Image Data Augmentation Methods with Deep Learning Neural Networks

Mohammad Zakir Hossain

Master's thesis

School of Computing

Computer Science

June 2024

Abstract:

Image data enhancement is a vital method in the field of deep learning for enhancing model performance and generalization capabilities. This thesis explores various image data augmentation methods and their effectiveness in improving the performance of deep learning neural networks.

The study begins with an overview of image data augmentation techniques, including rotation, flipping, scaling, cropping, translation, brightness adjustment, contrast adjustment, and Gaussian noise addition. Each augmentation method is examined in detail, discussing its impact on the variety and richness of the training collection.

Following the theoretical overview, a series of experiments are conducted using widely used deep learning models such as Convolutional Neural Networks (CNNs), VGG, and MobileNet. These experiments aim to evaluate the performance of different augmentation methods in terms of model accuracy, robustness, and generalization on benchmark datasets.

The results of the experiments demonstrate the effectiveness of certain augmentation techniques in improving model performance, while also highlighting potential trade-offs and considerations for real-world applications. Additionally, the study investigates the impact of augmentation parameters such as magnitude, probability, and combination strategies on model training dynamics and convergence.

This thesis provides valuable insights into the role of image data augmentation in deep learning and offers practical guidance for selecting and implementing augmentation methods to enhance the effectiveness of neural networks across various computer vision assignments.

Keywords: Class Imbalance, Sampling Techniques, Oversampling, Undersampling, Machine Learning, Evaluation, Artificial Neural Network, Image Data Augmentation, Image Processing, Deep Learning Neural Network, Image Augmentation.

Acknowledgments

I am deeply grateful to the University of Eastern Finland (UEF) for giving me the chance to join the IMPIT course. Additionally, my extend thanks to the professors who have contributed to my understanding of various aspects of computer science. During my term at UEF, I have gained invaluable knowledge and skills.

I am especially grateful to my supervisor, Professor Xiao-Zhi Gao for his invaluable guidance and support during my studies. His insights into my research have significantly enhanced my proficiency and expertise in computer science. Professor Xiao-Zhi Gao has been instrumental in addressing numerous technical inquiries throughout my master's program, aiding me in completing my thesis.

Special thanks are also due to Jukka Pitkänen, a IT Specialist at UEF, and Oili Kohonen, whose assistance was indispensable in my academic journey at UEF.

I desire to extend my sincere appreciativeness to my mother, Halima Khanam, and my wife, Sharmin Sultana, for their solid moral assistance and encouragement. I am immensely grateful for everything my parents have done for me, and I have strived to fulfill my mother's and spouse lifelong aspiration of securing a master's degree. I am always be grateful to my family and my life partner, Sharmin Sultana, for their enduring support and encouragement.

List of abbreviations

ANN	Artificial Neural Network
VGG	Visual Geometry Group
CNN	Convolution Neural Network
MobileNet	Mobile Network
ADAM	Adaptive Momentum Optimizer
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural n=Networks
GAN	Generative Adversarial Networks
ReLU	Rectifier Linear Unit
SGD	Stochastic Gradient Descent
Conv2D	2-Dimensional Convolutional Layer
GPU	Graphical Processing Unit
CPU	Central Processing Unit
TPU	Tensor Processing Units

List of Tables

Table 1. Single Data Augmentation Method CNN Model.....36
Table 2. Single Data Augmentation Method with Changing Hyperparameters CNN Model...…38
Table 3. Double Data Augmentation Method CNN Model ……..40
Table 4. Single Data Augmentation Method VGG16 Model…….....42
Table 5. Single Data Augmentation Method MobileNetV2 Model……...44
Table 6. Double Data Augmentation Method MobileNetV2 Model…….46

List of Figures

Figure 1. Exploring and understanding the principles and methodologies behind image data augmentation 8
Figure 2. Illustration of CNN architecture for image classification 11
Figure 3. The representation of RGB channels 17 11
Figure 4. Convolutional architecture layer. Image source: IBM 2020 12
Figure 5. The Max pooling operation 14
Figure 6. The Max pooling operation 14
Figure 7. A fully connected layer. Figure adapted from Mao, et al. 2018 15
Figure 8. Classes of activation functions: (a)Sigmoid, (b)Tanh, (c) ReLU,and (d) LReLU 16
Figure 9. A residual block of ResNet 18
Figure 10: Importing a VGG 16 model from Keras 21
Figure 11. The augmentation techniques employed in the study 22
Figure 12. The channel shifting augmentation technique 26
Figure 13. The CNN architecture 28
Figure 14. The MobileNet architecture 30
Figure 15. The VGG architecture 32
Figure 16. The MobileNet architecture 33
Figure 17. The summary of the implementation of models 34

Table of Contents

1 Introduction 8

2 Related Work 9

2.1 Deep Learning 9

2.2 Data Augmentation..... 10

2.3 Convolutional neural network..... 10

2.3.1 Convolutional layers..... 12

2.3.2 Pooling layer..... 13

2.3.3 Complete Connected Layer..... 15

2.3.4 Activation Function..... 16

2.3.5 Optimization and loss function..... 17

2.3.6 Residual neural network..... 18

2.3.7 Residual Block..... 19

2.3.8 Batch Normalization..... 19

2.3.9 Hyper-parameter..... 20

2.4 VGG Model..... 21

2.4.1 VGG-16 Architecture..... 21

2.5 MobileNetV2 Model..... 23

3 Framework..... 25

3.1 Data preparation and Preprocessing.....25

3.1.1 Datasets..... 25

3.1.2 Data pre-processing..... 26

3.1.3 Augmentation..... 27

3.2 Set-up specification..... 32

3.2.1 Hardware..... 32

3.2.2 Software..... 32

3.3 Model Architecture and Implementation..... 33

3.3.1 CNN Architecture..... 33

3.3.2 VGG Atchitecture 34

3.3.3 MobileNet Atchitecture.....36

3.3.4 Implementation of methods.....37

4 Outcome and Discussion.....39

4.1 Results of CNN Model with Different Combination Augmentation 39

4.1.1 Single Data Augmentation Method.....40

4.1.2 Enhancing Model Robustness with Double Data Augmentation.....44

4.1.3 Comparing One augmentation method and two augmentation.....46

4.2 Results of VGG16 model with Single Combination Augmentation.....46

4.2.1 Optimizing Training Efficiency and Augmentation Effectiveness References 48

4.3 Results of MobileNetV2 model with Different Combination Augmentation 48
4.3.1 Outcome of MobileNetV2 model with Single Combination Augmentation 48
4.3.2 Outcome of MobileNetV2 model with Double Combination Augmentation 51
5 Conclusion…… 54
6 References 57

1. INTRODUCTION

Image data augmentation competes a pivotal role in enhancing model robustness and performing in the realm of deep learning neural networks. By artificially expanding the dataset through transformations, such as rotation, scaling, and flipping, augmentation techniques enable models to generalize better to unseen data and improve overall accuracy.

The integration of image data augmentation techniques has become increasingly essential in various applications, ranging from image classification and object detection to medical imaging and satellite imagery analysis. In essence, these techniques allow for the creation of more diverse and representative training datasets, thereby enhancing the capability of model to learn complex patterns and variants present in real-world images.

Moreover, image data enhancement serves as a vital tool for mitigating the challenges posed by inadequate training data, remarkably in scenarios someplace collecting large volumes of labeled images is impractical or costly. By synthesizing new training samples from existing data, augmentation techniques help alleviate issues related to overfitting and improve the generalization performance of model.

In this particular context, exploring and understanding the principles and methodologies behind image data augmentation are essential for practitioners and researchers alike. This introduction sets the stage for delving deeper into the various augmentation techniques, their applications, and their impact on the implementation of deep learning neural systems.

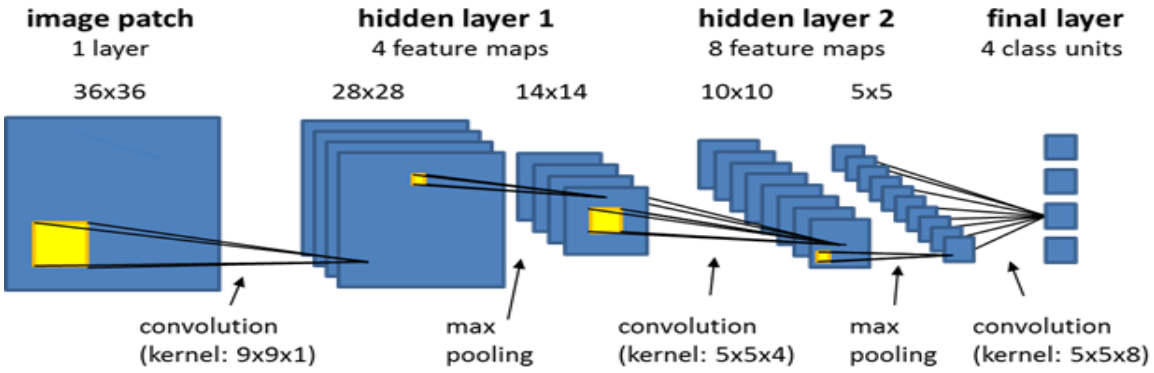

Figure 1. Exploring and understanding the principles and methodologies behind image data augmentation
Source:
https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm

2. Related Work

2.1 DEEP LEARNING

Deep learning, an advanced part of machine learning, is rooted in artificial neural networks. Through deep learning, machines autonomously glean insights from vast datasets by organizing algorithms into layers within artificial neural networks. This method excels in processing extensive datasets and furnishing predictive and precise outcomes (Hatcher & Yu, 2018). Its applications span diverse spheres such as vision recognition, natural semantic processing, speech recognition, biomedicine, and acoustic modeling (Emmert-Streib et al., 2020). Additionally, deep learning powers innovations like self-driving cars, virtual assistants, and sentiment analysis in digital content (Alzubaidi et al., 2021).

Employing neural networks, deep learning amalgamates low-level attributes to extract high-level features, thereby enhancing accuracy in tasks like image recognition compared to traditional methods. Commonly employed networks include convolutional neural network (CNN), multilayer perceptron, and recurrent neural network (RNN) (Lu et al., 2021). Notably, deep learning has found successful applications in medical imaging and clinical diagnostics, delivering precise disease diagnosis and detection, thus significantly impacting healthcare and well-being (Aggarwal et al., 2021).

Deep learning has been utilized for automated classification of radio modulation categories. Two convolutional neural network (CNN)-based deep learning models, GoogleNet [13] and AlexNet [14], originally developed for image classification after convertion radio signals into imagery, are used for modulation classification [15], [16]. By a modified deep residual network (ResNet) [17], the modulation classification accuracy is further advanced, which is fed with modulated in-phase (I) and quadrature phase (Q) signals. The CNN structure also attains a considerable classification accuracy [18] considering channel interference,. In adjunct to the CNN-based models, the Long Short-Term Memory (LSTM) design with time-dependent amplitude and phase statistics can achieve the state-of-the-art classification accuracy [16]. Various subsampling techniques are investigated in [15] to reduce the dimensions of input signals to reduce the training time of deep learning models.

2.2 DATA AUGMENTATION

The importance of data enhancement (Cheng, Benlin, Dong, Shu and Zhenyu) in improving the generalization performance of deep learning models for vision classification. It presents a preliminary examine evaluating the impact of four variables (enhancement method, enhancement rate, size of important dataset per label, and method grouping) on model accuracy. The study provides recommendations based on the experiment's findings, such as the effectiveness of certain augmentation methods and the optimal augmentation rate for training. Additionally, it highlights the influence of method combinations on model performance. Deep learning models require extensive training data to generalize well, leading to challenges in data availability and labeling.

Data enhancement is proposed as a solution to boost both the quantity and variety of training data, thereby enhancing model performance. The study examines ten augmentation methods across triplet datasets: MNIST, Fashion-MNIST, and CIFAR-Guidelines are derived from the experimental results, including the recommendation of specific augmentation methods and optimal augmentation rates.

2.3 Convolutional neural network

A convolutional neural network (CNN) is a sophisticated neural network architecture comprising multiple concealed layers, extensively employed in tasks like pattern recognition and image classification, owing to its capability to analyze images with structured arrays. Typically, CNNs are trained using backpropagation via Stochastic Gradient Descent (SGD), aiming to minimize the loss function by adjusting weights and biases, thereby mapping arbitrary inputs to desired outputs (Albelwi & Mahmood 2017). CNN architectures are structured with interleaved convolutional and pooling layers, culminating in completely connected layers, as depicted in Figure 2. The input layer encodes the pixel values of the image, while convolutional layers identify features within the pixels. Subsequent pooling layers abstract these features, and fully connected layers leverage these acquired features to predict the output.

Compared to traditional neural networks, CNNs offer several advantages, including reduced pre-processing requirements, higher accuracy, simpler implementation at scale, mitigation of overfitting, and improved computational efficiency.

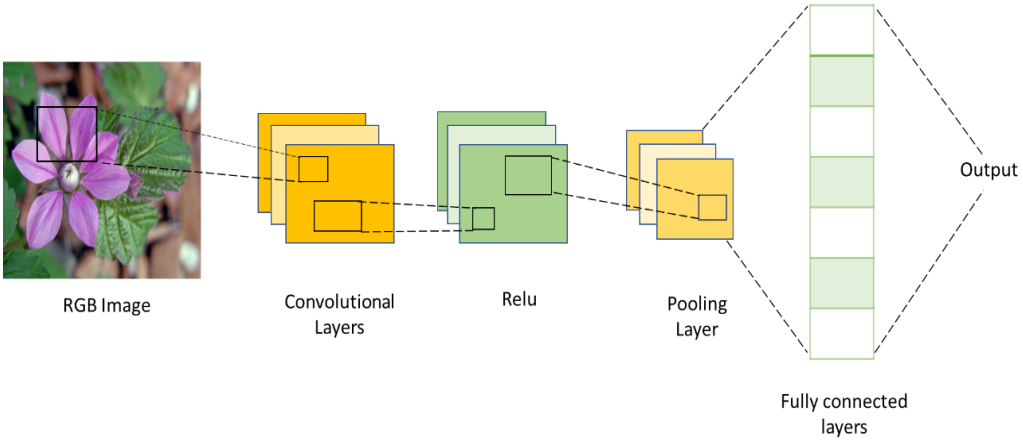

Figure 2. Illustration of CNN architecture for image classification. Figure adapted from Alzubaidi, et al. 2021.

CNNs have primarily found application in image classification tasks. In the case of colored images, particularly those based on the RGB (Red, Green, Blue) model, as illustrated in figure 3, various colors contribute to a three-dimensional input dataset. For instance, in an RGB image with dimensions of 255 x 255 pixels (Width x Height), three matrices correspond to each image, representing each color channel. Consequently, the image is structured as a three-dimensional array known as an Input Volume (255x255x3)

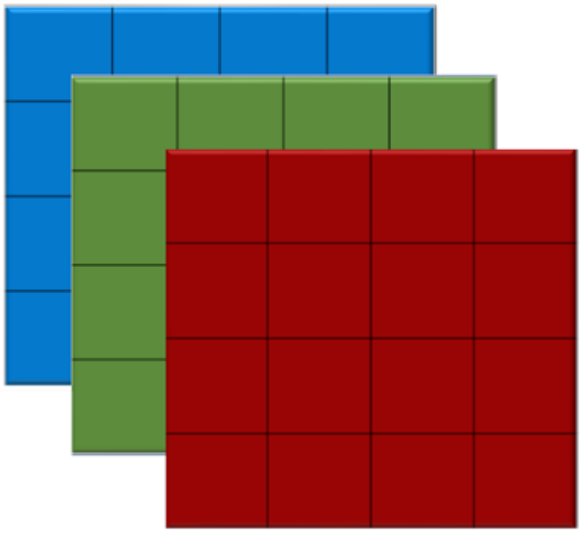

Figure 3. The representation of RGB channels.

Likewise, CNNs are utilized for the categorization of plant species, wherein features are extracted from leaf images. This facilitates an automated computational approach for the efficient and time-effective identification and classification of plants (Sobha & Thomas, 2019).

2.3.1 Convolutional layers

The convolutional layer serves as an underlying component of CNNs, bearing a significant portion of the network's computational burden. It operates as the initial layer responsible for discerning various features within input images. To function, the convolutional layer necessitates input data, a filter, and a feature map. In the case of a colored vision comprising RGB channels, the input possesses trio dimensions: height, width, and depth. The filter, also known as a kernel, assesses the presence of features by traversing the receptive fields of the image. Each segment of the image is denoted by a matrix of weights termed a detector of feature. The size of the filter controls the extent of the sensitive field. Upon applying the filter to a specific region of the imagery, a dot product ensues between the pixels and the filter of the input. Subsequently, the dot product is fed into an output array, after which the filter modifications by a stride. This iterative method continues until the filter traverses the complete image. The resultant output, known as a feature map, provides insights into the image's features. The outcome derived from the input and the filter is referred to as the convolved feature of the feature map. Figure 4, presented below, illustrates a convolutional layer featuring an input image, filter, and output array.

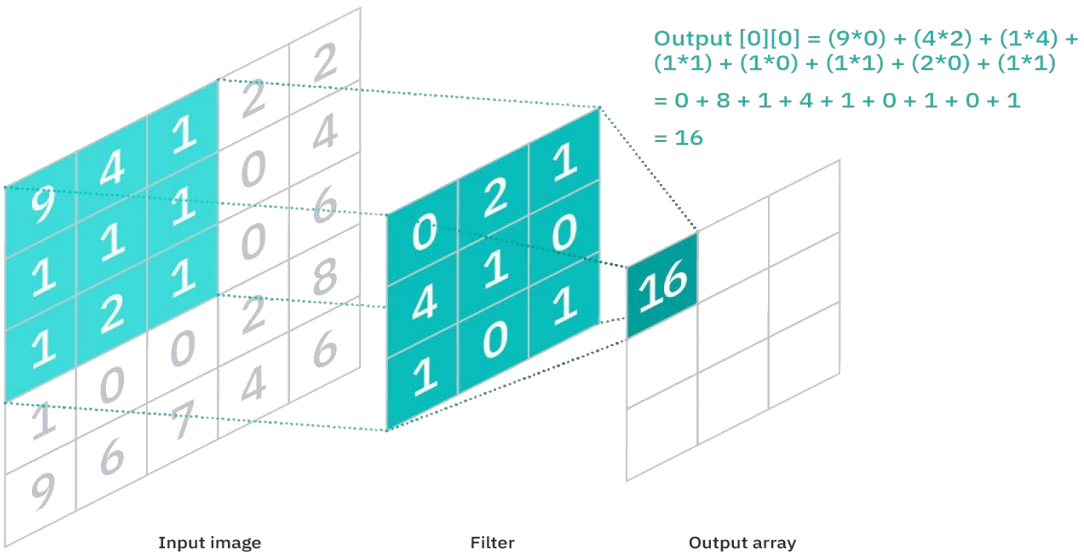

Figure 4. Convolutional architecture layer. Image source: IBM 2020.

Figure 4 illustrates that the feature map of each output value corresponds to the territory of the input where the filter is operated, rather than each individual pixel value in the input vision. Consequently, convolutional layers are often indicated to as sparsely connected layers, as the resultant array does not necessitate a direct mapping to each input value.

Before training the neural network, three parameters that influence the volume size must be established:

The output depth is determined by the number of filters employed.

- A greater stride results in a reduced output. In this context, stride refers to the count of pixels the kernel traverse over the input matrix.
- Zero-padding, which involves surrounding the matrix with zeroes or adding a border of pixels with zero values around the edges of the beginning photographs, is commonly implemented to preserve initial photographs features and regulate the size of output.

There varieties of padding exist:

- Valid padding, also known as no padding, where the convolutional layer remains unpadded, or the last convolution is omitted if the dimensions do not align.
- Same padding, where the original input undergoes padding to confirme the output layer maintains equal in size to the input layer.
- Full padding, which entails adding zeroes to surrounding area of the input, thereby increasing the output dimension.

2.3.2 Pooling Layer

Pooling layers decrease the quantity of input parameters and enable the reduction in sizes of the feature maps. The Pooling operation uses a filter like convolution layer, but it is devoid of weights and the kernel applies aggregate values and fills the output array.

Two foremost types of pooling exist:

Max pooling: In this operation, the element with highest value will be selected by the filter and sent to the output array, thus producing the output with the most vital features of the feature map. The following Figure 5 shows the Max pooling process.

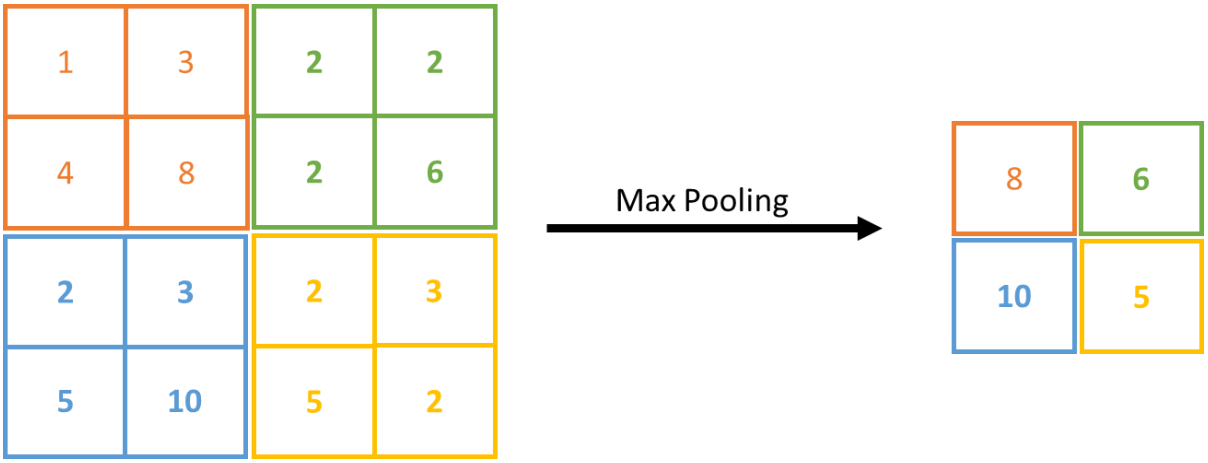

Figure 5. The Max pooling process.

Average pooling: This pooling method calculates the average value of elements within the feature map and forwards it to the output array (IBM 2020). The process of average pooling is illustrated in Figure 6 below

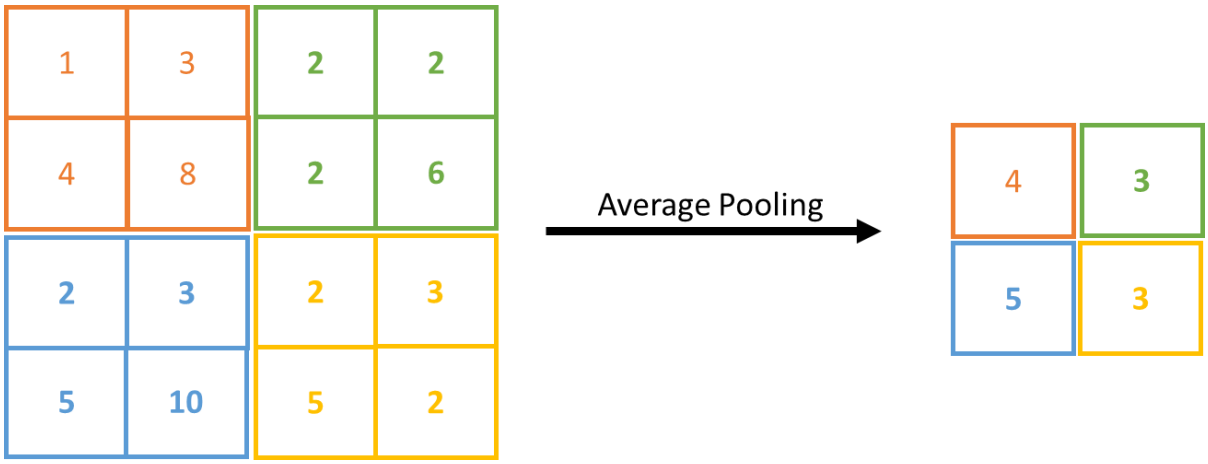

Figure 6. The Max pooling process.

2.3.3 Complete Connected Layer

The complete connected layer constitutes the final layers within the neural network architectural model. It receives the output from the convolutional layer of last pooling as its input and synthesizes the final output by amalgamating the information gathered from preceding layers. Functioning akin to a conventional multi-layer perceptron, a fully connected layer establishes connections between all connections in the earlier layer and those in the subsequent layer. Its role encompasses two primary functions: firstly, learning nonlinear combinations of various features for feature extraction, and secondly, serving as the concluding layer that translates the ultimate feature map into the classification outcome (Mao, et al. 2018). Illustrated in Figure 7 below, the fully connected layer employs the plant image as input. Layer m-1 and layer m depict dual consecutive hidden layers that are completely interconnected. The two nodes within the output layer denote the two predicted plant classes by the CNN model.

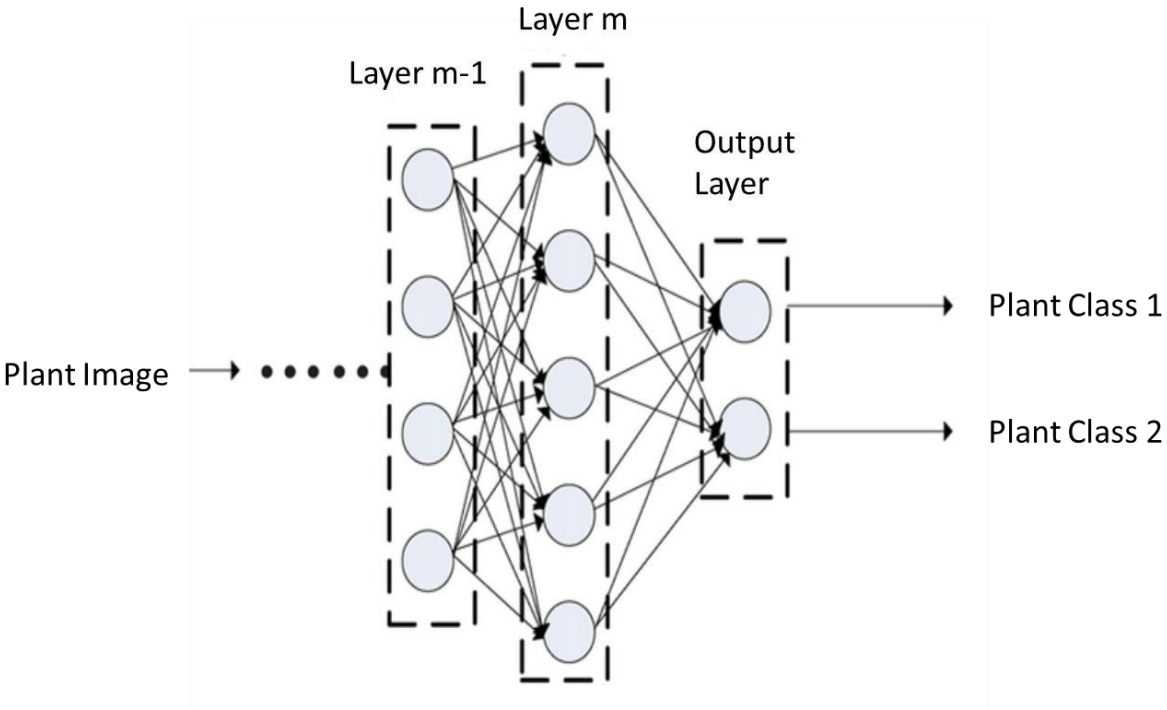

Figure 7. A fully connected layer. Figure adapted from Mao, et al. 2018.

2.3.4 Activation Function

The activation function has a vital role in neural networks lies in deciding whether a neuron should be activate, influenced by the weighted sum of inputs and biases. It produces an output within a defined range, typically between 0 and 1 or -1 and 1, and can be classified into two types: linear and non-linear activation functions. Essentially, it serves to normalize data and introduce nonlinearity into the network, influencing its performance.

Among the common activation functions, Sigmoid, Tanh, and ReLU are notable (Kaloev & Krastev 2021). Sigmoid is effective for output prediction due to its restricted range, facilitating probabilistic decision-making. However, it fails to tackle the issue of gradient diminishing. Tanh, a scaled version of Sigmoid, shares similar limitations.

Rectified Linear Unit (ReLU) stands out as a widely recommended non-linear activation function. It applies a simple transformation to the feature map after each convolution operation, enhancing model nonlinearity. ReLU produces the input value for positive inputs and return zero for negative inputs, making it computationally efficient and easily optimized with gradient-based methods (Gibson & Patterson 2017).

Despite its advantages, ReLU may encounter the "dying ReLU" problem, where neurons deactivate for negative inputs, leading to dead regions in the network. In such cases, Leaky ReLU offers a solution by assigning a small slope to negative inputs, preventing complete deactivation (Xu, et al. 2020). The figure below illustrates the characteristics of varieties activation functions, including Sigmoid, Tanh, ReLU, and Leaky ReLU.

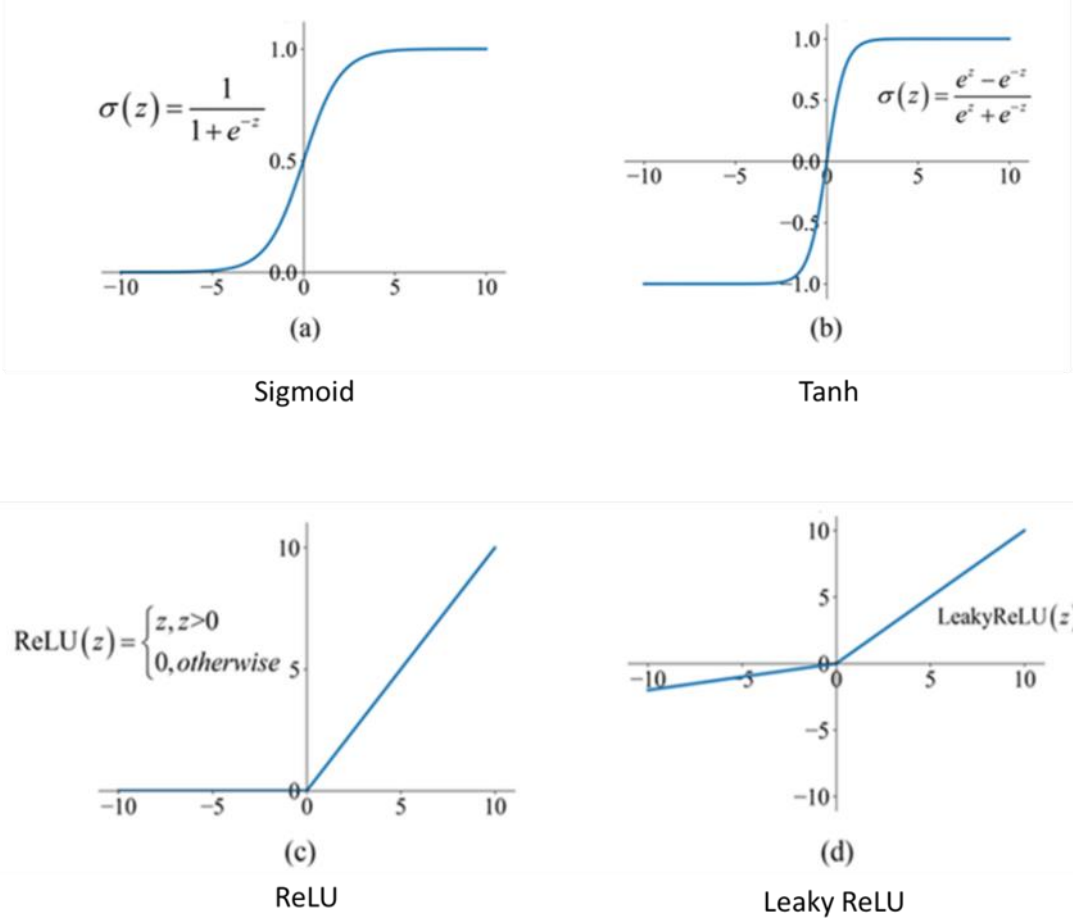

Figure 8. Classes of activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU, and (d) LReLU. Image source: Feng et al. 2019.

2.3.5 Optimization and loss function

Gradient descent is an iterative algorithm utilized to minimize loss by adjusting and updating the learning parameters within the network. It serves as a fundamental technique in training machine learning models and neural networks. Mathematically, the gradient of the loss function represents the partial derivative of the loss with respect to each learnable parameter. A single parameter update is expressed as:

$$w:=w-\alpha\cdot\partial L\partial w \text{w:=w−α·∂w∂L}$$

Here, ww denotes each learnable parameter, αα signifies the learning rate, and LL denotes the loss function.

However, processing the entire dataset with a large sample size during each iteration demands significant computational resources and time. Therefore, a more refined

optimization algorithm, known as Stochastic Gradient Descent (SGD), has been developed and broadly adopted for educating deep neural networks. SGD computes gradients of the loss function with respect to the parameters and applies them to variable updates. It minimizes the cost function by employing Gradient Descent to determine the parameters, with each training data point's parameters being updated by SGD (Bhuiyan et al., 2021). Consequently, SGD demonstrates enhanced computational efficiency compared to traditional gradient descent methods.

Adaptive Moment Estimation (Adam) represents a method for optimizing stochastic gradient descent. Adam estimates individual adaptive learning rates for different parameters determined by the first and second moments of the gradients (Kingma & Ba, 2017). By incorporating bias correction, Adam optimizers achieve superior test accuracy (Rowel, 2018). Moreover, Adam tackles non-convex problems using minimal resources compared to alternative optimization techniques. By amalgamating the advantages of other stochastic gradient methods, such as Adaptive Gradients and Root Mean Square Propagation, Adam introduces a novel learning approach.

2.3.6 Residual neural network

The Residual neural network (ResNet) constitutes an architecture for artificial neural network composed of building blocks known as residual units, which maintain the same connection shape (He et al., 2016). These residual units employ convolution, batch normalization (BN), and ReLU activation functions to learn the residual mapping function (He et al., 2016). By incorporating shortcuts, ResNet addresses the challenge of vanishing gradients, enabling direct backpropagation of gradients to preceding layers. Additionally, ResNet mitigates issues related to covariate shifts and enhances network performance through the utilization of batch normalization (Chandran et al., 2021).

2.3.7 Residual Block

In contrast to traditional networks, where each layer absolutely connects to the successive layer, networks with residual blocks incorporate skip connections, allowing layers to feed into layers located a few blocks away. This technique applies non-linearity by adding the output of the corresponding layer in the main path. The residual block typically contains of a convolution layer succeeded by batch normalization and a ReLU activation function. To summarize the basic residual function: If xx represents the input and $F(x)$F(x) denotes the output from the layer, then the output of the residual block can be expressed as $Y=F(x)+x$Y=F(x)+x. This concept is explained in Figure 9 below.

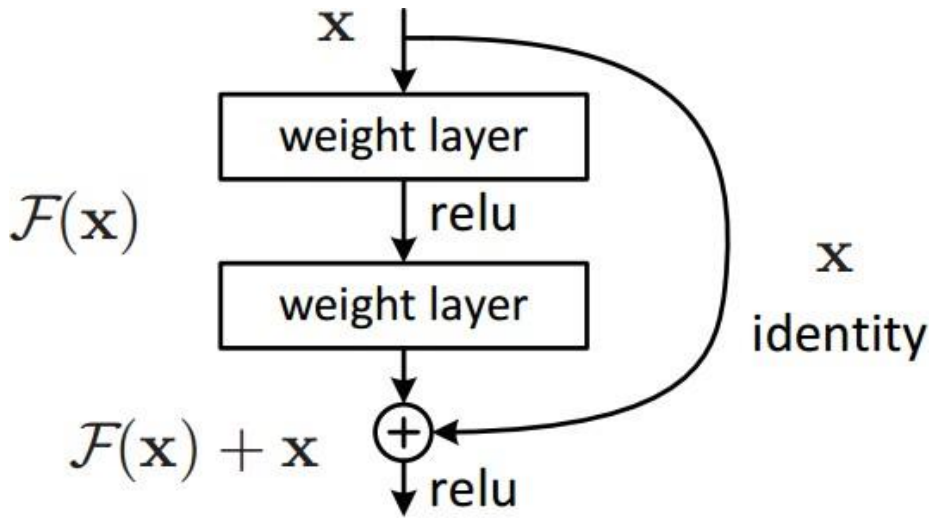

Figure 9. A residual block of ResNet. Image source: He, et al. 2016.

2.3.8 Batch normalization

Batch normalization addresses the challenge of disappearing/exploding gradients that can happen at the time of the training of machine learning algorithms using gradient-based optimization methods. By reducing internal covariate shift, it enhances the speed and stability of neural networks, accelerating the training process. Furthermore, it promotes a smoother gradient flow through the network, minimizing the dependence on initial values or parameter scale. Additionally, batch normalization enables the utilization of saturating non-linearities while preventing network saturation.

The advantages of using batch normalization, as proposed by Khan et al. (2018), include:

- Decreased sensitivity of network training to hyper-parameter choices
- Mitigation of the vanishing gradient problem
- Prevention of activation function saturation, such as with Tanh and sigmoid functions
- Enhanced stability in neural network training
- Improved convergence rate of the network
- Integration of normalization into the network, facilitating error backpropagation.

2.3.9 Hyperparameters

The hyperparameters of deep learning models are predefined parameters that remain constant throughout the process of training, influencing the performance and accuracy of the model. These hyperparameters, including the number of epochs, batch size, learning rate, and architectural elements such as layer size and number of layers, play a crucial role in optimizing model performance.

Epochs, which represent iterations of the training process, determine how frequently the learning algorithm processes the entire dataset for training. Each epoch consists of multiple batches, where the batch size specifies the amount of samples processed before updating internal parameters for model. While substantial batch sizes provide more accurate gradient estimates, memory limitations in parallel processing systems often constrain their size (Brownlee 2018). Conversely, smaller batch sizes offer advantages such as reduced memory usage, increased regularization due to added noise, and faster training as weights are updated more frequently (Kandel & Castelli 2020).

For example, if a dataset contains 300 samples and a batch size of 6 is chosen with 1,200 epochs, the dataset is divided into 50 batches of 6 samples each. With each epoch comprising 50 batches, the model undergoes 1,200 passes through the dataset, totaling 60,000 batches throughout the training process.

Convolutional neural networks propose added hyperparameters such as filter size, padding, filter numbers, stride, and all of which impact model performance. Optimal hyperparameter combinations must be carefully selected to enhance the efficiency and effectiveness of the learning model.

2.4 VGG15 Model

there isn't a widely recognized or standardized "VGG15" model. However, given the naming convention of the VGG (Visual Geometry Group) models, it's likely that "VGG15" refers to a variant or custom implementation of the VGG architecture with 15 layers. The original VGG architectures, VGG16 and VGG19, are well-known for their deep convolutional neural network structures and their effectiveness in image classification tasks.

2.4.1 VGG-16 Architecture

The VGG-16 architecture (Prerepa, Aiswarya, Sufyan, Rahul, and Reena) developed by researchers at the University of Oxford, stands as a stalwart in image recognition and classification tasks due to its simplicity, moderate complexity, and effectiveness. Comprised of 16 layers, involving thirteen convolutional layers, thrice fully connected layers, and five max-pooling layers, VGG-16 operates hierarchically, with each layer building upon the features extracted by its predecessors.

Key Components and Functions:

1. **Convolution and Pooling Layers:** VGG-16 utilizes small 3x3 filters in its convolutional layers to isolate intricate patterns from input images, enabling the network to discern hierarchical representations with fewer parameters. These filters slide through the input data, performing element-wise multiplication and summation to generate feature maps. Max-pooling layers follow specific convolutional layers, reducing spatial dimensions and aiding computational efficiency by retaining maximum values within pooling windows.

2. **Fully Connected Layers:** The final layers of VGG-16 combine extracted features and make class predictions. An activation function of SoftMax is employed in the last totally connected layer to output probabilities for each class.

Advantages of VGG-16:

1. **Simplicity and Understandability:** VGG-16's straightforward architecture, coupled with the use of small filters, facilitates ease of understanding and implementation.
2. **Reduced Computational Complexity:** Compared to other architectures, VGG-16 demands fewer parameters, reducing computational resources required for training and inference.
3. **Transfer Learning Capabilities:** Pre-trained VGG-16 models, trained on datasets such as ImageNet, can be fine-tuned for specific tasks, leveraging existing knowledge and adapting it to new applications.
4. **Robust Feature Learning:** The hierarchical structure of VGG-16 enables the model to learn robust feature representations, rendering it suitable for diverse image recognition tasks, including medical image analysis (S. Serte, A. Serener, 2022; A. Ajit, K. Acharya, 2020, S. Batra, S. S. Malhi, 2019)

In essence, the VGG-16 architecture's blend of simplicity, efficiency, and robust feature learning makes it a versatile tool for various image-related tasks, offering both performance and interpretability.

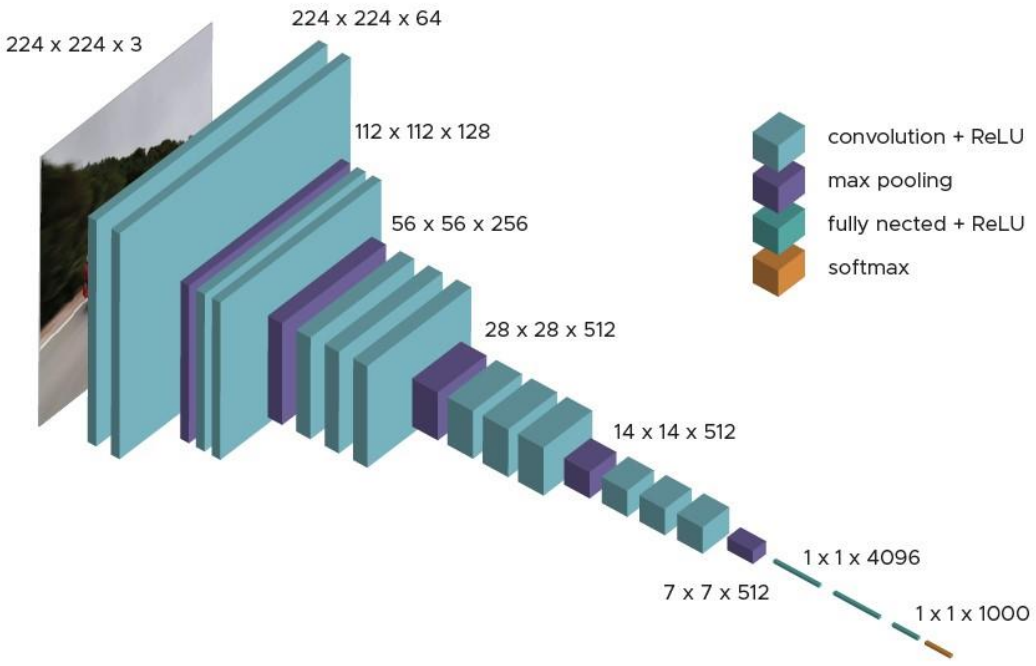

Figure 10: Importing a VGG 16 model from Keras, source: https://graiphic.io/importing-a-vgg-16-model-from-keras/

2.5 MobileNetV2 Model

MobileNetV2 is a state-of-the-art convolutional neural network architecture tailored specifically for mobile and embedded vision functions. It is an evolution of the original MobileNet architecture, introduced by Google researchers in 2017. MobileNetV2 improve upon the achievements of its forerunner by incorporating various innovative design choices aimed at improving performance and efficiency.

An important key characteristic of MobileNetV2 is its use of inverted residual blocks with linear bottlenecks. This design assists the network to achieve a good balance between model complexity and computational efficiency. Inverted residuals consist of a lightweight bottleneck layer pursued by a depth wise separable convolution layer, ultimately supports to minimize the number of parameters and computational cost while maintaining dramatic power. Additionally, linear bottlenecks ensure that information flow through the network remains efficient by reducing the impact of non-linear activations.

Another important aspect of MobileNetV2 is its use of shortcut connections, similar to those found in residual networks (ResNets). These connections help facilitate gradient flow during training, enabling deeper networks to be trained more effectively. By incorporating shortcuts, MobileNetV2 is able to achieve higher accuracy without significantly increasing computational overhead.

MobileNetV2 also introduces a novel architecture design called "efficient inverted residuals with linear bottlenecks." This design includes a new layer called the "squeeze-and-excitation" block, by adaptively recalibrating channel-wise feature responses enhances feature representation. This mechanism allows the network to prioritize on informative features while suppressing irrelevant ones, resulting in enhance performance.

The study developed a WeChat applet using the MobileNetV2 network (Liying, Le Ma, Dandan, Liping, 2023), enabling users to classify household garbage via mobile upload. Compared to a CNN model, MobileNetV2 achieved a 15.42% higher classification accuracy. The paper's novelty lies in training lightweight network models and integrating them into a user-friendly applet, prioritizing practical application over network performance improvement. This approach addresses real-world garbage classification

needs while leveraging cutting-edge deep learning algorithms within widely used platforms like WeChat.

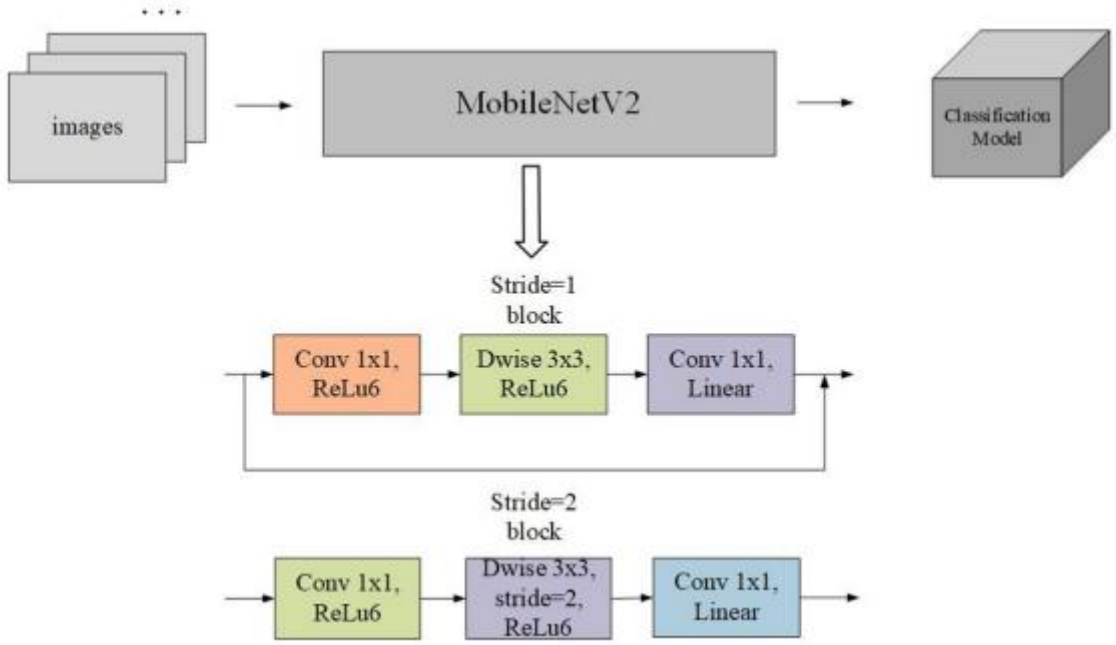

Figure: 11 MobileNetV2 network structure, image source (Liying, Le, Dandan, Liping, 2023)

3. Framework

The main goal of this research work is to conduct a comparative analysis of various augmentation techniques, both individually and in combination. In order to assess the efficacy of these augmented techniques, we employed several models including CNN, VGG, and MobileNET.

This chapter offers comprehensive insights into the experimental procedures employed in the research. Section 3.1 delves into details regarding the datasets, their pre-processing, and the image augmentation techniques applied in the project. Section 3.2 outlines the necessary software and hardware for image classification, while Section 3.3 elaborates on the architectures of the deep learning models CNN, VGG and MobileNet.

3.1 Data preparation and Preprocessing

3.1.1 Datasets

The thesis delves into an in-depth exploration of the CIFAR-10 and CIFAR-100 datasets (https://www.cs.toronto.edu/~kriz/cifar.html), which are meticulously curated subsets derived from the extensive tiny images of 80 million dataset. These subsets, crafted by the collaborative efforts of Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, serve as pivotal resources for studying image classification and deep learning methodologies. CIFAR-10 encompasses a vast array of color images contains 60,000 32x32 distributed through 10 distinct classes, with each class comprising six thousand meticulously labeled images. These images are thoughtfully divided into training and test sets, with training purposes of fifty thousand images allocated and for rigorous testing is 10,000. The layout of the dataset, including its organization into batches, is comprehensively outlined, providing researchers with valuable insights into its structure and composition. Moreover, the CIFAR-100 dataset, mirroring the architecture of CIFAR-10 but expanded to encompass 100 diverse classes, is thoroughly examined, shedding light on its unique characteristics and potential applications.

Furthermore, the thesis meticulously documents baseline results obtained from these datasets, offering valuable insights into the performance metrics achieved by various convolutional neural network (CNN) architectures. These insights serve as a foundational framework for further research endeavors, guiding researchers in the exploration of novel methodologies and techniques for image classification tasks.

Overall, the comprehensive analysis presented in this thesis provides researchers with a robust foundation for conducting in-depth investigations into image classification, deep learning model training, and the broader applications of machine learning in computer vision.

3.1.2 Data Preprocessing

Data preprocessing implies converting into a format from raw data that is both usable and efficient. Within the realm of Deep Learning, data preprocessing stands out as a critical initial step in dataset preparation. Numerous image preprocessing techniques, including normalization, grayscale conversion, noise reduction, and image augmentation, serve to enhance the image features. This experiment specifically utilizes image augmentation, which will be elaborated upon in the subsequent section.
The significance of preprocessing in deep learning cannot be overstated, as it serves to mitigate issues such as overfitting and subpar results, as highlighted by Mridha et al. (2021). Furthermore, image preprocessing plays a crucial role in optimizing training times, particularly when dealing with large datasets, by reducing image sizes.

In this research, the data preprocessing comprised the subsequent procedures, as outlined by Papandrianos et al. (2020).

1. The RGB images are organized into folders labeled as "Magnoliopsida" and "Pinopsida" based on their respective classes for CNN training. The "Magnoliopsida" folder corresponds to class label 0, while the "Pinopsida" folder corresponds to class label 1.

2. Input data normalization involves adjusting the RGB values of each image in the dataset. This process subtracts the mean RGB values across all images and divides by the standard deviation, as outlined by Zakir et al. (2021). In PyTorch, this normalization is achieved using the torchvision.transforms.Normalize() function.
3. Data partitioning: For the CNN/VGG/MobileNet image classification, splitting the dataset is into three segments: training, validation, and testing. This division, in an 80:10:10 ratio, is accomplished using the random split function, ensuring non-overlapping datasets.
4. Data Loader: When dealing with large datasets, memory constraints and slow execution can be common issues. GPU/TPU setting and High Level Server addresses this challenge by employing Data Loader, a tool that facilitates parallel data loading and automatic batching. By leveraging Data Loader, memory usage is optimized and processing speed is enhanced. This utility is readily available within the torch.utils.data package.

3.1.3 Augmentation

Augmentation involves artificially expanding the image dataset by generating additional training samples through various processing techniques such as rotation, brightness adjustment, pixel shifting, and horizontal or vertical flipping (Gu et al., 2019). When working with deep learning models, extensive datasets are typically required. However, in limited number of images scenarios, augmentation techniques become essential (Zakir et al., 2021). Studies have shown that training models with augmented images can reduce error rates and improve generalization performance. Imbalance in data distribution often leads to overfitting and poor model generalization. To mitigate these issues and bolster model robustness (Sokolova et al., 2021), diverse augmentation techniques like rotation, flipping, brightness correction, and contrast adjustments are applied to enhance the training dataset, consequently expanding its volume.

Augmentation Model

CNN (Convolutional Neural Network), MobileNet, and VGG (Visual Geometry Group) are all types of deep neural network models, particularly designed for tasks involving image recognition, classification, and related computer vision applications. Each of these models is composed of neurons in multiple layers, including convolutional layers, pooling layers, and completely connected layers, to achieve specific goals with organized in different architectures such as high accuracy, computational efficiency, or suitability for implementation on devices with limited resources such as smartphones, mobile phones or the systems of embedded.

1. CNN (Convolutional Neural Network):

- A type of deep neural network is CNN that is primarily used for analyzing visual imagery. It is widely used in image recognition and classification tasks.
- They are composed of multiple layers, including convolutional layers, pooling layers, and complete connection of layers.
- Convolutional specific layers apply filters or kernels to the specific input image, extracting the features such as textures, edges, and shapes.
- Pooling layers reduce the dimensionality of the feature maps produced considering convolutional layers, helping to make the model more computationally efficient and reducing overfitting.
- Fully connected layers utilize the high-level features obtained from preceeding layers to make predictions or classifications.

2. VGG (Visual Geometry Group):

- A convolutional neural network named VGG which architecture proposed by the Visual Geometry Group at the University of Oxford.
- The VGG network is renowned for its simplicity and uniform architecture, consisting mainly convolutional layers with 3x3 filter and stride 1 followed by max-pooling layers with 2x2 windows with two stride.
- VGG networks come in several variants, including VGG19 and VGG16, which

differ in the number of layers.

- Despite its simplicity, VGG has been widely used and achieved competitive performance on various image recognition benchmarks.

3. MobileNetV2:

- An efficient convolutional neural network architecture named MobilenetV2 tailored for use on embedded devices and smartphones with restricted computational resources.
- MobileNet uses convolutions that are depthwise separable, which factorize standard convolutions into pointwise convolutions and depthwise convolutions.
- A single convolutional filter applied by depthwise convolutions per input channel, separating spatial and channel-wise operations reducing computation.
- Pointwise convolutions then apply 1x1 convolutions to combine the output channels of depthwise convolutions, enabling the network to learn complex features efficiently.
- MobileNet models are characterized by their small size, low latency, and high efficiency, making them suitable for applications where computational resources are limited, such as mobile recognition of image and detection of real-time object.

Augmentation techniques

An augmentation technique employed in the experiments are presented in Figure 12 below:

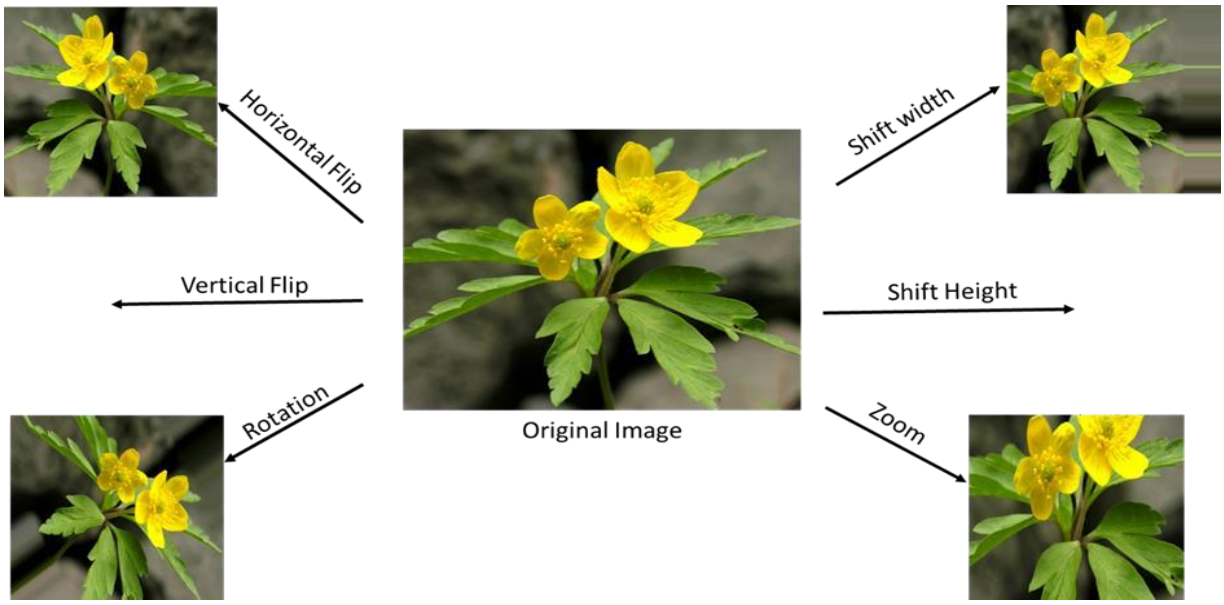

Figure 12. The augmentation techniques employed in the study.

The augmentation techniques utilized in this study are

1. **Horizontal Flip**: In horizontal flip augmentation, an image is flipped horizontally along its vertical axis. This means that the pixels on the image left side are swapped with the pixels on the right side. For example, if there's an image of a cat facing towards the left, after applying horizontal flip augmentation, the cat would appear to be facing towards the right.
2. **Vertical Flip:** In vertical flip augmentation, an image is flipped vertically along its horizontal axis. This means that the pixels on the top of the image are swapped with the pixels on the bottom. For instance, if there's an image of a person standing upright, after applying vertical flip augmentation, the person would appear upside down.
3. **Zoom Range**: This parameter controls the range by which an image can be zoomed in or out during augmentation. It is typically specified as a range or tuple depicting the smallest and largest zoom levels. For example, a zoom range of (0.8, 1.2) means that the image can be zoomed in by up to 20% or zoomed out by up to 20%. When

applying zoom augmentation, the image is magnified or shrunk along its width and height dimensions, which can simulate the effect of objects appearing closer or further away in the image. This augmentation technique helps the training dataset for the increment of diversity and improvement for the ability of model to generalize to invisible data.

4. **Width Shift Range**: This parameter controls the range by which an image can be horizontally shifted during augmentation. It is typically specified as a range or tuple representing the minimum and maximum shift distances as the total width fraction of the image. For instance, a width shift range of (-0.1, 0.1) means that it is possible to shift the image horizontally by up to 10% of its total width to the left or right. When applying width shift augmentation, the pixels of the image are shifted horizontally, which can simulate changes in the objects position within the image. This augmentation technique helps to introduce variability in the spatial location of objects and improve the ability of model to learn robust features regardless of their image position.
5. **Channel Shifting:** This technique randomly changes the RGB channel values of the image. The channel_shift_range parameter is used to specify the channel values. The following Figure 13 shows the RGB channel shifting of an image.

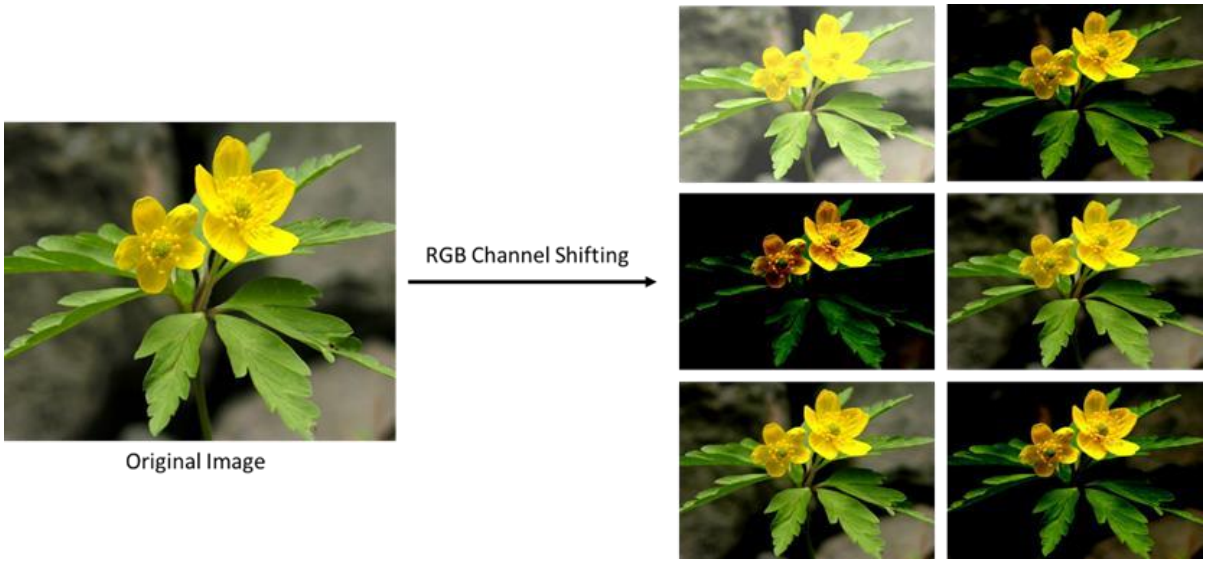

Figure 13. The channel shifting augmentation technique.

The data size of the class increased from 5 to 775 images after the application of the image augmentation techniques to the Pinopsida plant class.

3.2 Setup Specifications

3.2.1 Hardware

Deep learning (Potluri et al. 2012) relies heavily on computational resources to perform its tasks efficiently. During the training phase, the neural network undertakes intensive tasks, passing inputs through its layers, including hidden layers. These inputs are processed within the hidden layers using weights that are continually adapted during training, ultimately contributing to the model's predictions. The refinement of these weights allows the model for identification of patterns and make more perfect predictions. Notably, matrix multiplications are frequently employed in both training and prediction operations.

Deep learning demands and is dependent upon a great amount of computational power (Thompson et al. 2020). GPU parallelization, particularly for tasks like convolution and backpropagation, is pivotal for efficient processing. Consequently, the display memory of GPU devices is indispensable for facilitating machine learning tasks. Notably, the conducted experiments on a laptop equipped with an Intel(R) Core (TM) i7-1065G7 CPU, 16 GB RAM, and Intel(R) Iris(R) Plus Graphics with 4 GB of display memory.

3.2.2 Software

The experiments were conducted utilizing Python 3.10.12on Google colab. PyTorch, TensorFlow, and Keras were the primary libraries employed. PyTorch, developed by the Facebook AI Research team, serves as an optimized tensor for Deep Learning tasks, offering an open-source machine learning framework for Python. Similarly, Keras, another open-source library, furnishes an interface of Python for constructing artificial neural networks and serves for the TensorFlow library as an interface.

Matplotlib is a comprehensive library for the static creation, interactive, and animated visualizations in Python. It presents a wide range of functionalities for generating plots, charts, histograms, and diverse types of visualizations that are achieving it a robust tool for data exploration and presentation.

Numerical Python, short form is NumPy, is a fundamental package for scientific computing in Python. Along with a collection of mathematical functions to operate on these arrays efficiently it provides support for large, multi-dimensional arrays and matrices,.

ImageDataGenerator is a versatile device for efficiently preprocessing and augmenting image data during deep learning model training, contributing to improved model performance and generalization. The process is simplified for the data augmentation and integration with deep learning pipelines, managing it an essential component of image-based machine learning workflows in Keras.

3.3 Architectures Model and Implementation

Following section explains the architectures of CNN, VGG and MobileNet models and the implementation of these models to classify the plant classes for the study

3.3.1 CNN Architecture

The architecture of CNN comprises three convolutional layers, three max-pooling layers, one flattening layer, and two completely connected layers.

The initial input is an image of dimensions 224 x 224 with 3 RGB channels. It undergoes processing through the first 2D convolutional layer, featuring a 3x3 kernel size, a stride of 1x1, padding of 1x1, 32 filters, and a ReLU activation function. This layer yields an output shape of 224 x 224 x 64. Subsequently, max-pooling with a dimension of 2 x 2 is applied, reducing the image dimensions to 112 x 112 x 64.

Passing the image through additional sets of 2D convolutional layers and max-pooling layers twice this sequence repeats, until the image size reaches 28x28x256. The flattening operation transforms into a single-dimensional tensor from the multi-dimensional tensor, preparing it for input into the fully connected layers. For classifying the images these layers are responsible into two categories: Pinopsida and Magnoliopsida.

The CNN architecture described above is represented in Figure 14.

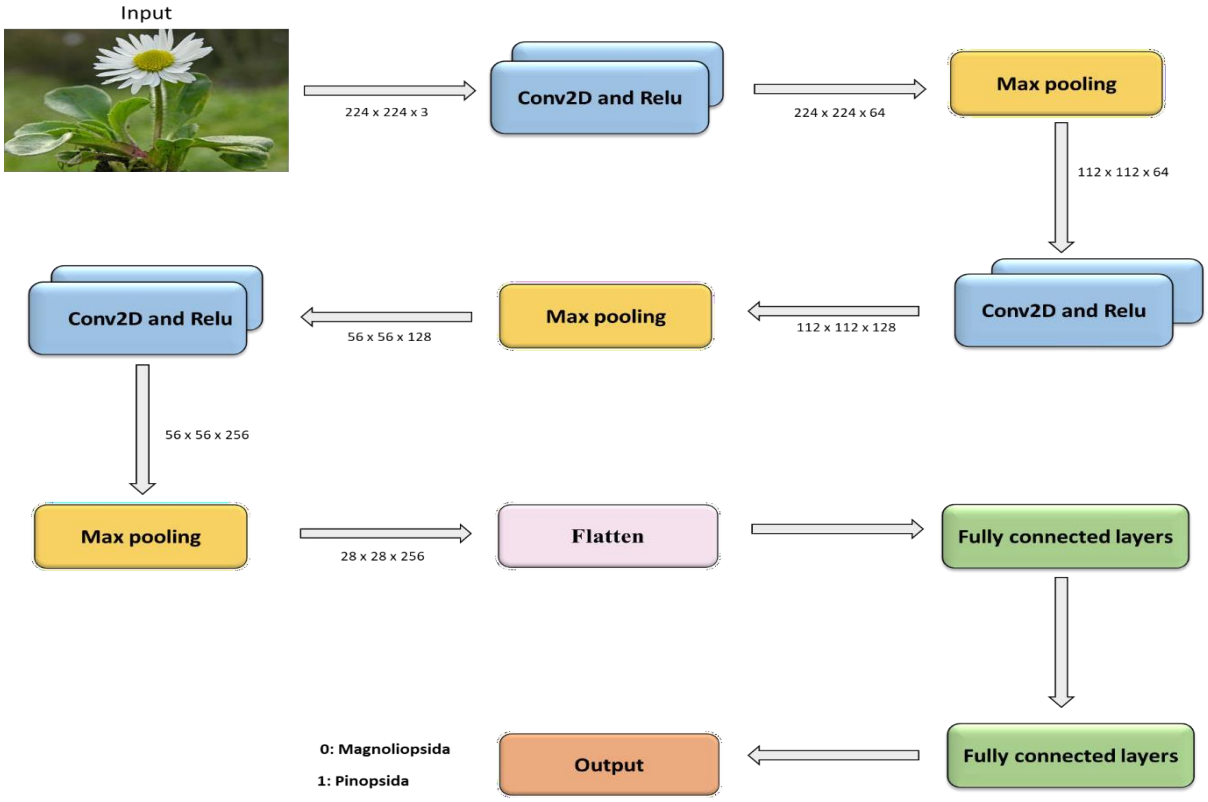

Figure 14. The CNN architecture.

3.3.1 VGG Architecture

The VGG (Visual Geometry Group) architecture is a deep convolutional neural network (CNN) architecture proposed by the Visual Geometry Group at the University of Oxford. For its simplicity and effectiveness in image classification tasks it gained prominence.

Key features of the VGG architecture include:

1. **Depth:** VGG is characterized by its deep architecture, with up to 19 layers (VGG-19) in the original configuration. Followed by max-pooling layers to reduce spatial dimensions the basic building blocks are repeated convolutional layers with small 3x3 filters,
2. **Convolutional Layers:** VGG consists primarily of convolutional layers, where each layer performs convolutions on the input image to extract features. The use of small 3x3 filters with a stride of 1 pixel allows for better feature extraction and more parameters compared to larger filters.

3. **Pooling Layers:** After a few convolutional layers, max-pooling layers are used to reduce the size of the feature maps, reducing their spatial dimensions while retaining important features. Max-pooling is typically applied with 2x2 filters and a stride of 2 pixels.
4. **Completely Connected Layers:** Towards the network ending, fully connected layers are utilized for high-level reasoning and classification using the extracted features. These layers aggregate information from the feature maps and output the final class predictions.
5. **Activation Function**: Rectified Linear Unit (ReLU) activation functions are commonly applied followed by each convolutional and fully connected layer for the introduction of non-linearity into the network and allow it to learn complex relationships in the data.
6. **Architecture Variants:** VGG architectures come in different variants, such as VGG-16 and VGG-19, which differ layers in amount of numbers. VGG-16 has 16 layers (thirteen convolutional and three fully connected), while VGG-19 has 19 layers (sixteen convolutional and three fully connected).

The VGG architecture which is known For its simplicity, uniformity, and effectiveness in feature extraction and classification tasks of image. Despite being deeper than previous architectures like AlexNet, VGG achieved competitive performance on benchmark datasets such as ImageNet.

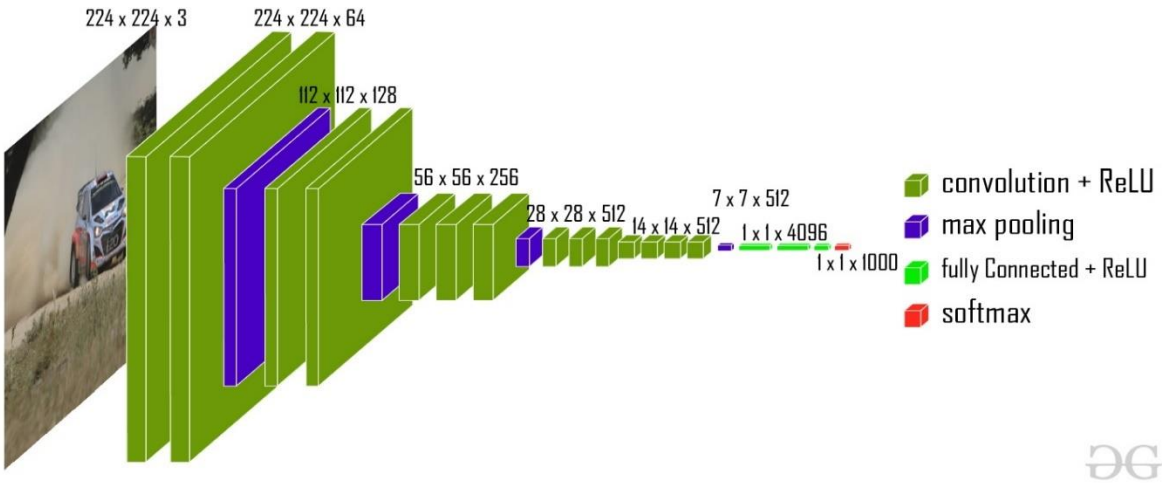

Figure 15. The VGG architecture.

3.3.1 MobileNetV2 Architecture

A lightweight convolutional neural network (CNN) architecture, MobileNetV2 is designed for efficient and fast deployment on mobile and embedded devices with limited computational resources. Google researchers Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam developed it.

Key features of the MobileNet architecture include:

1. **Depthwise Separable Convolution:** The fundamental component of MobileNet is the depthwise separable convolution operation, which factorizes standard convolutions into two different operations: depthwise convolution and pointwise convolution. Depthwise convolution applies a single convolutional filter to each input channel individually, followed by pointwise convolution, which combines the output of the depthwise convolution across channels using 1x1 convolutions. This approach substantially reduces the number of parameters and computational cost compared to traditional convolutional layers.
2. **Depthwise Convolution:** Depthwise convolution operates independently on each input channel, resulting in a set of intermediate feature maps for each channel. This operation captures spatial information within each channel while keeping computational costs low.
3. **Pointwise Convolution:** Pointwise convolution combines the intermediate feature maps from depthwise convolution across channels using 1x1 convolutions. This operation enables the network to learn complex feature combinations and interactions across channels.
4. **Depthwise Separable Convolution Block:** MobileNet uses a series of depthwise separable convolution blocks stacked sequentially to form the network architecture. Each block comprises a depthwise convolution layer succede by a pointwise convolution layer, optionally followed by batch normalization and ReLU activation.
5. **Width Multiplier and Resolution Multiplier:** MobileNet presents hyperparameters known as width multiplier and resolution multiplier to control the computational cost and model size. The width multiplier scales the number of channels in each layer, while

the resolution multiplier scales the input image resolution. These hyperparameters allow for trade-offs between model size, accuracy, and inference speed, making MobileNet adaptable to different deployment scenarios.

6. **Global Average Pooling and Softmax:** MobileNet typically ends with a global average pooling layer followed by a softmax layer for classification tasks. Global average pooling aggregates spatial information across feature maps and produces a single feature vector, which is then fed into a softmax layer to compute class probabilities.

MobileNet architecture is optimized for efficiency and performance, making it suitable for the classification of real-time image, detection of object, and other computer vision tasks on devices with limited resources such as smartphones, drones, and devices of IoT.

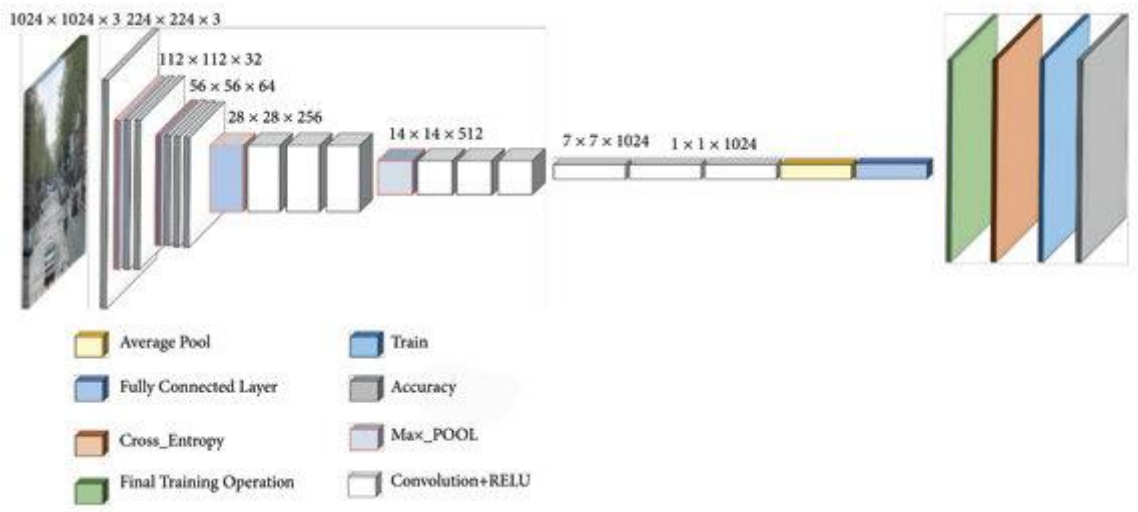

Figure 16. The MobileNetV2 architecture.

3.3.3 Implementation of Models

The process begins with an input images that is subjected to the pre-processing of data. The dataset is subsequently divided into training, validation, and testing subsets. This entails constructing the models, which are then trained using the training dataset. The performance of model is assessed using the validation dataset. Later, the test dataset is utilized for image classification predictions, ultimately culminating in the final output. Figure 15 below illustrates a summary of the model implementation process.

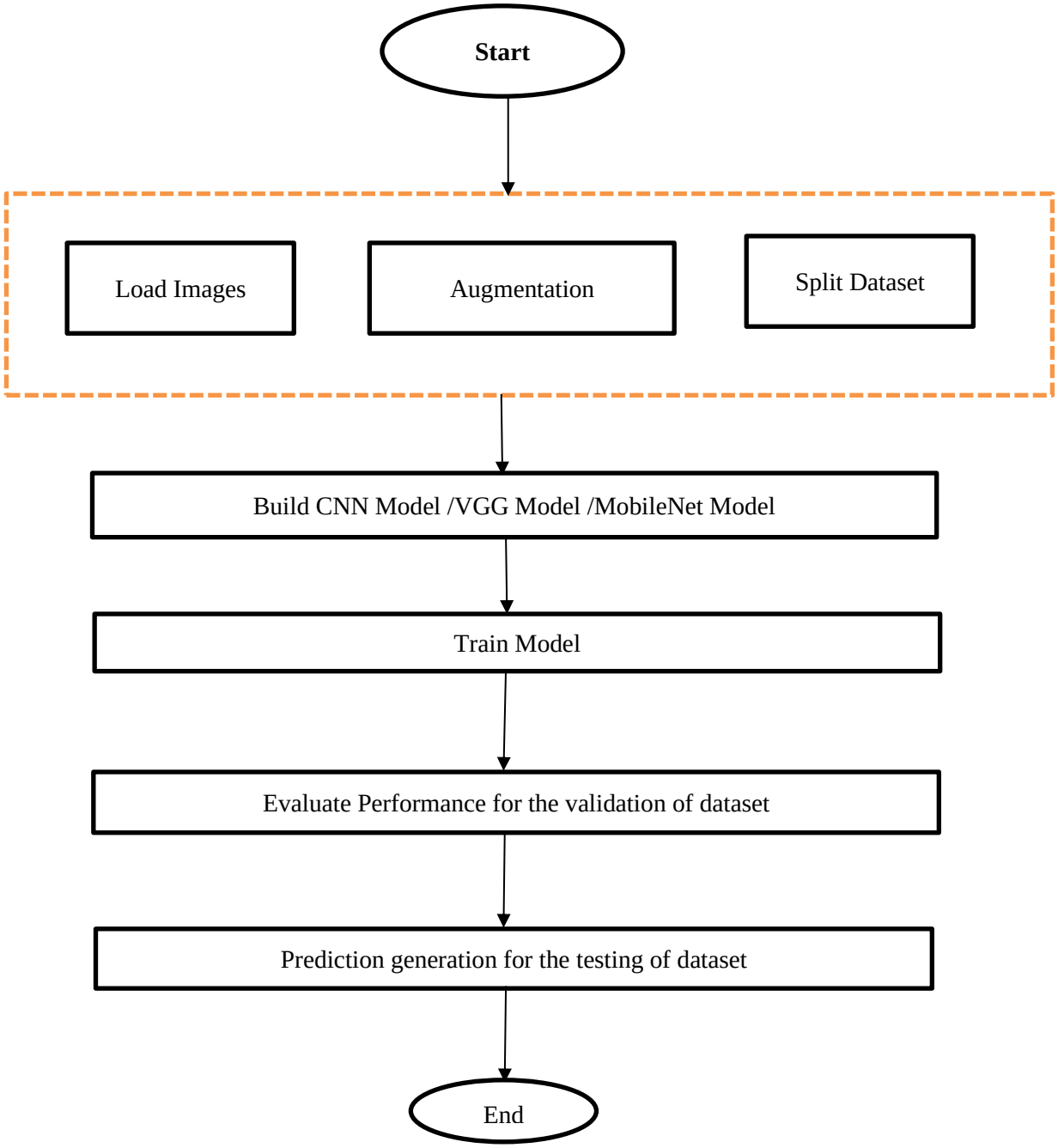

Figure 17. The summary of the implementation of models

4. OUTCOME AND DISCUSSIOINS

This chapter presents the outcomes derived from image classification and augmentation utilizing Deep Learning models. Below are the detail findings of the CNN model, VGG model and MobileNet models across distinct image datasets for the original image dataset and augmented image dataset with varying different parameters.
Various combinations of augmentation techniques have been applied, including single augmentation, double augmentation, and triple augmentation.

4.1 Outcome of CNN Model with Different Combination Augmentation

Optimizing Model Performance through Gradual Epoch Incrementation: A Study on CNN with Single Data Augmentation Method

Upon experimenting with varying numbers of epochs, a notable trend emerges. Initially, with fewer epochs, the model's performance appears suboptimal, showcasing poorer results. However, as the epochs number increases gradually, reaching the range of 40 to 45 epochs, a significant improvement is observed in the model's outcomes.
This progression suggests that the model requires an extended training period to adequately learn the underlying patterns within the data. By incrementally increasing the epoch count, the model can refine its representations and achieve superior performance metrics.
It's evident that a careful balance must be struck to prevent overfitting, as excessively prolonged training may lead to diminishing returns or even degradation in performance on unseen data. Nonetheless, the observed enhancement in results with an extended epoch range underscores the importance of sufficient training iterations for optimizing model performance.

4.1.1 Single Data Augmentation Method

The training process spans 40 epochs, during which the model continually refines its performance. Notably, the loss of training steadily diminishes from 0.7177 to 0.1392, while the accuracy of training steadily increases from 0.7493 to 0.9517. These trends suggest that the model effectively learns from the training data over successive epochs.

Likewise, the loss of validation falls from 0.8682 to 2.1932, and the accuracy of validation increases from 0.7070 to 0.6911. However, there seems to be a point of diminishing returns, as the validation metrics start to degrade after around epoch 25, despite improvements in the training metrics. This indicates that the model may start to overfit the training data as training progresses, leading to reduced generalization of the validation set performance.

Overall, the CNN model with single data augmentation demonstrates the capability learning from the training data and achieve reasonably high accuracy on the validation set. However, further investigation into mitigating overfitting beyond epoch 25 may be necessary to improve generalization performance.

Table

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
7	0.7177	0.7493	0.8682	0.7070
8	0.6725	0.7637	0.8892	0.6990
9	0.6299	0.7791	0.8660	0.7117
10	0.5991	0.7901	0.8782	0.7102
11	0.5624	0.8012	0.8604	0.7171
12	0.5276	0.8130	0.8658	0.7183
13	0.4991	0.8247	0.9323	0.7147
14	0.4660	0.8363	0.9146	0.7122
15	0.4444	0.8414	0.9964	0.7054
16	0.4143	0.8522	1.0218	0.7067
17	0.3932	0.8603	1.0263	0.7104
18	0.3658	0.8696	1.0708	0.7046
19	0.3472	0.8746	1.0874	0.7141
20	0.3220	0.8855	1.1498	0.7108
21	0.3021	0.8924	1.1703	0.7079
22	0.2864	0.8964	1.2339	0.7130
23	0.2721	0.9021	1.3217	0.7034
24	0.2564	0.9076	1.3485	0.7033

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
25	0.2485	0.9103	1.4355	0.6963
26	0.2291	0.9160	1.4381	0.7000
27	0.2149	0.9236	1.5113	0.7055
28	0.2053	0.9261	1.6045	0.6933
29	0.2039	0.9267	1.6566	0.7000
30	0.1956	0.9312	1.7448	0.6913
31	0.1814	0.9347	1.8570	0.6822
32	0.1664	0.9398	1.8167	0.6854
33	0.1688	0.9396	1.8768	0.6928
34	0.1630	0.9414	1.8643	0.6957
35	0.1590	0.9432	2.0126	0.6990
36	0.1542	0.9462	1.9535	0.6942
37	0.1551	0.9440	2.0561	0.6852
38	0.1427	0.9492	2.0763	0.6942
39	0.1436	0.9495	2.1502	0.6867
40	0.1392	0.9517	2.1932	0.6911

Table: 1 Single Data Augmentation Method CNN Model

Performance Analysis of CNN Model with Additional Convolutional Layers and Dropout Regularization for Image Classification

This model architecture aims to increase the efficiency and robustness of the convolutional neural network (CNN) through the introduction of additional convolutional layers and the incorporation of a dropout layer for regularization. The key changes in this architecture are outlined below:

1. Additional Convolutional Layers: The model incorporates two sets of convolutional layers, each comprising two convolutional layers succeeded by a max-pooling layer. This design choice enables the model for capturing more complex features from the input data, potentially enhancing its ability to discriminate between different classes.
2. Dropout Layer: A dropout rate of 0.5 in dropout layer is inserted before the final dense layer. A regularization technique, dropout that helps mitigate overfitting by randomly disabling a fraction of input units during the process of training, thereby influencing the network to become more resilient and adaptable representations.

Experimentation with this architecture and adjustment of hyperparameters offer avenues for exploring its impact on model performance across diverse datasets.

Table:

Epoch	Time/Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	66s	1.5505	0.4368	1.2843	0.5413
2	62s	1.1862	0.5788	1.1034	0.6125
3	63s	1.0403	0.6330	1.0827	0.6185
4	62s	0.9449	0.6690	0.9735	0.6628
5	63s	0.8713	0.6937	0.9070	0.6840
6	64s	0.8193	0.7125	0.8959	0.6859
7	60s	0.7662	0.7316	0.8758	0.6974
8	63s	0.7256	0.7468	0.8568	0.7088
9	63s	0.6842	0.7593	0.9245	0.6840
10	62s	0.6536	0.7706	0.9085	0.6898
11	64s	0.6169	0.7836	0.8716	0.7098
12	64s	0.5922	0.7914	0.9075	0.7077
13	61s	0.5609	0.8032	0.8912	0.7103
14	63s	0.5353	0.8111	0.9357	0.7089
15	62s	0.5122	0.8184	0.9190	0.7101
16	61s	0.4844	0.8276	0.9451	0.7105
17	62s	0.4604	0.8372	0.9819	0.7068
18	60s	0.4367	0.8459	1.0162	0.7062
19	62s	0.4140	0.8523	1.0491	0.7071
20	63s	0.3977	0.8585	1.0740	0.7065
21	62s	0.3761	0.8668	1.1069	0.7015
22	62s	0.3562	0.8727	1.1847	0.6935
23	68s	0.3407	0.8785	1.1979	0.6972
24	62s	0.3169	0.8857	1.3513	0.6792
25	62s	0.3031	0.8916	1.2966	0.6954
26	62s	0.2893	0.8960	1.3419	0.6914
27	62s	0.2747	0.9014	1.4347	0.6934
28	60s	0.2591	0.9081	1.4245	0.6890
29	61s	0.2530	0.9082	1.4679	0.6907
30	61s	0.2384	0.9140	1.5357	0.6922
31	61s	0.2261	0.9169	1.5546	0.6850
32	61s	0.2194	0.9211	1.6999	0.6775

Table: 2 Single Data Augmentation Method with Changing Hyperparameters CNN Model

The introduction of additional convolutional layers and the incorporation of a dropout layer in this model architecture demonstrate a gradual improvement in performance over epochs. Initially, the model exhibits relatively lower accuracy and higher loss, which is typical during the early stages of training. However, as training progresses, the accuracy steadily increases, while gradually decreases the loss, demonstrating that the model is learning to better classify the data.

The accuracy of training consistently outperforms the accuracy of validation, suggesting that the model may be slightly training data overfitting. This phenomenon is further corroborated by the observation of increasing validation loss over epochs, indicating that the generalization ability of model may be compromised.

Nevertheless, the model reaches a peak accuracy of validation of approximately 71%, indicating that it can effectively classify the validation data. However, further experimentation with regularization techniques or model architecture adjustments may be necessary to mitigate overfitting and enhance generalization performance.

While the model demonstrates promising performance, there is room for optimization to achieve better generalization and robustness across diverse datasets.

4.1.2 Enhancing Model Robustness with Double Data Augmentation: A Comprehensive Investigation

The model trained with two augmentation methods, horizontal flipping and rotation range of 45 degrees, shows decent performance during training, achieving an accuracy of around 95% on the training set. However, the validation accuracy hovers around 68-71%, indicating some degree of overfitting which occurs when the model learns to memorize the training data instead of generalizing well to invisible data.

Despite the overfitting, the model maintains a relatively stable training accuracy throughout the epochs, suggesting that the model is continuously learning from the training data effectively. However, the accuracy of validation plateaus early on and does not substantially improve even with further training, indicating that the model struggles to generalize to data which is invisible.

Further regularization techniques, such as early stopping or additional dropout layers, could potentially help overfitting mitigation and improve the performance of model generalization. Additionally, fine-tuning the augmentation parameters or exploring different augmentation techniques might also yield improvements in model performance. Refer to the table below.

Table:

Epoch	Loss	Accuracy	Validation Loss	Validation Accuracy
1	1.4977	0.4536	1.2789	0.5412
2	1.1408	0.5955	1.0282	0.6363
3	0.9886	0.6532	0.9676	0.6611
4	0.8918	0.6873	0.9370	0.6697
5	0.8165	0.7147	0.8645	0.7013
6	0.7547	0.7351	0.8528	0.7038
7	0.7025	0.7526	0.8884	0.6995
8	0.6579	0.7688	0.9290	0.6886
9	0.6169	0.7845	0.8687	0.7104
10	0.5784	0.7950	0.8753	0.7130
11	0.5434	0.8073	0.8855	0.7172
12	0.5052	0.8220	0.9076	0.7146
13	0.4718	0.8329	0.9250	0.7187
14	0.4388	0.8459	0.9514	0.7157

Epoch	Loss	Accuracy	Validation Loss	Validation Accuracy
15	0.4143	0.8520	1.0996	0.6953
16	0.3855	0.8622	1.0659	0.7032
17	0.3619	0.8712	1.0873	0.7075
18	0.3288	0.8829	1.1478	0.7037
19	0.3128	0.8883	1.1534	0.7022
20	0.2971	0.8946	1.2130	0.6978
21	0.2762	0.9015	1.2684	0.7032
22	0.2604	0.9060	1.3405	0.6964
23	0.2458	0.9107	1.3990	0.7004
24	0.2305	0.9163	1.4457	0.6999
25	0.2273	0.9177	1.5291	0.6865
26	0.2099	0.9247	1.5791	0.6913
27	0.1964	0.9298	1.6212	0.6896
28	0.1962	0.9285	1.7715	0.6880
29	0.1803	0.9360	1.8138	0.6880
30	0.1747	0.9365	1.8347	0.6861
31	0.1711	0.9385	1.8862	0.6871
32	0.1644	0.9410	1.8753	0.6864
33	0.1568	0.9438	1.9177	0.6855
34	0.1506	0.9469	2.0464	0.6805
35	0.1484	0.9476	2.1215	0.6810
36	0.1442	0.9485	2.0696	0.6817
37	0.1457	0.9489	2.2542	0.6846
38	0.1398	0.9507	2.2578	0.6857
39	0.1408	0.9492	2.2615	0.6822
40	0.1382	0.9506	2.2147	0.6889

Table: 3 Double Data Augmentation Method CNN Model

4.1.3 Comparing One augmentation method and two augmentation method

Combining Both Augmentation Methods:

- Validation Accuracy after Epoch 1: 0.5412
- Validation Accuracy after Epoch 2: 0.6363

One Augmentation Method:

- Validation Accuracy after Epoch 1: 0.5616
- Validation Accuracy after Epoch 2: 0.6341

4.2 Results of VGG16 model with Single Combination Augmentation

Observations for the VGG16 model with ImageDataGenerator(horizontal_flip=True, vertical_flip=True):

1. **Training Accuracy and Loss**: The training accuracy starts at 52.80% and increases steadily over the epochs, reaching 84.69% by the 30th epoch. Similarly, the loss of training decreases from 1.3455 to 0.4392 during the epochs, representing that the perfect model is learning successfully from the accurate training information.
2. **Validation Accurateness and Loss**: The validation accurateness starts at 55.27% and fluctuates around 60-62% throughout the training process, showing some signs of overfitting as the progression of training. The validation loss follows a similar inclination, initially decreasing but then increasing gradually, indicating that the model's performance on unseen data is not improving significantly.
3. **Epoch Time**: Each epoch takes a considerable amount of time, ranging from around 620 to 689 seconds. This suggests that training the VGG16 model with the given dataset and augmentation settings is computationally intensive.
4. **Performance with Data Augmentation**: Although data augmentation (horizontal and vertical flips) is used, it does not seem to have a positive substantial impact on the performance of the model in terms of validation accuracy. The model might benefit from further experimentation with different augmentation techniques or tuning of hyperparameters.
5. **Potential Overfitting**: The increasing gap between the training and the accuracy of validation after several epochs indicates potential overfitting, where the model starts to memorize the training of data rather than generalizing well to invisible of data.

Finally, while the model achieves decent training accuracy, there is room for improvement in terms of validation accuracy and addressing overfitting. Further optimization and experimentation with hyperparameters and augmentation techniques could help improve the model's performance.

Table

Epoch	Duration per Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	630s	1.3455	52.80%	1.2646	55.27%
2	666s	1.1663	59.21%	1.1730	58.59%
3	629s	1.1014	61.44%	1.1633	59.03%
4	666s	1.0522	63.15%	1.1341	60.29%
5	626s	1.0083	64.79%	1.1434	59.83%
6	668s	0.9714	65.79%	1.1262	60.58%
7	667s	0.9358	67.09%	1.1064	61.94%
8	655s	0.9013	68.10%	1.1119	61.28%
9	664s	0.8713	69.36%	1.1194	61.71%
10	620s	0.8430	70.44%	1.1159	61.56%
11	661s	0.8131	71.29%	1.1397	61.26%
12	663s	0.7860	72.29%	1.1437	61.65%
13	621s	0.7608	73.32%	1.1567	61.38%
14	621s	0.7360	74.10%	1.1539	61.78%
15	662s	0.7099	74.85%	1.1725	62.43%
16	666s	0.6874	75.90%	1.1812	62.21%
17	689s	0.6683	76.54%	1.2108	61.37%
18	663s	0.6445	77.28%	1.2338	61.06%
19	666s	0.6240	78.24%	1.2618	61.30%
20	672s	0.6073	78.76%	1.2888	60.93%
21	624s	0.5846	79.51%	1.2790	61.16%
22	664s	0.5672	80.02%	1.3111	61.10%
23	662s	0.5494	80.83%	1.3828	60.33%
24	664s	0.5326	81.18%	1.3844	60.19%
25	664s	0.5154	81.94%	1.3742	60.73%
26	662s	0.5010	82.45%	1.3979	60.81%
27	661s	0.4840	83.12%	1.4437	60.54%
28	661s	0.4689	83.54%	1.4487	61.13%
29	660s	0.4562	83.98%	1.4885	60.06%

Epoch	Duration per Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
30	671s	0.4392	84.69%	1.5591	60.05%

Table: 4 Single Data Augmentation Method VGG16 Model

4.2.1 Optimizing Training Efficiency and Augmentation Effectiveness: A Journey through Epochs

The training process exhibits prolonged execution times per epoch, likely attributable to the augmentation technique employed. Initially, a modest attempt with 5 epochs was made, but the augmentation's efficacy fell short of expectations. Subsequently, an extensive training session spanning 35 epochs was initiated, demanding nearly 8 hours of computational resources. Regrettably, the process stalled at the 33rd epoch. In response to this setback, a more streamlined approach was adopted, limiting the training regimen to 30 epochs, yet ensuring a thorough evaluation of the model's performance.

4.3 Results of MobileNetV2 model with Different Combination Augmentation

MobileNetV2 represents a significant advancement in convolutional neural network architectures for mobile and embedded devices. Its efficient design, combined with powerful feature representation capabilities, including image classification, object detection, and semantic segmentation, especially in resource-constrained environments where computational resources are limited makes it well-suited for a wide range of computer vision tasks.

4.3.1 Results of MobileNetV2 model with Single Combination Augmentation

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
8	1.8754	0.3234	1.8942	0.3168
9	1.8723	0.3236	1.8943	0.3163
10	1.8702	0.3250	1.8939	0.3162
11	1.8687	0.3262	1.8932	0.3170
12	1.8674	0.3264	1.8940	0.3170
13	1.8662	0.3260	1.8938	0.3157

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
14	1.8652	0.3272	1.8935	0.3169
15	1.8641	0.3278	1.8942	0.3174
16	1.8636	0.3273	1.8936	0.3200
17	1.8628	0.3283	1.8944	0.3155
18	1.8620	0.3281	1.8949	0.3161
19	1.8614	0.3283	1.8951	0.3196
20	1.8607	0.3291	1.8959	0.3140
21	1.8603	0.3279	1.8955	0.3190
22	1.8599	0.3287	1.8958	0.3187
23	1.8594	0.3286	1.8963	0.3182
24	1.8589	0.3291	1.8966	0.3176
25	1.8585	0.3292	1.8966	0.3206
26	1.8580	0.3293	1.8975	0.3190
27	1.8578	0.3288	1.8972	0.3199
28	1.8575	0.3306	1.8979	0.3182
29	1.8571	0.3296	1.8980	0.3202
30	1.8567	0.3298	1.8984	0.3200
31	1.8566	0.3298	1.8988	0.3199
32	1.8563	0.3303	1.8997	0.3202
33	1.8561	0.3303	1.8995	0.3188
34	1.8557	0.3294	1.9003	0.3208
35	1.8555	0.3303	1.9002	0.3192

Table: 5 Single Data Augmentation Method MobileNetV2 Model

Analysis

The training and validation performance of the model over the 35 epochs show a gradual improvement in accuracy and a corresponding decrease in loss.

- **Training Loss and Accuracy**: The training loss consistently decreases from 1.8754 to 1.8553, indicating that the model is learning and fitting the training data better as the epochs progress. The accuracy of training shows a steady increase from 0.3234 to 0.3303, reflecting improved performance.
- **Validation Loss and Accuracy**: The validation loss shows minor fluctuations but overall remains stable, starting at 1.8942 and ending at 1.9007. This suggests that while the model is continuously learning, its ability to generalize to invisible data does not

show significant improvement. The validation accuracy similarly fluctuates but does not show a marked increase, beginning at 0.3168 and ending at 0.3192.

Observations

- **Plateau in Performance**: Both the training and validation metrics suggest that the model's performance is plateauing. Despite the decrease in training loss, the validation loss and accuracy do not improve significantly after a certain point, which might indicate overfitting.
- **Small Improvements**: There are slight improvements in training accuracy, but these are not mirrored in the validation accuracy, suggesting that the model might be learning noise in the training data rather than useful patterns that generalize well.
- **Need for Regularization**: The small fluctuations and lack of significant improvement in validation performance suggest that regularization techniques such as dropout, weight decay, or data augmentation might be necessary to enhance the model's generalizability.

Recommendations

- **Hyperparameter Tuning**: Experimenting with different learning rates, batch sizes, and optimization algorithms could help in achieving the performance in better way.
- **Regularization**: Implementing regularization techniques might help in reducing overfitting and improving validation accuracy.
- **Early Stopping**: Using early stopping based on the performance of validation can prevent the model from overfitting and helps to save training time.
- **More Data**: If possible, acquiring more data or using data augmentation techniques could help the model learn more robust features and improve generalization.

In conclusion, while the model shows improvements in training performance, the validation metrics indicate a need for strategies to enhance generalizability and prevent overfitting.

4.3.2 Results of MobileNetV2 model with Double Combination Augmentation

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	1.4977	0.4536	1.2789	0.5412
2	1.1408	0.5955	1.0282	0.6363
3	0.9886	0.6532	0.9676	0.6611
4	0.8918	0.6873	0.9370	0.6697
5	0.8165	0.7147	0.8645	0.7013
6	0.7547	0.7351	0.8528	0.7038
7	0.7025	0.7526	0.8884	0.6995
8	0.6579	0.7688	0.9290	0.6886
9	0.6169	0.7845	0.8687	0.7104
10	0.5784	0.7950	0.8753	0.7130
11	0.5434	0.8073	0.8855	0.7172
12	0.5052	0.8220	0.9076	0.7146
13	0.4718	0.8329	0.9250	0.7187
14	0.4388	0.8459	0.9514	0.7157
15	0.4143	0.8520	1.0996	0.6953
16	0.3855	0.8622	1.0659	0.7032
17	0.3619	0.8712	1.0873	0.7075
18	0.3288	0.8829	1.1478	0.7037
19	0.3128	0.8883	1.1534	0.7022
20	0.2971	0.8946	1.2130	0.6978
21	0.2762	0.9015	1.2684	0.7032
22	0.2604	0.9060	1.3405	0.6964
23	0.2458	0.9107	1.3990	0.7004
24	0.2305	0.9163	1.4457	0.6999
25	0.2273	0.9177	1.5291	0.6865
26	0.2099	0.9247	1.5791	0.6913
27	0.1964	0.9298	1.6212	0.6896
28	0.1962	0.9285	1.7715	0.6880
29	0.1803	0.9360	1.8138	0.6880
30	0.1747	0.9365	1.8347	0.6861
31	0.1711	0.9385	1.8862	0.6871
32	0.1644	0.9410	1.8753	0.6864
33	0.1568	0.9438	1.9177	0.6855
34	0.1506	0.9469	2.0464	0.6805
35	0.1484	0.9476	2.1215	0.6810

Epoch	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
36	0.1442	0.9485	2.0696	0.6817
37	0.1457	0.9489	2.2542	0.6846
38	0.1398	0.9507	2.2578	0.6857
39	0.1408	0.9492	2.2615	0.6822
40	0.1382	0.9506	2.2147	0.6889

Table: 6 Double Data Augmentation Method MobileNet Model

Analysis

Training Performance

- **Improvement Over Epochs:** The training loss consistently decreases from 1.4977 to 0.1382 over 40 epochs, indicating a strong learning trend. Similarly, the training accuracy shows a significant improvement starting from 0.4536 to 0.9506, determining that the model is becoming increasingly proficient at ranking the training data perfectly.

Validation Performance

- **Initial Improvement:** The loss of validation diminishing substantially in the initial epochs, from 1.2789 to a low of 0.8528 around epoch 6, and validation accuracy improves from 0.5412 to 0.7038 by epoch 6. This suggests that the model generalizes well to unseen data initially.
- **Plateau and Decline**: From around epoch 7 onwards, validation loss begins to fluctuate and generally increases, peaking at 2.2615 by epoch 39. Validation accuracy, after an initial rise, fluctuates around the 0.68 to 0.71 range and does not show significant improvement, ending at 0.6889 by epoch 40.

Observations

- **Overfitting:** The divergence between training and validation performance, particularly after epoch 6, suggests overfitting. The model continues to improve on training data but fails to generalize to validation data, as evidenced by the rising validation loss and stagnant validation accuracy.
- **Early Stopping Point**: The optimal stopping point appears to be around epoch 6-10, where validation accuracy is at its peak (0.7130) and validation loss is relatively low (0.8753). Training beyond this point does not yield better validation performance and leads to overfitting.

- **Regularization Need:** The increase in validation loss and stagnation in accuracy indicate the need for regularization techniques such as dropout, batch normalization, or L2 regularization to improve generalization and prevent overfitting.

Recommendations

- **Early Stopping**: To halt training, implementation of early stopping when validation performance stops improving, around epoch 10 in this case.
- **Regularization**: Introduce regularization techniques to combat overfitting. Options include dropout layers, L2 regularization, or data augmentation.
- **Iterative-Validation**: Use iterative-validation to ensure that the performance of model's is consistent across various subdivisions of the data, which can help in identifying and mitigating overfitting.
- **Hyperparameter Tuning**: Hyperparameters tuning such as learning rate, batch size, and architecture for the network might help in achieving a better balance between training and validation performance.

By addressing overfitting and optimizing the stopping point, the model can achieve better generalization and more reliable performance on unseen data.

5. CONCLUSION

This thesis investigated the performance and robustness of convolutional neural networks (CNNs) in image classification tasks through various model architectures and data augmentation techniques. The study spans 40 epochs of training, analyzing the interplay between training and validation metrics to understand model behavior and identify potential overfitting issues. Key observations include:

Initial Learning Phase**:**

- All models demonstrated effective learning during the initial epochs, as evidenced by a consistent decrease in training loss and an increase in training accuracy.
- Validation metrics showed substantial initial improvement, indicating good generalization to unseen data.

Overfitting Concerns:

- Post the initial learning phase, a divergence was observed between training and validation metrics. Training accuracy continuously improving while the accuracy of validation plateaued or declined, accompanied by increasing validation loss, signaling overfitting.
- The optimal stopping point for training was identified around epoch 10, where validation accuracy peaked.

Impact of Regularization Techniques**:**

- The incorporation of additional convolutional layers and dropout regularization showed a gradual performance improvement and helped mitigate overfitting to some extent.
- Despite these enhancements, further regularization was necessary to achieve better generalization.

Data Augmentation:

- Single and double data augmentation methods were explored, including horizontal flipping and rotation. Although these techniques contributed to improved training performance, their impact on validation accuracy was limited.
- The models with more extensive data augmentation demonstrated relatively better generalization but still faced overfitting challenges.

Model Architecture Adjustments**:**

- Comparisons between different architectures, including VGG16 and MobileNetV2, highlighted that while advanced architectures could capture intricate features and improve training accuracy, they also necessitated careful regularization to prevent overfitting.

Recommendations

Based on the analysis, several recommendations are proposed to enhance the generalization and robustness of CNN models in image classification tasks:

1. **Early Stopping**:

- Implementation of early stopping mechanisms to halt training when validation performance ceases to improve, effectively preventing overfitting.

2. Enhanced Regularization**:**

- Introduce more robust regularization techniques, such as higher dropout rates and L2 regularization, to improve the ability of model to generalize.

3. Diverse Data Augmentation**:**

- Utilize a broader range of augmentation of data methods to generate additional varied samples, which can help the model learn more features in generalize way.

4. **Hyperparameter Tuning**:

- Conduct extensive hyperparameter tuning, including adjustments to learning rates, batch sizes, and optimization algorithms, to find the optimal configuration for balanced performance.

5. Cross-Validation**:**

- Employ cross-validation ensuring consistent model performance through different data subsets, providing a more reliable assessment of the model's generalization capabilities.

6. **Architectural Adjustments**:

- Explore architectural adjustments, such as for the enhancement of the number of convolutional layers and incorporating batch normalization, to enhance feature extraction and model stability.

This thesis underscores the critical balance between model complexity and regularization in achieving robust image classification performance. While CNNs demonstrate powerful learning capabilities, preventing overfitting remains a significant challenge. Through strategic implementation of early stopping, regularization techniques, diverse data augmentation, and thorough hyperparameter tuning, it is possible to develop CNN models that not only perform well

on training data but also generalize effectively to unseen data. Future work should continue to explore innovative regularization methods and more complex augmentation techniques to push the boundaries of CNN performance in image classification tasks.

6. References

1. Cheng Lei, Benlin Hu, Dong Wang, Shu Zhang, Zhenyu Chen, “A Preliminary Study on Data Augmentation of Deep Learning for Image Classification.” State Key Laboratory of Novel Software Technology, Nanjing University Software Testing Engineering Laboratory of Jiangsu Province
2. LIANG HUANG , LIPING QIAN , YUAN WU, “Data Augmentation for Deep Learning-Based Radio Modulation Classification.” IEEE Access SPECIAL SECTION ON ARTIFICIAL INTELLIGENCE FOR PHYSICAL-LAYER WIRELESS COMMUNICATIONS.
3. Phillip Chlap, Hang Min, Nym Vandenberg, Jason Dowling, Lois Holloway and Annette Haworth, “A review of medical image data augmentation techniques for deep learning applications.” Journal of Medical Imaging and Radiation Oncology 65 (2021) 545–563
4. Jason Wang, Luis Perez, The Effectiveness of Data Augmentation in Image Classification using Deep Learning
5. Van Hiep Phung and Eun Joo Rhee, “A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets.” J. lnf. Commun. Converg. Eng. 16(3): 173-178, Sep. 2018
6. Sihyeon Kim, Sanghyeok Lee, Dasol Hwang, Jaewon Lee, Seong Jae Hwang, Hyunwoo J. Kim, “Point Cloud Augmentation with Weighted Local Transformations.” Korea University 2University of Pittsburgh
7. Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, and Joseph Barfett, “IMAGE AUGMENTATION USING RADIAL TRANSFORM FOR TRAINING DEEP NEURAL NETWORKS.” Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada †Department of Medical Imaging, St. Michael’s Hospital, University of Toronto, Toronto, Canada
8. Cheng Lei, Benlin Hu, Dong Wang, Shu Zhang, Zhenyu Chen , “A Preliminary Study on Data Augmentation of Deep Learning for Image Classification.” State Key Laboratory of Novel Software Technology, Nanjing University Software Testing Engineering Laboratory of Jiangsu Provinc
9. W. G. Hatcher and W. Yu, "A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends," in IEEE Access, vol. 6, pp. 24411-24432, 2018. doi: 10.1109/ACCESS.2018.2830661
10. Emmert-Streib F, Yang Z, Feng H, Tripathi S, Dehmer M. An Introductory Review of Deep Learning for Prediction Models With Big Data. Front Artif Intell. 2020;3:4. Published 2020 Feb 28. doi:10.3389/frai.2020.00004

11. Alzubaidi L, Zhang J, Humaidi AJ, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data. 2021;8(1):53. doi:10.1186/s40537-021-00444-8

12. Aggarwal R, Sounderajah V, Martin G, et al. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. NPJ Digit Med. 2021;4(1):65. Published 2021 Apr 7. doi:10.1038/s41746-021-00438-z

13. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, ‘‘Going deeper with convolutions,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2015, pp. 1–9.

14. A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘‘Imagenet classification with deep convolutional neural networks,’’ in Proc. Adv. Neural Inf. Pro- cess. Syst. (NIPS), 2012, pp. 1097–1105.

15. S. Peng, H. Jiang, H. Wang, H. Alwageed, Y. Zhou, M. M. Sebdani, and Y.-D. Yao, ‘‘Modulation classification based on signal constellation dia- grams and deep learning,’’ IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 3, pp. 718–727, Mar. 2019.

16. B. Tang, Y. Tu, Z. Zhang, and Y. Lin, ‘‘Digital signal modulation classifica- tion with data augmentation using generative adversarial nets in cognitive radio networks,’’ IEEE Access, vol. 6, pp. 15713–15722, 2018.

17. T. J. O’Shea, T. Roy, and T. C. Clancy, ‘‘Over-the-air deep learning based radio signal classification,’’ IEEE J. Sel. Topics Signal Process., vol. 12, no. 1, pp. 168–179, Feb. 2018

18. P. Triantaris, E. Tsimbalo, W. H. Chin, and D. Gündüz, ‘‘Automatic mod- ulation classification in the presence of interference,’’ in Proc. Eur. Conf. Netw. Commun., Jun. 2019, pp. 549–553.

19. Albelwi S, Mahmood A. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. Entropy. 2017; 19(6):242. https://doi.org/10.3390/e19060242

20. P. G. M. Sobha and P. A. Thomas, "Deep Learning for Plant Species Classification Survey," 2019 International Conference on Advances in Computing, Communication and Control (ICAC3), 2019, pp. 1-6. doi: 10.1109/ICAC347590.2019.9036796

21. Mao K, Lu D, E D, Tan Z. A Case Study on Attribute Recognition of Heated Metal Mark Image Using Deep Convolutional Neural Networks. Sensors (Basel). 2018 Jun 7;18(6):1871. doi: 10.3390/s18061871

22. M. Kaloev and G. Krastev, "Comparative Analysis of Activation Functions Used in the Hidden

Layers of Deep Neural Networks," 2021 3rd International Congress on HumanComputer Interaction, Optimization and Robotic Applications (HORA), 2021, pp. 1-5. doi: 10.1109/HORA52670.2021.9461312

23. Adam Gibson; Josh Patterson. (2017). Deep Learning. O'Reilly Media, Inc.
24. J. Xu, Z. Li, B. Du, M. Zhang and J. Liu, "Reluplex made more practical: Leaky ReLU," 2020 IEEE Symposium on Computers and Communications (ISCC), 2020, pp. 1-7. doi: 10.1109/ISCC50000.2020.9219587
25. Bhuiyan MAM, Sahi RK, Islam MR, Mahmud S. Machine Learning Techniques Applied to Predict Tropospheric Ozone in a Semi-Arid Climate Region. Mathematics. 2021; 9(22):2901. https://doi.org/10.3390/math9222901
26. Diederik P. Kingma, Jimmy Ba (2017). Adam: A Method for Stochastic Optimization. arXiv:1412.6980
27. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).
28. Chandran P, Asber J, Thiery F, Odelius J, Rantatalo M. An Investigation of Railway Fastener Detection Using Image Processing and Augmented Deep Learning. Sustainability. 2021; 13(21):12051. https://doi.org/10.3390/su132112051
29. Salman Khan; Hossein Rahmani; Syed Afaq Ali Shah; Mohammed Bennamoun; Gerard Medioni; Sven Dickinson, A Guide to Convolutional Neural Networks for Computer Vision, Morgan & Claypool, 2018.
30. J Brownlee. What is the Difference Between a Batch and an Epoch in a Neural Network? Machine Learning Mastery, 2018.
31. I. Kandel and M. Castelli, The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset, ICT Express (2020). https://doi.org/10.1016/j.icte.2020.04.010.
32. Mridha MF, Hamid MA, Monowar MM, Keya AJ, Ohi AQ, Islam MR, Kim J-M. A Comprehensive Survey on Deep-Learning-Based Breast Cancer Diagnosis. Cancers. 2021; 13(23):6116. https://doi.org/10.3390/cancers13236116
33. Papandrianos N, Papageorgiou E, Anagnostis A, Papageorgiou K. Bone metastasis classification using whole body images from prostate cancer patients based on convolutional neural networks application. PLoS One. 2020 Aug 14;15(8):e0237213. doi: 10.1371/journal.pone.0237213

34. Zakir Ullah M, Zheng Y, Song J, Aslam S, Xu C, Kiazolu GD, Wang L. An Attention- Based Convolutional Neural Network for Acute Lymphoblastic Leukemia Classification. Applied Sciences. 2021;11(22):10662. https://doi.org/10.3390/app112210662

35. Gu, Shanqing; Pednekar, Manisha; and Slater, Robert (2019) "Improve Image Classification Using Data Augmentation and Neural Networks," SMU Data Science Review: Vol. 2 : No. 2, Article 1.

36. Sokolova M, Mompó Alepuz A, Thompson F, Mariani P, Galeazzi R, Krag LA. A Deep Learning Approach to Assist Sustainability of Demersal Trawling Operations. Sustainability. 2021; 13(22):12362. https://doi.org/10.3390/su132212362

37. Potluri S., Fasih A., Vutukuru L.K., Machot F.A., Kyamakya K. (2012) CNN Based High Performance Computing for Real Time Image Processing on GPU. In: Unger H., Kyamaky K., Kacprzyk J. (eds) Autonomous Systems: Developments and Trends. Studies in Computational Intelligence, vol 391. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24806-1_20

38. Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F. Manso (2020); The Computational Limits of Deep Learning. arXiv:2007.05558.

39. Prerepa Gayathri, Aiswarya Dhavileswarapu, Sufyan Ibrahim, Rahul Paul, and Reena GuptaExploring the Potential of VGG-16 Architecture for Accurate Brain Tumor Detection Using Deep Learning

40. S. Serte, A. Serener, and F. Al-Turjman, “Deep learning in medical imaging: A brief review,” Transactions on Emerging Telecommunications Technologies, vol. 33, oct 2022. [30] A.

41. Ajit, K. Acharya, and A. Samanta, “A Review of Convolutional Neural Networks,” in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), pp. 1–5, IEEE, feb 2020.

42. S. Batra, S. S. Malhi, G. Singh, and M. Mahajan, “A brief overview on deep learning methods for lung cancer detection using medical imaging,” Think India Journal, vol. 22, no. 30, pp. 1279–1288, 2019.

43. Liying Yong, Le Ma, Dandan Sun, Liping Du, Application of MobileNetV2 to waste classification, 2023