the realm of Computer Vision, CNNs have been applied to tasks such as sentiment analysis and gender classification. The ability of CNNs to capture spatial and sequential relationships in data makes them well-suited for tasks involving text and sequences. In gender classification, for example, CNNs have been shown to outperform traditional methods in accurately classifying the age and gender of a person (Levi & Hassner, 2015).

Overall, the versatility and effectiveness of CNN architectures in various domains highlight their potential to revolutionize the way we approach complex data analysis tasks. As research in this field continues to evolve, we can expect to see even more innovative applications of CNNs that push the boundaries of what is possible in machine learning and artificial intelligence. Through advancements in CNN architectures, we are one step closer to unlocking the full potential of deep learning in solving real-world problems.

Moreover, deep learning techniques have not only been used for disease detection (Ferentinos, 2018) in crops but also for weed identification and detection (Mohanty, et al., 2016). By training CNNs on large databases of weed images, researchers have been able to accurately classify and locate various types of weeds in agricultural fields. This allows farmers to selectively target and remove weeds, minimizing the use of herbicides and increasing crop yields. In addition to disease and weed detection, image classification is also useful for monitoring crop growth and assessing overall plant health. By analysing aerial imagery and satellite data with CNNs, farmers can quickly track the progress of their crops and identify areas that may require additional attention or resources. This data can help optimize irrigation, fertilizer application, and pest control efforts, ultimately leading to more efficient and sustainable farming practices. Overall, the integration of deep learning and image classification into agriculture has the potential to revolutionize the way we approach farming. By leveraging the power of artificial intelligence, farmers can make more informed decisions, reduce waste, and increase productivity. As technology continues to advance, we can expect to see even more sophisticated applications of CNNs in agriculture, leading to a more sustainable and environmentally friendly food production system.

One key challenge in plant disease classification is the vast diversity of plant species and the wide array of potential diseases that can affect each plant. This diversity requires specialized CNN models with the ability to discriminate between subtle differences in plant symptoms and accurately identify diseases across multiple plant species. Additionally, the development of efficient data augmentation techniques and transfer learning strategies is crucial for improving the generalization performance of CNN models, especially in cases where annotated datasets may be limited or contain imbalanced classes. Moreover, the integration of advanced sensor technologies, such as hyperspectral imaging and multi-spectral sensing, can enhance the accuracy and efficiency of automated disease detection systems. By capturing detailed spectral information from plants, these technologies can provide insights into the physiological changes caused by diseases, allowing for earlier and more accurate diagnoses. Furthermore, the implementation of real-time monitoring systems utilizing IoT devices and cloud-based analytics can enable continuous surveillance of crops, alerting farmers to potential disease outbreaks and facilitating timely interventions. The ongoing research in the field of plant disease classification holds great promise for revolutionizing agricultural sustainability and improving food security worldwide. By leveraging the power of CNN architectures, advanced sensor technologies, and computational tools, scientists and researchers are paving the way for more precise and efficient methods of detecting and combating plant diseases. Through continued collaboration and innovation, we can work towards a future where automated disease identification systems play a vital role in ensuring the health and productivity of global crop yields.

In conclusion, the development of CNN architectures has elevated the field of image classification to previously unheard-of levels, with applications ranging from the classification of plant diseases to agriculture. This work aims to contribute to the current efforts in applying deep learning for real-world difficulties in agriculture and food production by expanding upon previous research and utilizing cutting-edge approaches.

# Methodology:

## Introduction:

Convolutional Neural Networks (CNNs) have been widely utilized for image classification tasks, prompting this research to investigate the performance of several novel CNN architectures for classifying plant diseases. All 5 Deep CNN architectures considered for this study were developed by researchers over the years. The Methodology is depicted in Figure 6.

The overall methodology employed in this study was to train the 5 DCNN architectures on the Plant Village dataset using 3 different optimizers for each. The convergence of the 5 CNN architectures to the final training/validation values was observed to update the hyperparameters. Subsequently, the CNN models were compared in terms of training, validation, and testing accuracy/loss, as well as F1-score. This comparison led to the utilization of DL optimization algorithms to further enhance the performance of their CNN architectures, resulting in the highest F1 score in their specific category. This methodology was inspired by the works of (Saleem, et al., 2020).

## Dataset:

All DCNN models were trained on the publicly available Plant Village dataset curated by (Hughes & Salathe, 2015), which contains 54,306 images with 38 different healthy/diseased leaves associated with their 14 plant species (some of the plant diseases are shown in Figure 7), image size Converted to 224 × 3 and normalized by dividing the pixels values ​​by 255 to match the initial values ​​of the models To avoid overfitting, 70%, 20% and 10% data sets were allocated of the training, validation and test data sets, respectively.

## Software and Hardware Specifications:

The software environment for this research work is developed using Python 3.11.4 in compatible operating systems such as Linux, Windows, or macOS. The primary integrated development environment (IDE) used for coding and testing is Google Colab and Kaggle. These environments provide a robust platform for the development and successful implementation of machine learning models.

This research relies on several core libraries and resources for data binding, deep learning model development, and visualization. System libraries such as os, time, shutil, and pathlib facilitate file management, time-sensitive operations, and access management in the operating system. For data processing and transformation, the OpenCV library (cv2) is used for graphical representation, while numpy and pandas are used for numerical calculations and data analysis tasks, respectively.

Deep learning functions are implemented using TensorFlow version 2.9.1 and Keras libraries. TensorFlow provides the backbone for building and training deep learning models, while Keras provides a high-level API for building simple neural networks The Keras.preprocessing.image module supports image data processing and enhancement through the ImageDataGenerator class. Various layers (e.g., Conv2D, MaxPooling2D, Dense) and optimizers (e.g., Adamax, RMSprop, Adadelta) from Keras are used for model architecture and optimization.

The software stack also includes tools for model evaluation and visualization. The sklearn library is used for classification reports, and model evaluation metrics. In addition, data visualization is facilitated by the use of seaborn and matplotlib.pyplot to create informative plot graphs.

All experiments were performed on a graphical processing unit (2 x NVIDIA T4) with specifications: 16GB GDDR6 memory, 2560 CUDA cores and 320 Tensor cores, 1590 MHz core clock, and 320 GB/sec memory bandwidth.

## Deep Learning Architectures:

Following the creation of the AlexNet (Krizhevsky, et al., 2012) architecture, a groundbreaking era of cutting-edge CNN architectures was initiated for several image categorization applications. Thus, we examined well-liked and effective CNN models including VGG-16 (Simonyan & Zisserman, 2014), ResNet-50 (He, et al., 2015), MobileNet (Adam, et al., 2017), EfficientNetB3 (Le & Tan, 2019), and Xception (Chollet, 2016) in this article.

Using information taken from the Keras website, the study provides a thorough analysis of a number of well-known convolutional neural network (CNN) models, including EfficientNetB3, MobileNet, ResNet50, VGG16, and Xception. The features of every model are examined to offer a thorough comprehension of their performance in relation to different metrics.

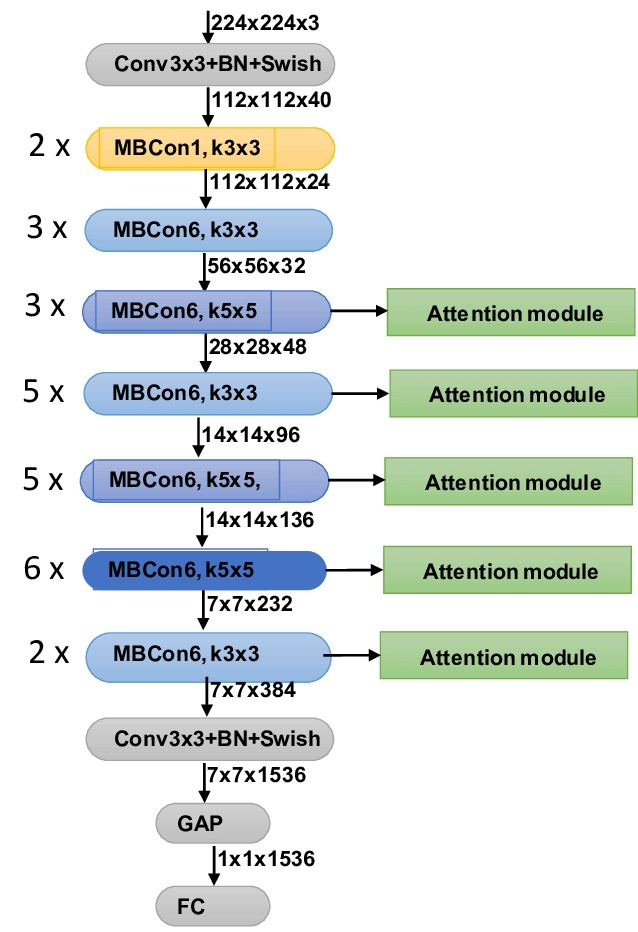
The EfficientNetB3 architecture developed by (Le & Tan, 2019) is renowned for its effectiveness in striking a balance between model size and performance. It is composed of several convolutional layer blocks, each of which is followed by activation functions and batch normalization. A stem convolutional layer is the first layer in the network, and it uses the input images to extract fundamental information. Then, a sequence of blocks comprised of inverted bottleneck layers, squeeze-and-excitation modules, and depth wise separable convolutions are stacked to capture progressively more complicated characteristics. Lastly, fully connected layers and global average pooling are used to carry out classification tasks. Because of its focus on scalability and parameter efficiency, EfficientNetB3 is an excellent choice for environments with limited resources. EfficientNetB3 stands out because of its moderate size of 48 MB. It achieves impressive performance with a top-1 accuracy of 81.6% and a top-5 accuracy of 95.7%, while having a comparatively compact architecture. The number of parameters in the model—12.3 million—shows how well it uses computational power. Furthermore, its significant depth of 210 layers demonstrates its ability to extract complex features from the data. EfficientNetB3 exhibits efficiency in inference time, requiring 140.0 milliseconds on CPU and 8.8 milliseconds on GPU for each inference step.

Figure 1.EfficientNetB3-Architecture

A lightweight CNN architecture called MobileNetV1 was created by (Adam, et al., 2017) for embedded and mobile devices. It has depth wise separable convolutions, which reduce computational complexity and model size by splitting typical convolutions into distinct depth wise and pointwise convolutions. A sequence of convolutional layers precedes depth-wise separable convolutional blocks in the network's architecture. To effectively capture spatial and channel-wise information, pointwise convolutional layers are inserted across these blocks. The last two layers are a fully connected layer for classification and global average pooling. MobileNetV1 is suited for real-time applications on systems with limited resources because of its architecture, which places a high priority on computational efficiency while retaining competitive accuracy. Even with its significantly reduced size of 16 MB, MobileNet shows good performance metrics. Despite its decreased complexity, its top-1 accuracy of 70.4% and top-5 accuracy of 89.5% demonstrate its competence. Combining computational efficiency and performance, MobileNet has a low parameter count of 4.3 million and a deep layer count of 55. Its acceleration to 22.6 milliseconds per step on the CPU and GPU further confirms that it is suitable for contexts with limited resources.

A diagram of a computer

Description automatically generated

Figure 2.MobileNetV1-Architecture

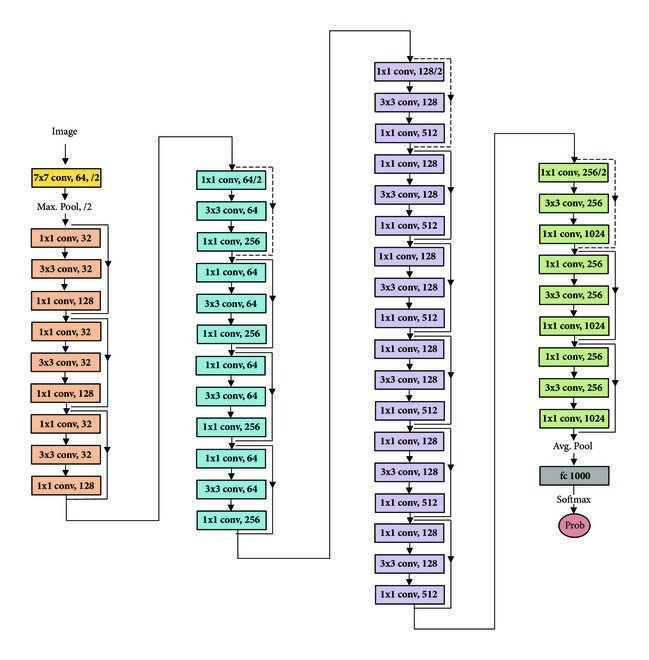
Renowned for its residual learning framework, ResNet50 is a deep CNN architecture developed by (He, et al., 2015) that helps to train very deep networks by mitigating the vanishing gradient issue. The network consists of numerous residual blocks, each of which has identity shortcuts after a number of convolutional layers. By facilitating the gradient flow during training, these shortcuts enable deep network optimization that is both efficient and effective. A stem convolutional layer and successive blocks of residual units are the architecture's foundation. The last two layers are a fully connected layer for classification and global average pooling. Because of ResNet50's architecture, deep networks with hundreds of layers may be trained with effective optimization and top-notch performance. ResNet50 has a bigger, 98 MB architecture and is more reliable. Its 74.9% top-1 accuracy and 92.1% top-5 accuracy are praiseworthy, but they come with a hefty 25.6 million parameter count. Nonetheless, the model's 107 layers of depth enable it to identify intricate patterns in the data. ResNet50 shows competitive inference times despite its high processing requirements, needing 4.6 milliseconds on GPU and 58.2 milliseconds on CPU each step.

Figure 3.ResNet50-Architecture

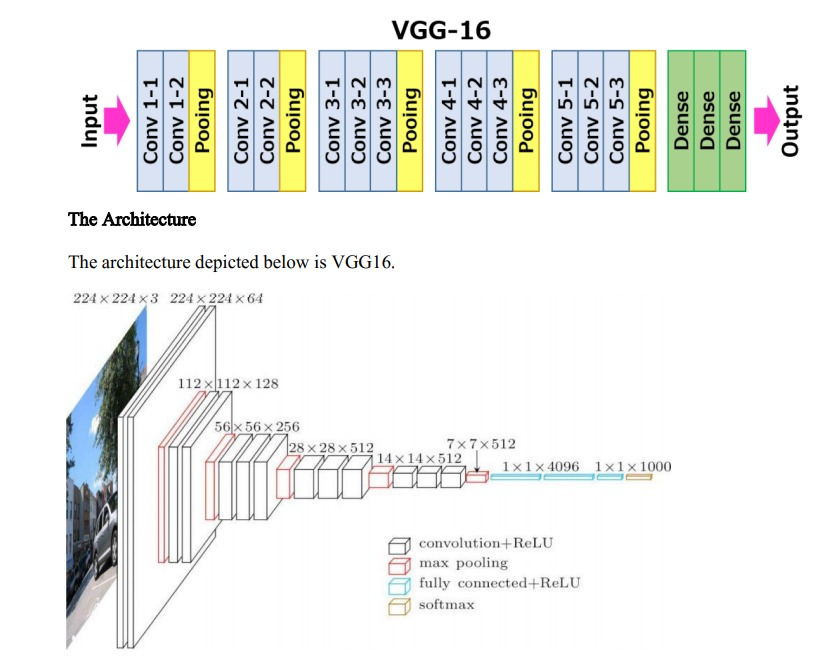
The traditional VGG16 architecture was curated by (Simonyan & Zisserman, 2014) and is renowned for its simplicity. It is composed of several convolutional layers, then layers for spatial down sampling called max pooling. Repeated blocks of convolutional layers, which get deeper as the network grows, define the architecture of the network. VGG16 is made up of five convolutional blocks, each of which has a max-pooling layer after two or three convolutional layers. The completely connected categorization layers are among the last layers. Because of its simple yet effective architecture, VGG16 is a well-liked option for a variety of computer vision applications. On the other hand, VGG16 is notable for having a large number of parameters—138.4 million—and a large model size (528 MB). Although its top-5 accuracy of 90.1% and top-1 accuracy of 71.3% are impressive, they come at the cost of processing power. Compared to its competitors, VGG16 has a comparatively shallow design with a depth of 16 layers. Its inference time, which requires 4.2 milliseconds on the GPU and 69.5 milliseconds on the CPU per step, is still competitive.

Figure 4.VGG16-Architecture

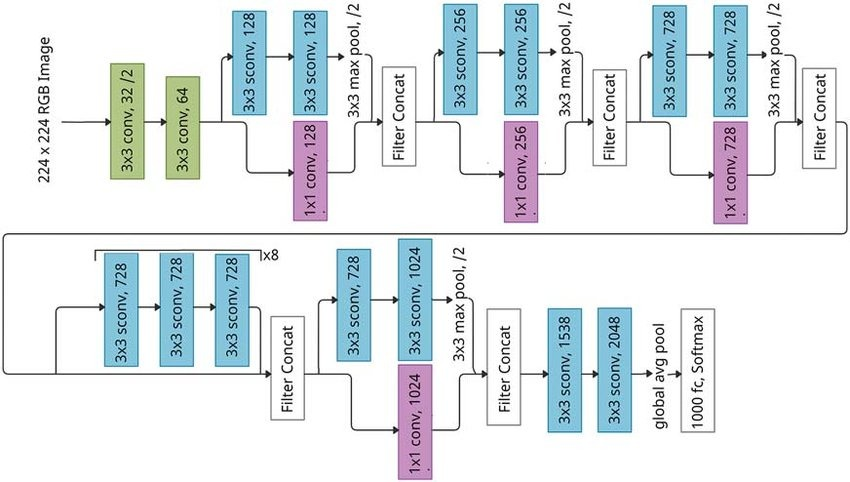
Xception is an expansion of the Inception architecture. (Chollet, 2016) separated the spatial and channel-wise convolutions using depth wise separable convolutions. The goal of this architecture is to reduce both the size of the model and computational complexity while capturing complicated features. Each depth wise separable convolutional block in Xception is followed by activation functions and batch normalization. The network is composed of numerous depth wise separable convolutional blocks stacked on top of a stem convolutional layer. Global average pooling and fully connected categorization layers are the last layers. Comparing Xception's architecture to conventional convolutional architectures, the latter require more parameters to reach state-of-the-art performance. Xception provides a balance of efficiency and performance, even at 88 MB in size. It is positioned well among its contemporaries thanks to its top-1 accuracy of 79.0% and top-5 accuracy of 94.5%. With 81 layers and 22.9 million parameters, Xception is able to reconcile computational efficiency with model complexity. It’s fit for real-time applications is demonstrated by its inference time of 8.1 milliseconds on the GPU and 109.4 milliseconds per step on the CPU.

Figure 5.Xception-Architecture

In summary, the study carefully analyses each CNN model's features as depicted in Table 1, highlighting its advantages and disadvantages in terms of several performance measures and the reasons for which they were chosen for this research.

Table 1.Architecture Metrics

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Size (MB)** | **Top-1**  **Accuracy** | **Top-5**  **Accuracy** | **Parameters** | **Depth** | **Time (ms) per inference step (CPU)** | **Time (ms) per**  **inference step (GPU)** |
| [EfficientNetB3](https://keras.io/api/applications/efficientnet/#efficientnetb3-function) | 48 | 81.6% | 95.7% | 12.3M | 210 | 140.0 | 8.8 |
| [MobileNet](https://keras.io/api/applications/mobilenet) | 16 | 70.4% | 89.5% | 4.3M | 55 | 22.6 | 22.6 |
| [ResNet50](https://keras.io/api/applications/resnet/#resnet50-function) | 98 | 74.9% | 92.1% | 25.6M | 107 | 58.2 | 4.6 |
| [VGG16](https://keras.io/api/applications/vgg/#vgg16-function) | 528 | 71.3% | 90.1% | 138.4M | 16 | 69.5 | 4.2 |
| [Xception](https://keras.io/api/applications/xception) | 88 | 79.0% | 94.5% | 22.9M | 81 | 109.4 | 8.1 |

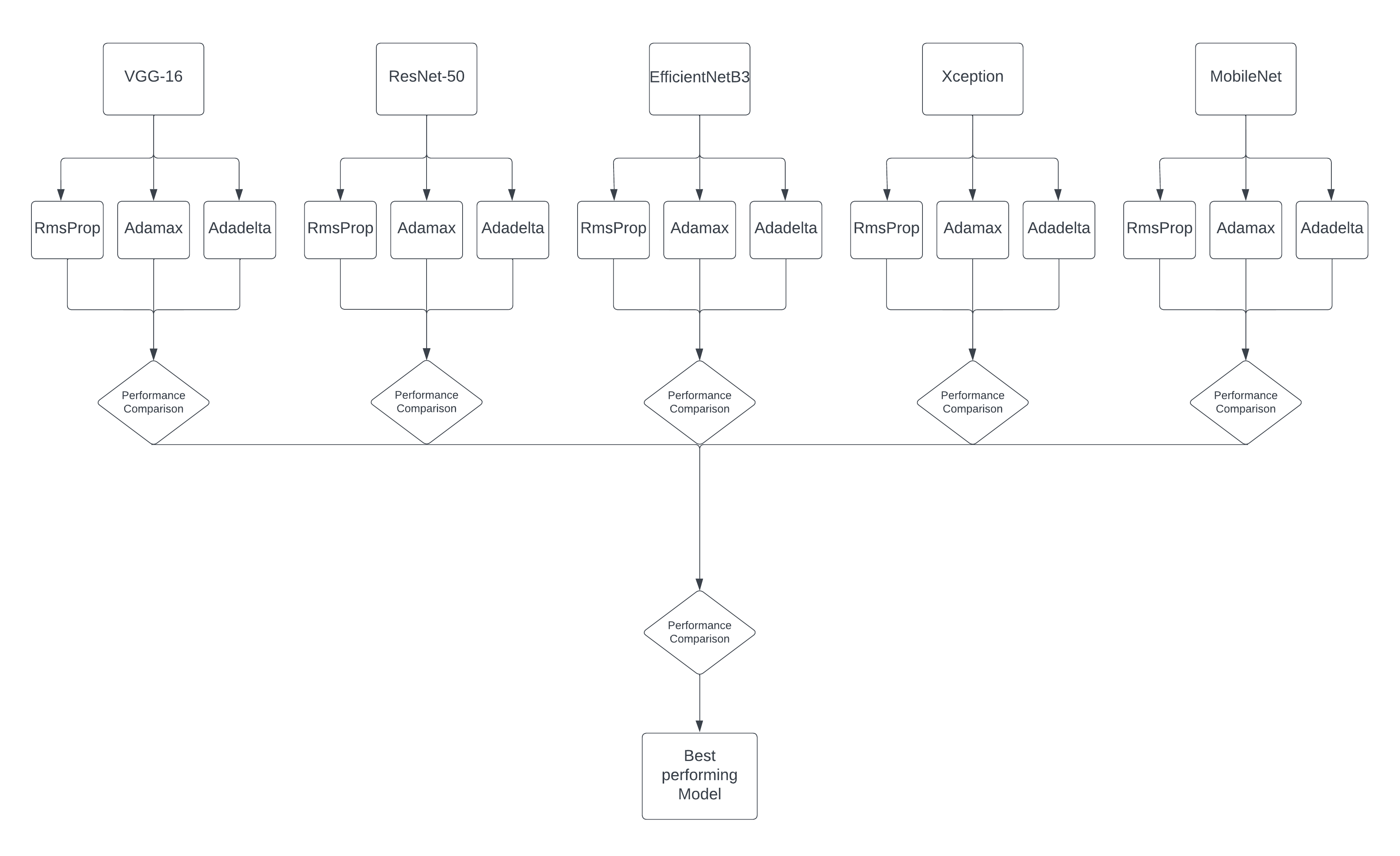


Figure 6. The methodology of this research.

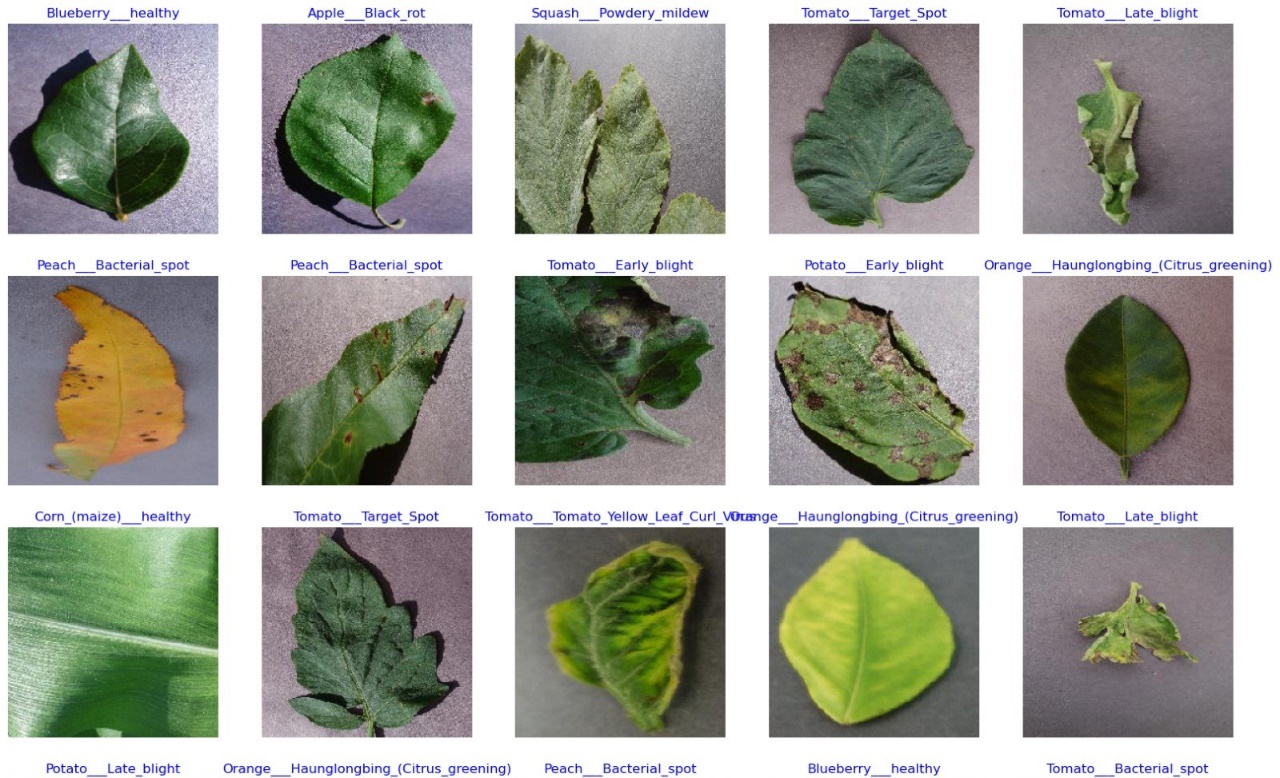


Figure 7. Some of the plant diseases from the PlantVillage dataset.

## Deep Learning Optimizers:

All the DL models were trained using 3 state-of-the-art deep learning optimizers. Some characteristics of these optimizers are provided as follows:

1. RMSProp: Learning rates were adjusted individually for each parameter during training. This was achieved by maintaining a moving average of squared gradients for each parameter, thus normalizing updates. This adaptability enabled faster convergence and stability in training deep neural networks (Huang, 2020).
2. Adadelta: Adadelta, an extension of the Adagrad algorithm, further addressed its tendency to decrease the learning rate monotonically. This was accomplished by using a sliding window of the squared gradients to compute adaptive learning rates, allowing for more stable and efficient training (Zeiler, 2012).
3. Adamax: Adaptive learning rates for each parameter were computed by maintaining a moving average of the squared gradients. This normalization of learning rates allowed for more stable and efficient training of deep neural networks (Ba & Kingma, 2014).

## Training Specifications:

All DCNN models were trained from scratch on the PlantVillage dataset. The hyperparameters were set following specified guidelines. The internal covariate shift problem was observed in neural networks due to variations in the input data distribution caused by changes in the number of parameters in the previous layers. This problem was addressed using Batch Normalization, a method known for its effectiveness in handling the high learning rate. The specifications of all DL optimizers are summarized in Table 2.

Table 2. Hyperparameters of the deep learning optimizers

|  |  |
| --- | --- |
| Optimizers | Specifications |
| Adamax | Learning rate = 0.002, beta1 = 0.9, beta2 = 0.999, epsilon = 1 \* 10-8 |
| Adadelta | Learning rate = 1.0, rho = 0.95, epsilon = 1 \* 10-6 |
| RMSProp | Learning rate = 0.001, rho = 0.9, epsilon = 1 \* 10-7 |

## Code implementation and explanation:

#### Data Preparation and Splitting:

A screen shot of a computer program

Description automatically generated

Figure 8. Data Splitting - Code Snippet

1. define\_paths(data\_dir)

* This function takes the directory path where the dataset is stored as input.
* It iterates through each fold (subdirectory) in the dataset directory and collects file paths and their corresponding labels.
* File paths and labels are stored in separate lists (file paths and labels).
* Returns the list of file paths and corresponding labels.

2. define\_df (files, classes)

* This function takes two lists as input: files (list of file paths) and classes (list of corresponding labels).
* It creates two pandas Series objects: Fseries for file paths and Lseries for labels.
* Then it concatenates these two Series along the columns axis to form a DataFrame.
* Returns the DataFrame containing file paths and labels.

3. split\_data (data\_dir)

* This function orchestrates the process of splitting the dataset into training, validation, and testing sets.
* It utilizes the previously defined functions define\_paths () and define\_df () to obtain the DataFrame containing file paths and labels.
* Firstly, it creates the training DataFrame by splitting the entire dataset into 80% for training and 20% for validation and testing combined.
* Then, it splits the remaining data (validation and testing combined) equally into validation and testing DataFrames, each containing 10% of the original dataset.
* Stratified splitting is applied to maintain the distribution of classes across splits.
* Returns three DataFrames: train\_df for training, valid\_df for validation, and test\_df for testing.

#### Data Augmentation and Preprocessing using Image Data Generators:

A screen shot of a computer program

Description automatically generated

Figure 9. Data Augmentation - Code Snippet

Function: create\_gens (train\_df, valid\_df, test\_df, batch\_size)

This function prepares data generators for training, validation, and testing data. It utilizes the Keras ImageDataGenerator to perform data augmentation and preprocessing.

1. Defining Model Parameters:

* img\_size: The desired size for input images after resizing.
* channels: The number of color channels in the images (e.g., 3 for RGB).
* color: The colour mode for images ('RGB' or 'grayscale').
* img\_shape: The shape of input images (height, width, channels).

1. Customizing Test Batch Size:

* It calculates a suitable batch size for the test data based on the length of the test DataFrame to ensure efficient memory usage.
* The batch size is dynamically adjusted to fit the test data while keeping it within a reasonable range.

1. Defining Data Augmentation Function:

* A custom function scalar(img) is defined to perform minimal preprocessing on images. In this case, it simply returns the input image unchanged.

1. Creating ImageDataGenerators:

* Two ImageDataGenerator objects (tr\_gen for training and ts\_gen for validation and testing) are instantiated.
* tr\_gen is configured with data augmentation parameters such as horizontal flipping to augment training data.
* ts\_gen is used for validation and testing and does not perform data augmentation.

1. Flowing Data from DataFrames:

* Training, validation, and testing data are fed into the corresponding ImageDataGenerator objects using flow\_from\_dataframe () method.
* DataFrames (train\_df, valid\_df, test\_df) are passed along with relevant parameters such as file paths, labels, target size, colour mode, class mode, shuffle, and batch size.
* For training and validation, shuffle is set to True to shuffle the data for each epoch. For testing, shuffle is set to False to maintain the order of data.

1. Returning Data Generators:

* Returns three data generators: train\_gen, valid\_gen, and test\_gen containing augmented and pre-processed data ready to be fed into the deep learning model.

#### Utilizing Custom Callback in Model Training:

The custom callback class, named as MyCallback, constitutes a pivotal component within the research, offering a suite of customizable functionalities aimed at optimizing the training process. It facilitates real-time monitoring of crucial training metrics, such as accuracy and validation loss, both at the culmination of each epoch and batch iteration. Through this continuous evaluation, insights were gained into the model's performance, enabling informed decisions regarding learning rate adjustments and overall model optimization strategies. Moreover, MyCallback affords a level of flexibility that allows for user interaction during training, enabling intervention, parameter adjustments, or decision making based on real-time observations or experimental requirements.

A computer screen shot of a program

Description automatically generated

Figure 10. Custom Callback - Code Snippet

1. Setting up Callback Parameters:

* The parameters such as patience, stop\_patience, threshold, factor, ask\_epoch, and batches are predefined to configure the behaviour of the custom callback during training.

2. Instantiating the Callback:

* The custom callback MyCallback is instantiated with the defined parameters and passed to the callbacks list.
* This list holds the callback objects that will be executed during model training.

3. Training the Model with Callbacks:

* During model training, the callbacks list is passed to the fit () method of the model.
* This ensures that the defined callback functions are executed at specific points during training.

# Results:

## Introduction to Results:

The study's findings are discussed in this part of the report, with major focus on comparing the effectiveness of the DCNN architectures trained with the 3 optimizers for the goal of predicting leaf disease. The goal is to find out which architecture and optimizer combination produced the greatest overall performance and maximum accuracy in classifying leaf diseases using detailed experimentation and analysis.   
The investigation's main focuses were covered by the results that were given. First, per-epoch data analysis revealed information about each model's training progress, including adjustments to learning rate, accuracy, and loss. Through this study, it was possible to see how well each model learnt from and adjusted to the training set over the course of several epochs.

Each model's performance in training, validating, and testing was also evaluated, and measures like accuracy and loss were looked at to see how well the models generalized to new data. Important insights into the robustness and generalization capacities of the trained models were obtained by comparing performance across several datasets.   
In addition, a thorough examination of the classification reports produced by each model was generated, examining parameters like F1-score, precision, and recall for distinct classes. This research allowed for the discovery of each model's strengths and weaknesses by providing a detailed understanding of how well it classified particular leaf diseases.

The study sought to determine which DCNN architecture(s) and optimizer(s) provided the most promising outcomes for leaf disease prediction by thoroughly comparing the models.   
The findings were thoroughly analysed, their ramifications were explored, and possible directions for further research in this area were indicated in the sections that followed.

## Training Progress:

The dynamics of the training process can be better understood by examining the training times for each architecture using a variety of optimizers. EfficientNetB3 showed significant differences in training times among the architectures based on the optimizer that was used. Notably, Adadelta yielded the least training time of 101 minutes, whereas RMSProp produced the longest training time of 151 minutes. This discrepancy implies that each optimizer has different optimization efficiencies and convergence rates, which affect the total training time needed for model convergence.

By comparison, MobileNet showed impressive uniformity in training durations for all optimizers, with the lowest variation found to be between 50 and 54 minutes. This consistency points to a strong optimization process in which the intrinsic properties of the design support steady and effective model convergence. ResNet50 also showed a similar pattern, albeit Adadelta had the longest training period at 140 minutes, as opposed to 116 minutes for Adamax and 108 minutes for RMSProp. This disparity highlights the impact of the optimizer on training effectiveness, with some optimization techniques showing better convergence performance for topologies.

In addition, VGG-16 demonstrated comparatively consistent training times for all optimizers, spanning from 118 to 126 minutes. This consistency points to a well-balanced optimization process involving many optimization techniques, wherein the optimization of parameters and the complexity of the architecture converge effectively. On the other hand, out of all the architectures, Xception showed the most differences in training times; RMSProp produced the longest training period, at 236 minutes, followed by Adadelta (210 minutes) and Adamax (154 minutes). This discrepancy highlights how sensitive some structures are to optimization techniques; more complex models necessitate longer optimization procedures to reach convergence.

All things considered, the training time study clarifies the complex relationship among architecture complexity, optimizer choice, and training effectiveness. A thorough analysis of training times for different optimizers and designs provides important information about the parameters affecting computing resource needs and model convergence. These results advance our knowledge of the deep learning optimization process and guide future attempts to design models with the goal of improving training effectiveness and convergence rates.

Table 3.Training Time Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Training time with Adamax | Training time with Adadelta | Training time with RMSProp |
| EfficenetNetB3 | 111 | 101 | 151 |
| MobileNet | 50 | 54 | 53 |
| ResNet50 | 116 | 140 | 108 |
| VGG-16 | 119 | 126 | 118 |
| Xception | 154 | 210 | 236 |

## Training, Validation, and Testing Performance:

The training accuracy and loss metrics for every CNN architecture that was trained using various optimizers are displayed in Table 4. With scores ranging from 0.99952 to 0.99991, Adamax consistently produced high training accuracy across architectures. Excellent results were also obtained using Adadelta, whose training accuracy ranged from 0.90652 to 0.99984. With values ranging from 0.95424 to 0.99979, RMSProp shows a slightly worse training accuracy than the other optimizers.

All optimizers and architectures show a similar pattern in terms of training loss. Lower training loss values were typically the outcome of using Adadelta and Adamax, indicating successful optimization and model convergence. For some designs, like VGG-16, where the loss value reached an exceptionally high value of 1,498,191.375, RMSProp showed larger training loss values. This anomaly points to possible problems with the stability or convergence of the optimization for the VGG-16 when trained using RMSProp.

All things considered; the training performance measurements shed light on how well various optimizers support model convergence. While Adadelta and Adamax consistently produce minimal training loss and excellent training accuracy across architectures, RMSProp's performance is inconsistent, with rare disparities seen, especially in VGG-16 situations. These results highlight how crucial it is to choose optimizers carefully in order to achieve the best model performance possible during training. To clarify the underlying causes of these variances and anomalies in training performance, more examination and research may be necessary.

Table 4. Training Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Architectures | Optimizers | Training Accuracy | Training Loss |  |
| EfficientNetB3 | Adadelta | 0.99984 | 0.09715 |  |
| Adamax | 0.99982 | 0.12424 |  |
| RMSProp | 0.99965 | 0.06087 |  |
| MobileNet | Adadelta | 0.99961 | 0.10478 |  |
| Adamax | 0.99991 | 0.12012 |  |
| RMSProp | 0.99726 | 0.10817 |  |
| ResNet50 | Adadelta | 0.99965 | 0.11693 |  |
| Adamax | 0.99991 | 0.14739 |  |
| RMSProp | 0.99846 | 0.10754 |  |
| VGG-16 | Adadelta | 0.98340 | 0.27617 |  |
| Adamax | 0.99636 | 0.21265 |  |
| RMSProp | 0.95424 | 1498191.37500 |  |
| Xception | Adadelta | 0.90652 | 7.82996 |  |
| Adamax | 0.99952 | 0.12464 |  |
| RMSProp | 0.99979 | 0.05484 |  |

The validation accuracy and loss metrics for every CNN architecture trained with various optimizers are shown in Table 5. Adadelta and Adamax consistently produced good validation accuracy results across architectures, with values ranging from 0.89448 to 0.99908. With results ranging from 0.94328 to 0.99687, RMSProp likewise delivered acceptable validation accuracy, albeit marginally lower than Adadelta and Adamax.

For validation loss, all optimizers and architectures follow a similar pattern. Lower validation loss values were typically obtained by Adadelta and Adamax, suggesting strong model generalization and resilience to new data. But on occasion, RMSProp produced larger validation loss values; this was especially noticeable in architectures like VGG-16 and Xception. This may point to overfitting tendencies and possible problems with model generalization when trained with RMSProp.

A comparison of training and validation performance can be used to determine if overfitting or underfitting is present. Overfitting, in which the model learns to memorize the training data instead of generalizing to unseen data, is suggested if the training accuracy is noticeably higher than the validation accuracy. On the other hand, low training, and validation accuracy points to underfitting, in which the model is unable to identify the underlying patterns in the data.

After examination, it seems that the models trained with Adamax and Adadelta performed rather well overall, with training and validation accuracy nearly matching. On the other hand, disparities between training and validation performance were seen in a few models, most notably VGG-16 and Xception trained with RMSProp, indicating possible overfitting. The intricacy of these architectures and the optimization dynamics of RMSProp could be to blame for this mismatch. The latter could have caused unduly aggressive parameter updates during training, which would have overfitted the training set. To reduce overfitting and enhance these models' generalization capabilities, more research and regularization strategies would be necessary.

Table 5.Validation Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Architectures | Optimizers | Validation Accuracy | Validation Loss |  |
| EfficientNetB3 | Adadelta | 0.99908 | 0.10204 |  |
| Adamax | 0.99834 | 0.12952 |  |
| RMSProp | 0.99834 | 0.06790 |  |
| MobileNet | Adadelta | 0.99632 | 0.11667 |  |
| Adamax | 0.99613 | 0.13330 |  |
| RMSProp | 0.99079 | 0.13398 |  |
| ResNet50 | Adadelta | 0.99466 | 0.13368 |  |
| Adamax | 0.99687 | 0.15594 |  |
| RMSProp | 0.98950 | 0.12866 |  |
| VGG-16 | Adadelta | 0.97864 | 0.27381 |  |
| Adamax | 0.99006 | 0.22947 |  |
| RMSProp | 0.94328 | 0.33017 |  |
| Xception | Adadelta | 0.89448 | 7.85990 |  |
| Adamax | 0.99576 | 0.14055 |  |
| RMSProp | 0.99687 | 0.06476 |  |

The testing accuracy and loss metrics for each CNN architecture trained with various optimizers are shown in Table 6. With scores ranging from 0.89413 to 0.99871, all optimizers consistently produced good testing accuracy across architectures. Comparably, testing loss values were generally modest, indicating that the model performed well on untested data.

On the other hand, assessing loss values for several architectures trained using RMSProp show significant differences. Testing loss values for VGG-16 and Xception are abnormally high, reaching 9,670,691 and 7.85561, respectively. These anomalies point to possible problems with the stability and generalization of the model after training with RMSProp, which calls for more research into the regularization strategies and optimization dynamics used.

After examination, it seems that most models perform in a balanced manner, with measures for training and validation accuracy closely matching testing accuracy. On the other hand, disparities between training and testing performance are seen in VGG-16 and Xception trained with RMSProp, indicating possible overfitting tendencies. These architectures' intricacy and RMSProp's optimization dynamics may have caused excessively aggressive parameter updates during training, which overfit the training set.   
In summary, even though most models perform well on the testing dataset, more research into regularization techniques and optimization strategies is necessary to address overfitting and enhance generalization performance on untested data, given the anomalies found in VGG-16 and Xception trained with RMSProp.

Table 6. Testing Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Architectures | Optimizers | Testing Accuracy | Testing Loss |
| EfficientNetB3 | Adadelta | 0.99871 | 0.10284 |
| Adamax | 0.99816 | 0.13038 |
| RMSProp | 0.99853 | 0.06678 |
| MobileNet | Adadelta | 0.99816 | 0.11370 |
| Adamax | 0.99705 | 0.13106 |
| RMSProp | 0.99871 | 0.12558 |
| ResNet50 | Adadelta | 0.99705 | 0.12745 |
| Adamax | 0.99632 | 0.15948 |
| RMSProp | 0.99319 | 0.12347 |
| VGG-16 | Adadelta | 0.97606 | 0.27861 |
| Adamax | 0.99282 | 0.22509 |
| RMSProp | 0.93795 | 9.67069 |
| Xception | Adadelta | 0.89413 | 7.85561 |
| Adamax | 0.99761 | 0.13095 |
| RMSProp | 0.99871 | 0.06189 |
|  |  |  |  |

## Classification Report Analysis:

The accuracy, recall, and F1-scores of every CNN architecture that was trained using a variety of optimizers offered important information about how well the models classified plant diseases. The accuracy of recognizing positive cases—that is, correctly identifying diseased plants—was used as a proxy for precision. Conversely, recall was quantified as the capacity to record every instance of positive cases, guaranteeing that no plants with disease conditions escaped detection. The F1-score provided a fair evaluation of a model's capacity to correctly identify plant diseases since it is the harmonic mean of precision and recall.

The precision scores for each architecture represented the percentage of accurately diagnosed diseased plants among all plants that were projected to be diseased. Adadelta and RMSProp-trained models regularly displayed excellent precision scores, indicating that they are useful for correctly diagnosing sick plants. Notably, Xception and MobileNet that were trained with RMSProp continuously attained flawless precision, demonstrating their ability to identify damaged plants accurately and without misclassification.   
In a similar vein, recall scores demonstrated how well the models identified every occurrence of a diseased plant among the dataset's real infected plants. Adadelta and RMSProp-trained models regularly obtained good recall ratings, demonstrating their capacity to recognize most sick plants. MobileNet continuously attained perfect recall when trained using RMSProp, proving its capacity to accurately identify all cases of unhealthy plants without making any omissions.

Moreover, the F1-scores provided an extensive evaluation of the models' overall efficacy in the classification of plant diseases. Although memory and precision have unique significance, the F1-score offered a fair assessment that gave equal weight to each. In general, models trained with Adamax and RMSProp showed higher F1-scores than models trained with Adadelta, suggesting a more equitable trade-off between recall and precision.   
It can be concluded that the models trained with various optimizers provided useful information on how well they classified plant illnesses based on their precision, recall, and F1-scores. These metrics were important measures of model performance that helped researchers choose the best architectures and optimizers for tasks involving the categorization of diseases, ultimately leading to improvements in the detection and treatment of plant diseases.

Table 7.Classification Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Architectures | Optimizers | Precision | Recall | F1-Score |
| EfficientNetB3 | Adadelta | 0.970 | 0.980 | 0.975 |
| Adamax | 0.960 | 0.980 | 0.970 |
| RMSProp | 1.000 | 0.980 | 0.990 |
| MobileNet | Adadelta | 0.960 | 1.000 | 0.980 |
| Adamax | 0.965 | 0.950 | 0.957 |
| RMSProp | 0.990 | 0.990 | 0.99 |
| ResNet50 | Adadelta | 0.980 | 1.000 | 0.99 |
| Adamax | 1.000 | 0.97 | 0.98 |
| RMSProp | 0.990 | 0.990 | 0.99 |
| VGG-16 | Adadelta | 0.980 | 0.980 | 0.98 |
| Adamax | 0.990 | 0.990 | 0.99 |
| RMSProp | 0.950 | 0.940 | 0.94 |
| Xception | Adadelta | 0.890 | 0.890 | 0.88 |
| Adamax | 1.000 | 1.000 | 1.00 |
| RMSProp | 1.000 | 1.000 | 1.00 |
|  |  |  |  |  |

# Conclusion:

## Future Directions:

This study suggests prospective avenues for more investigation and development in the categorization of plant diseases. The research proposes using pre-trained models in conjunction with transfer learning approaches to improve classification efficiency and accuracy. It also suggests using sophisticated data augmentation techniques to broaden and diversify the training dataset, which could strengthen the robustness of the model.   
In order to improve overall classification performance, the research plans on investigating ensemble learning methodologies, which combine predictions from different models. It also emphasizes how crucial it is to incorporate strategies like gradient-based attribution methods and attention processes to improve the interpretability of the model.

The research suggests investigating domain adaptation techniques to make sure that models generalize to different environmental situations. These techniques try to balance differences between training and deployment domains. It also emphasizes the need of optimizing models for realistic deployment in real-world scenarios, with a focus on developing efficient and lightweight models appropriate for edge deployment.   
Growing datasets and integrating categorization models with decision support systems require cooperation with stakeholders and domain experts. The goal of this cooperative Endeavor is to offer timely analysis and suggestions for efficient disease control in agriculture.

The paper intends to advance the field of plant disease classification by addressing these directions for future research and development. This will eventually lead to the creation of more precise, understandable, and implementable solutions for disease diagnosis and management in agricultural contexts.

## Summary of Results:

An overview of the research’s main conclusions provides insight into how well convolutional neural network (CNN) architectures trained with different optimizers perform when it comes to predicting leaf disease using the Plant Village dataset.

Training Metrics:

* Adamax consistently yielded high training accuracy, ranging from 0.99952 to 0.99991 across architectures.
* Adadelta also produced excellent training accuracy, ranging from 0.90652 to 0.99984.
* RMSProp showed slightly lower training accuracy, ranging from 0.95424 to 0.99979, with occasional disparities.

Validation Metrics:

* Adadelta and Adamax consistently achieved good validation accuracy, ranging from 0.89448 to 0.99908.
* RMSProp delivered acceptable validation accuracy, albeit marginally lower than Adadelta and Adamax, with values ranging from 0.94328 to 0.99687.

Testing Metrics:

* All optimizers consistently produced good testing accuracy, ranging from 0.89413 to 0.99871.
* Significant disparities in testing loss values were observed for architectures trained with RMSProp, indicating potential issues with stability and generalization.

Precision, Recall, and F1-scores:

* Adadelta and RMSProp-trained models consistently exhibited excellent precision and recall, with notable performance from Xception and MobileNet trained with RMSProp.
* F1-scores suggested a more equitable trade-off between recall and precision for models trained with Adamax and RMSProp compared to Adadelta.

The study's overall findings emphasize how crucial optimizer selection is to obtain the best possible model performance throughout training. Although Adadelta and Adamax consistently performed well across metrics, there were sometimes differences with RMSProp, especially in architectures such as VGG-16 and Xception. In order to address potential overfitting difficulties and improve generalization performance, our findings indicate the necessity for additional study into regularization approaches and optimization tactics.   
  
Researchers were also able to choose the best architectures and optimizers for plant disease categorization tasks with the help of accuracy, recall, and F1-scores, which offered insightful information about the models' classification abilities. These indicators are crucial for assessing the effectiveness of the model and ultimately help to enhance the identification and management of plant diseases.

## Significance of the Study:

In the field of plant pathology and agricultural research, this work is highly significant since it provides important information about how to develop and optimize DCCN models for the classification of leaf diseases.   
The research fills a significant vacuum in the literature on the best way to choose model configurations for plant disease prediction tasks by methodically contrasting the performance of multiple DCNN architectures trained with different optimizers. For academics and practitioners looking to implement precise and dependable disease detection systems in agricultural contexts, the researchers have produced practical recommendations through thorough experimentation and analysis of training, validation, and testing metrics.

Farmers, agronomists, and agricultural regulators should take note of the study's practical implications as fast and accurate diagnosis of plant diseases is crucial to crop health and output. Through the use of cutting-edge deep learning methods, like DCNNs, stakeholders can reduce the financial losses resulting from agricultural diseases and maximize resource allocation for disease control strategies by automating the disease identification process.   
Moreover, the research advances our knowledge of the fundamental variables affecting model performance in disease categorization tasks, which benefits the larger scientific community. The thorough examination of precision, recall, and F1-scores offers insightful information about the advantages and disadvantages of various DCNN designs and optimization techniques, which will help guide future studies focused on enhancing the performance of illness detection systems.

Furthermore, the experimental protocols and methods described in this work are a great help to other researchers who are starting similar studies in the field of agricultural deep learning. Collaboration among researchers is encouraged by the study's transparent documentation of model training protocols, data preprocessing steps, and performance evaluation measures.

To sum up, this research contributes to our comprehension of the use of deep learning in plant pathology and provides useful strategies to deal with actual problems in agricultural disease control. Researchers may continue to innovate and create more effective techniques for ensuring global food security in the face of growing environmental dangers and socioeconomic issues by utilizing state-of-the-art technology and interdisciplinary collaboration.

# References

Adam, H. et al., 2017. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* [Online]   
Available at: https://arxiv.org/abs/1704.04861  
[Accessed April 2024].

Ahmed, S. U., Shuja, J. & Tahir, M. A., 2023. *Leaf classification on Flavia dataset: A detailed review.* [Online]   
Available at: https://www.sciencedirect.com/science/article/abs/pii/S2210537923000628  
[Accessed March 2024].

Ba, J. & Kingma, D. P., 2014. *Adam: A Method for Stochastic Optimization.* [Online]   
Available at: https://arxiv.org/abs/1412.6980  
[Accessed March 2024].

Chollet, F., 2016. *Xception: Deep Learning with Depthwise Separable Convolutions.* [Online]   
Available at: https://arxiv.org/abs/1610.02357  
[Accessed April 2024].

Ferentinos, K. P., 2018. *Deep learning models for plant disease detection and diagnosis.* [Online]   
Available at: https://www.sciencedirect.com/science/article/abs/pii/S0168169917311742  
[Accessed March 2024].

He, K., Zhang, X., Ren, S. & Sun, J., 2015. *Deep Residual Learning for Image Recognition.* [Online]   
Available at: https://arxiv.org/abs/1512.03385  
[Accessed April 2024].

Huang, J., 2020. *RMSProp.* [Online]   
Available at: https://optimization.cbe.cornell.edu/index.php?title=RMSProp  
[Accessed March 2024].

Hughes, M. & Salathe, M., 2015. *PlantVillage Dataset.* [Online]   
Available at: https://paperswithcode.com/dataset/plantvillage

Krizhevsky, A., Sutskever, I. & Hinton, G. E., 2012. *ImageNet Classification with Deep Convolutional Neural Networks.* [Online]   
Available at: https://papers.nips.cc/paper\_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html  
[Accessed March 2024].

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998. *Gradient-based learning applied to document recognition.* [Online]   
Available at: https://ieeexplore.ieee.org/abstract/document/726791  
[Accessed March 2024].

Le, Q. V. & Tan, M., 2019. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* [Online]   
Available at: https://arxiv.org/abs/1905.11946  
[Accessed April 2024].

Levi, G. & Hassner, T., 2015. *Age and Gender Classification Using Convolutional Neural Networks.* [Online]   
Available at: https://talhassner.github.io/home/publication/2015\_CVPR  
[Accessed March 2024].

Mohanty, S. P., Hughes, D. P. & Salathe, M., 2016. *Using Deep Learning for Image-Based Plant Disease Detection.* [Online]   
Available at: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2016.01419/full  
[Accessed March 2024].

Mollahosseini, A. et al., 2016. *Facial Expression Recognition from World Wild Web.* [Online]   
Available at: https://arxiv.org/abs/1605.03639  
[Accessed March 2024].

Nevavouri, P., Narra, N. & Lipping, T., 2019. *Crop yield prediction with deep convolutional neural networks.* [Online]   
Available at: https://www.sciencedirect.com/science/article/abs/pii/S0168169919306842  
[Accessed March 2024].

Saleem, M. H., Potgieter, J. & Arif, K. M., 2020. *Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers.* [Online]   
Available at: https://www.mdpi.com/2223-7747/9/10/1319  
[Accessed March 2024].

Simonyan, K. & Zisserman, A., 2014. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* [Online]   
Available at: https://arxiv.org/abs/1409.1556  
[Accessed April 2024].

Zeiler, M. D., 2012. *ADADELTA: An Adaptive Learning Rate Method.* [Online]   
Available at: https://arxiv.org/abs/1212.5701  
[Accessed March 2024].