**Plant Disease Detection using Convolutional Neural Networks: A Comparative Study**

Graduation Project

submitted by

**Abdel Rahman M. Rabai**

**(320200603053)**

**Khaled M. Alkouz**

**(320200603008)**

**Mohammad O. Shaheen**

**(320200603004)**

to

the TTU Tafila Technical University

In partial fulfillment of the requirements for awarding the degree

to

Bachelor of Information Technology

in

Computer Science/Data Science and Artificial Intelligence



Department of Computer Science

College of Information Technology & Communication

May 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMPUTER SCIENCE\ DATA SCIENCE & ARTIFIAL INTELLIGENCE**

**College of Information Technology & Communication**

# **CERTIFICATE**

This is to certify that the graduation project entitled Plant Disease Detection using Convolutional Neural Networks: A Comparative Study submitted by Abdel Rahman M. Rabai, Khaled M. Alkouz and Mohammad O. Shaheen to the TTU Tafila Technical University in partial fulfillment of the requirements for the award of the Bachelor of Information Technology in Computer Science/Data Science and Artificial Intelligence in is a bonafide record of the project work carried out by them under my guidance and supervision. This project in any form has not been submitted to any other University or Institute for any purpose.

Project Guide

Name:

Signature: .......................

Project Coordinator

Name:

Signature: .......................

Project Coordinator

Name:

Signature: .......................

Head of Department

Name:

Signature: .......................

# 

# **ACKNOWLEDGMENT**

*I dedicate this effort to my family and to all those who have contributed to my journey. To everyone who has supported me, to every friend who accompanied me on this path, to this beloved city and its people, who have shown us nothing but kindness and generosity. To my esteemed professors who have consistently guided, assisted, and facilitated our academic endeavors, especially my graduation project supervisor, Ph.D Ibrahim Al-Tarawneh, whose unwavering support and encouragement have been invaluable. I also extend my gratitude to the staff and employees of my university, and to all those whom I may not have mentioned. Thank you from the depths of my heart, and I look forward to a bright future where our paths may cross again.*

# **ABSTRACT**

This study concentrates on the detection of plant diseases using Convolutional Neural Networks (CNNs). Multiple CNN architectures, including ResNet50, VGG16, and a custom CNN model, were implemented and compared to identify the most accurate, stable, and effective model for this task. All three models employed the same regularization techniques, such as dropout, during training to ensure stability.

The results indicated high performance across models. The ResNet50 model achieved 99% accuracy with a training loss of 0.4291, a validation loss of 3.0567, and a validation accuracy of 97%. The VGG16 model attained an accuracy of 83% with a training loss of 0.48, a validation loss of 0.51, and a validation accuracy of 85%. The custom model achieved 98% accuracy with a training loss of 0.05, a validation loss of 0.15, and a validation accuracy of 95%.

The dataset used in this study comprised approximately 87,000 RGB images of healthy and diseased crop leaves, categorized into 38 classes. The data was split into an 80/20 ratio for training and validation, with an additional set of 33 images used for testing. This comparative study provides insights into the performance of different CNN architectures for plant disease detection, highlighting the strengths and weaknesses of each model.

## 

## Content

[CERTIFICATE 2](#_Toc167170266)

[ACKNOWLEDGMENT 2](#_Toc167170267)

[ABSTRACT 3](#_Toc167170268)

[Content 4](#_Toc167170269)

[**Chapter 1** 6](#_Toc167170270)

[**1.1 What is the problem?** 6](#_Toc167170271)

[**1.2** **What is our idea about and why it is important?** 7](#_Toc167170272)

[**1.3** **How to do it?** 7](#_Toc167170273)

[**Chapter 2** 8](#_Toc167170274)

[**2.1** **Dataset** 8](#_Toc167170275)

[**2.1.1** **Data Acquisition** 8](#_Toc167170276)

[**2.1.2** **Plant and Disease classes** 9](#_Toc167170277)

[**2.1.3** **Data Splitting** 11](#_Toc167170278)

[**2.2 Data Preprocessing** 11](#_Toc167170279)

[**2.2.1** **Libraries** 11](#_Toc167170280)

[**2.2.2** **Image Augmentation** 12](#_Toc167170281)

[**2.2.3** **Image Resizing** 14](#_Toc167170282)

[**Chapter 3** 14](#_Toc167170283)

[**3.1** **Convolution Neural Network** 14](#_Toc167170284)

[**3.1.1** **What is a neural network[17]?** 14](#_Toc167170285)

[**3.1.2** **What is Convolutional Neural Networks[17]?** 15](#_Toc167170286)

[**3.1.3** **How do convolutional neural networks work[17]?** 16](#_Toc167170287)

[**3.1.4** **Convolutional layer** 16](#_Toc167170288)

[**3.1.5** **Pooling layer** 17](#_Toc167170289)

[**3.1.6** **Fully-connected layer** 18](#_Toc167170290)

[**3.2** **CNN Models Architecture** 18](#_Toc167170291)

[**3.2.1** **VGG16[18]** 18](#_Toc167170292)

[**3.2.2** **ResNet[19]** 21](#_Toc167170293)

[**3.2.2.1 Residual Blocks** 21](#_Toc167170294)

[**3.2.2.2 ResNet50** 22](#_Toc167170295)

[**3.2.3** **Custom Model** 24](#_Toc167170296)

[**3.2.3.1**  **Convolutional Layers** 26](#_Toc167170297)

[**3.2.3.2**  **Dropout Layer** 26](#_Toc167170298)

[**3.2.3.3** **Fully Connected Layers** 26](#_Toc167170299)

[**Chapter 4** 27](#_Toc167170300)

[**4.1** **Training Parameters** 27](#_Toc167170301)

[**4.1.1** **Learning Rate** 27](#_Toc167170302)

[**4.1.2** **Optimizer** 27](#_Toc167170303)

[**4.2** **hardware/software environment** 27](#_Toc167170304)

[**Chapter 5** 28](#_Toc167170305)

[**5.1** **Model Performance** 28](#_Toc167170306)

[***5.1.1.1***  ***Accuracy and Loss*** 28](#_Toc167170307)

[***5.1.1.2***  ***Precision, Recall, F1-score for each Class*** 29](#_Toc167170308)

[**5.1.2** **ResNet Performance** 31](#_Toc167170309)

[***5.1.2.1***  ***Accuracy and Loss*** 31](#_Toc167170310)

[***5.1.2.2***  ***Precision, Recall, F1-score for each Class*** 31](#_Toc167170311)

[**5.1.3** **Custom Model Performance** 34](#_Toc167170312)

[***5.1.3.2***  ***Precision, Recall, F1-score for each Class*** 34](#_Toc167170313)

[**5.2** **Discussion and Summary** 36](#_Toc167170314)

[**Conclusion** 38](#_Toc167170315)

[References 39](#_Toc167170316)

## **Chapter 1**

INTRODUCTION AND PROBLEM FRAMING

This chapter introduces the project’s contents to the reader. Define a Clear, Concise, Concrete the Problem Statement

### **1.1 What is the problem?**

Over the past century, the world has experienced a significant increase in population alongside a decrease in mortality rates, as illustrated in *Figures 1 & 2[1][2]*. This growth can be attributed to several factors. Historically, the population has consistently risen since the end of the Black Death around 1350. The Industrial Revolution notably accelerated this growth through improvements in food production and sanitation.

Population growth is not uniform across the globe. Developing countries generally exhibit higher fertility rates, while developed countries have lower rates. According to the United Nations, the global population is projected to continue growing in the coming decades, although the rate of increase is expected to decelerate.

The rise in population inevitably leads to greater consumption of plant and animal products. Approximately 3 billion people live outside the cash economy in the world's poorest nations, where food security and consistent supply are daily concerns. Chronic malnutrition is a leading cause of death and disease, particularly affecting young children. Every 5-10 seconds, a child dies from undernutrition, and every minute, a person goes blind due to Vitamin A deficiency. Additionally, iron deficiency is widespread among individuals in tropical and subtropical regions. Increasing plant diseases contribute to food insecurity and the spread of illnesses[3].

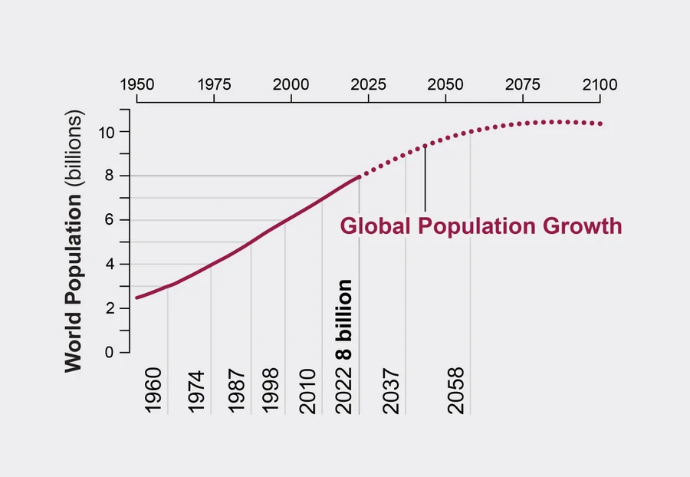


Figure 2: World Population Increase



Figure 2: Decrease in Mortality Rates

### **1.2 What is our idea about and why it is important?**

The increase in population has led to a corresponding rise in the consumption of both plant and animal resources. Specifically focusing on plant resources, there has been a noticeable increase in the prevalence of crop diseases, which can subsequently be transmitted to consumers. This observation inspired us to explore strategies for mitigating the spread of crop diseases.

We have decided to begin with a small-scale initiative, targeting a limited number of plants and diseases, with plans for future expansion. Our objective is to support amateur farmers who may lack adequate knowledge about plant diseases and their remedies. Our solution allows farmers to take a picture of an infected plant leaf and upload it to our model. The system will then diagnose the issue, provide general information about the disease, and suggest potential treatments.

By enabling early detection of plant diseases, our project aims to reduce the use of incorrect treatments, which can exacerbate the disease, and to enhance knowledge and awareness among farmers. Ultimately, our goal is to contribute to reducing the spread of crop diseases and to facilitate more effective farming practices.

### **1.3 How to do it?**

To address this problem using artificial intelligence, it is essential to develop a Convolutional Neural Network (CNN) model. CNNs, a subset of deep learning models, are particularly adept at processing visual data. Their effectiveness stems from their ability to learn spatial hierarchies of features through a multi-layered architecture, making them highly suitable for various image-related tasks.

The process begins with data collection. Numerous websites offer datasets that can be utilized for this purpose, such as Kaggle[4], UCI[5], Google Dataset Search[6], etc.…, after fetching the required dataset, we move to preprocessing step.

In this step we are dealing with the data and preparing to fit into the model. When we finish all the preprocess, we are ready now to create the CNN model, get the result, evaluation, etc.…

Until the end of the model, we will explain each part in detail.

## 

## **Chapter 2**

Data and Preprocessing

In this chapter we will discuss how we are dealing with the data before fitting it into the model

## **2.1 Dataset**

### **2.1.1 Data Acquisition**

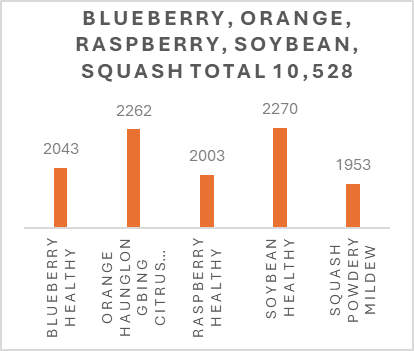
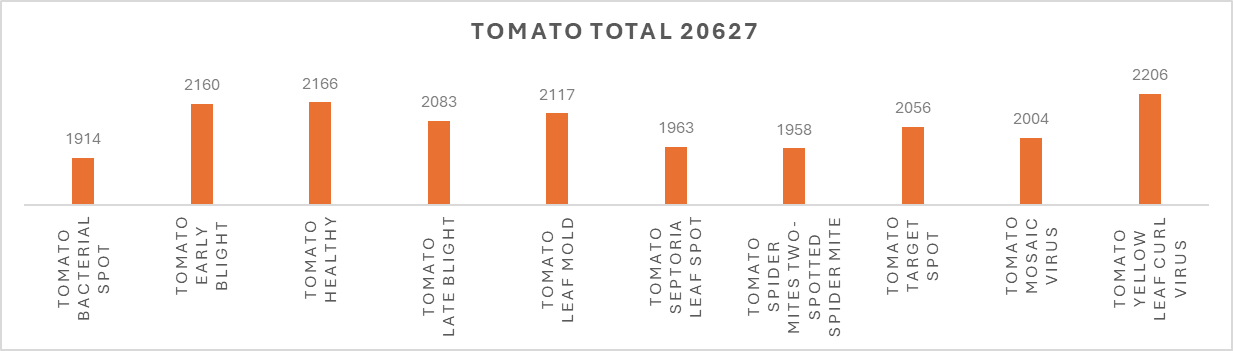
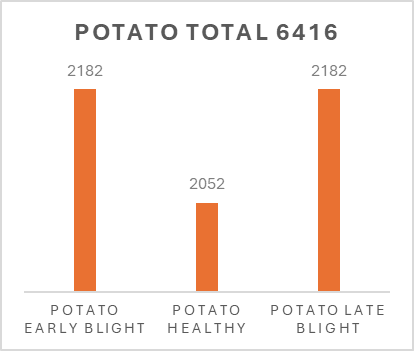
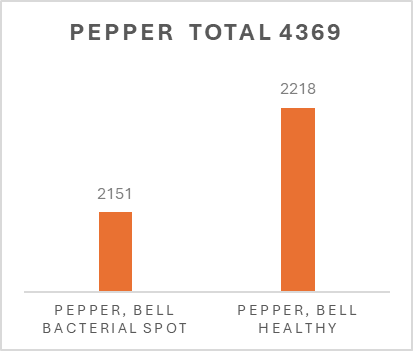
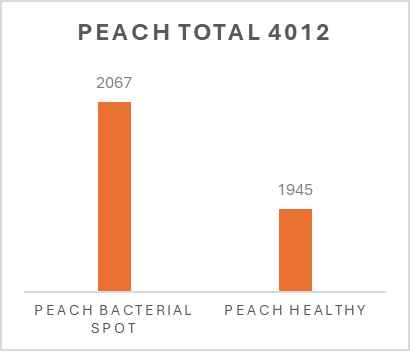
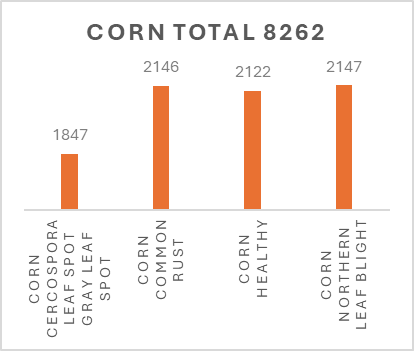
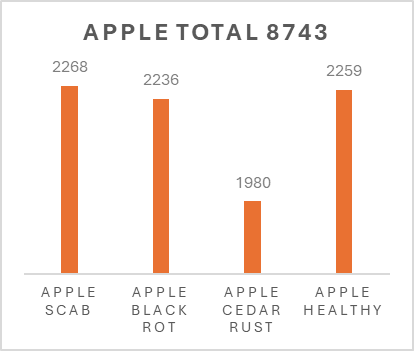
In this section we describe how we are collecting the data from open-source online Repositories or trusted website. We used the New Plant Diseases Dataset from Kaggle which in turn is an augmented data from the original dataset, the original data is PlantVallage-dataset[3]. The dataset contains 87,000 RGB images of diseased and healthy crop leaves, which are categorized into 38 different classes, the total is dataset divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose[4].

### **2.1.2 Plant and Disease classes**

This section shows us the plant and disease classes that we get for the model to deal with, thirty-eight classes, fourteen plants between healthy and diseased, twenty-six different diseases showing below:

1. Apple scab
2. Apple Black rot
3. Apple Cedar rust
4. Apple healthy
5. Blueberry healthy
6. Cherry Powdery mildew.
7. Cherry healthy
8. Corn Cercospora leaf spot gray leaf spot
9. Corn Cercospora leaf spot gray leaf spot
10. Corn Cercospora leaf spot gray leaf spot
11. Corn healthy
12. Grape Black rot
13. Grape Esca (Black Measles)
14. Grape Leaf blight (Isariopsis Leaf Spot)
15. Grape healthy
16. Orange Haunglongbing (Citrus greening)
17. Peach Bacterial spot
18. Peach healthy
19. Pepper, bell Bacterial spot
20. Pepper, bell healthy
21. Potato Early blight
22. Potato Late blight
23. Potato healthy
24. Raspberry healthy
25. Soybean healthy
26. Squash Powdery mildew.
27. Strawberry Leaf scorch
28. Strawberry healthy
29. Tomato Bacterial spot.
30. Tomato Early blight.
31. Tomato healthy.
32. Tomato Late Blight.
33. Tomato Leaf Mold
34. Tomato Septoria leaf spot.
35. Tomato Spider mites Two-spotted spider mites.
36. Tomato Target Spot
37. Tomato Yellow Leaf Curl Virus
38. Tomato mosaic virus.

Figure 3: Plant and Disease classes.



### **2.1.3 Data Splitting**

As we mentioned before our data is splitting into 80/20 ratio, 80% for the training set with total 70295 images, and 20% for the validation set with total 17572 images belonging to thirty-eight classes. Finally, thirty-three test images to test the performance of our model later.

Figure 4: Splitting Data

### **2.2 Data Preprocessing**

In this section we will dive into the preprocessing techniques and transforming the data, how to deal with unstructured data, and what steps applied to images (resizing, augmentation etc...).

### **2.2.1 Libraries**

There are many Python libraries we can use on preprocessing step; our choice was to use the Pandas[7], Open-cv, and TensorFlow[8] libraries Because it contains a huge number of build-in functions that will help us in the preprocessing step and other steps like model building and evaluation.

### **2.2.2 Image Augmentation**

Deep neural networks require a lot of training data to obtain good results and prevent overfitting. However, it is often very difficult to get enough training samples. Multiple reasons could make it very hard or even impossible to gather enough data:

* To make a training dataset, you need to obtain images and then label them. For example, you need to assign correct class labels if you have an image classification task. For an object detection task, you need to draw bounding boxes around objects. For a semantic segmentation task, you need to assign a correct class to each input image pixel. This process requires manual labor, and sometimes it could be very costly to label the training data. For example, to correctly label medical images, you need expensive domain experts.

* Sometimes even collecting training images could be hard. There are many legal restrictions for working with healthcare data, and obtaining it requires a lot of effort. Sometimes getting the training images is more feasible, but it will cost a lot of money. For example, to get satellite images, you need to pay a satellite operator to take those photos. To get images for road scene recognition, you need an operator that will drive a car and collect the required data.

Image augmentation is a process of creating new training examples from the existing ones. To make a new sample, you slightly change the original image. For instance, you could make a new image a little brighter; you could cut a piece from the original image; you could make a new image by mirroring the original one, etc.

Common image augmentation techniques include:

1. Geometric Transformations:

* Rotation: Rotating the image by a certain angle.
* Translation: Shifting the image horizontally or vertically.
* Scaling: Resizing the image.
* Shearing: Applying a shear transformation.
* Flipping: Flipping the image horizontally or vertically

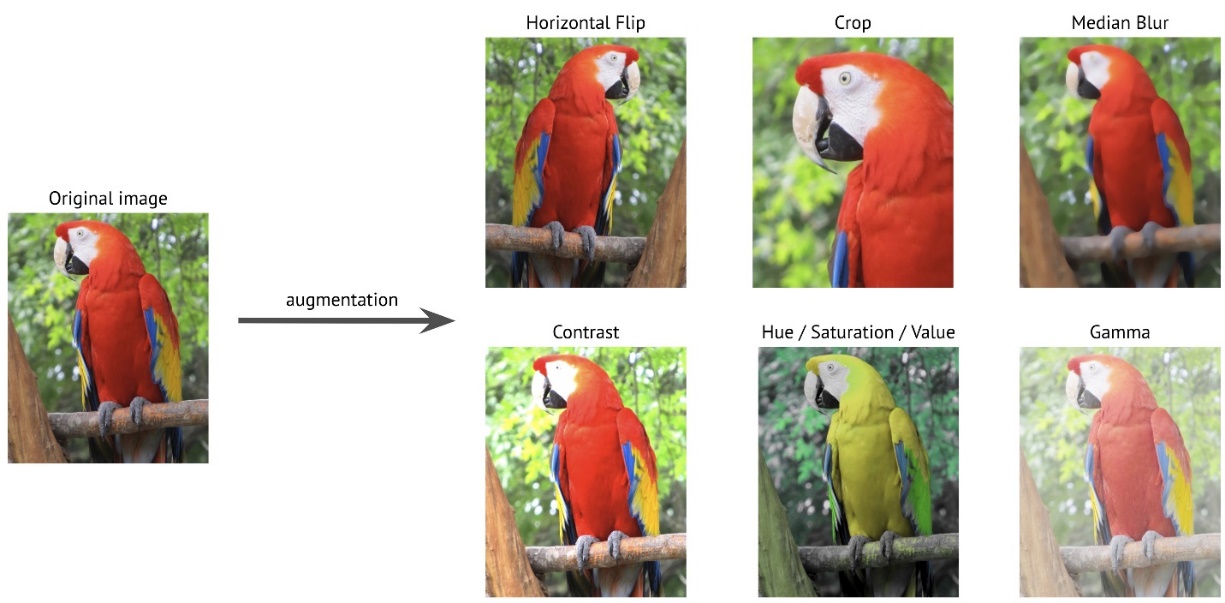
1. Color Space Transformations:

* Brightness Adjustment: Changing the brightness of the image.
* Contrast Adjustment: Modifying the contrast of the image.
* Saturation Adjustment: Changing the saturation levels.
* Hue Adjustment: Altering the hue of the image.

1. Noise Injection:
   * Gaussian Noise: Adding Gaussian noise to the image.
   * Salt and Pepper Noise: Introducing random black and white pixels.
2. Cropping:
   * Random Cropping: Cropping random portions of the image.
   * Central Cropping: Cropping the central part of the image.
3. Affine Transformations:
   * Applying linear transformations that preserve points, straight lines, and planes.
4. Elastic Deformations:
   * Distorting the image in a non-linear way, often used to simulate realistic variations like changes in perspective.
5. Occlusion:
   * Randomly covering parts of the image to simulate occlusions.
6. Normalization:
   * Adjusting the image pixel values to a standard range, often between 0 and 1 or -1 and 1.
7. advanced Techniques:
   * Cutout: Randomly masking out square regions of the image.
   * Mixup: Creating a new training example by combining two images and their corresponding labels.
   * CutMix: Combining two images by cutting and pasting parts of one image onto another.

In addition, we found many researchers that showed the efficiency of data augmentation for improving deep learning in image classification problem. Hu et al[11] shown that “***augmentation methods work well even on small dataset. Applying higher augmentation rate is not cost-effective, because the marginal benefit is gradually reduced while the time, space and other costs increase linearly. Our study recommends that the best augmentation rate is 2-3 times. The simple augmentation methods such as translation, rotation, scaling and shearing can achieve good results, and in fact, much better than more complicated methods*.**” Yang et al[12] Emphasize in their work that data augmentation is an effective solution to the lack of tagged image data.

Here are some examples of transformations of the original image that will create a new training sample.



By applying those transformations to the original training dataset, you could create an almost infinite number of new training samples[10].

In our situation, the data is already augmented so we can directly move to the next step.

### **2.2.3 Image Resizing**

There are several studies that have proven that resizing of images significantly affects for training time and model performance, Avidan et al [13] proposed seam caving which performs retargeting by inserting and removing streams of pixels, which are called seam. It passes through less important features. Pritch et al [14], presented movement of maps for the rearrangement of pixels. It is composed of graph labelling problems for editing different images in various applications. Mingxing et al [15] proposed a technique for neural network design architecture for object detection and suggested several ways of optimization to enhance efficiency of neural network models. Recently Saponara et al[16] shown that the experimental results when the original image downscaled, it improved the training computation time, while the training time increased when these images upscaled.

In our work to ensure that the fixed CNN architecture is not modified to achieve the highest performance desired from models like VGG16 and ResNet50. these models require resizing images to (224,224) as an input image size while our custom model has an (128,128) input size image to achieve the best performance.

## **Chapter 3**

Model Architecture

### **3.1 Convolution Neural Network**

### **3.1.1 What is a neural network[17]?**

A neural network is a machine learning program, or model, that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions. Every neural network consists of layers of nodes, or artificial neurons—an input layer, one or more hidden layers, and an output layer. Each node connects to others and has its own associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. Neural networks rely on training data to learn and improve their accuracy over time. Once they are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the best-known examples of a neural network is Google’s search algorithm. Neural networks are sometimes called artificial neural networks (ANNs) or simulated neural networks (SNNs). They are a subset of machine learning, and at the heart of deep learning models.

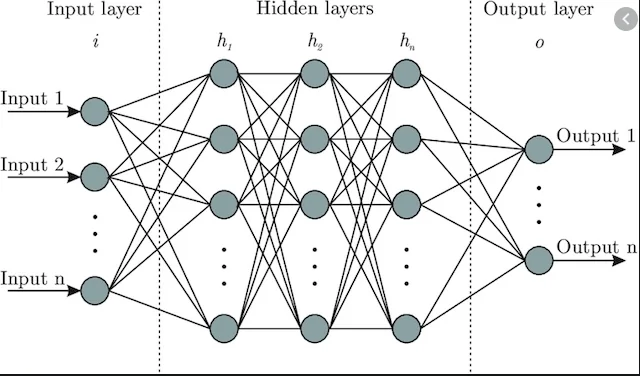


Figure 5: Neural Network

This was an overview of a Neural Network, now let’s dive into CNN.

### **3.1.2 What is Convolutional Neural Networks[17]?**

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

While we primarily focused on feedforward networks in that article, there are various types of neural nets, which are used for different use cases and data types. For example, recurrent neural networks are commonly used for natural language processing and speech recognition whereas convolutional neural networks (ConvNets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

### **3.1.3 How do convolutional neural networks work[17]?**

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

* Convolutional layer
* Pooling layer
* Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

### **3.1.4 Convolutional layer**

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyperparameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

1. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
2. Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
3. Zero-padding is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:

Valid padding: This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.

Same padding: This padding ensures that the output layer has the same size as the input layer.

Full padding: This type of padding increases the size of the output by adding zeros to the border of the input.

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

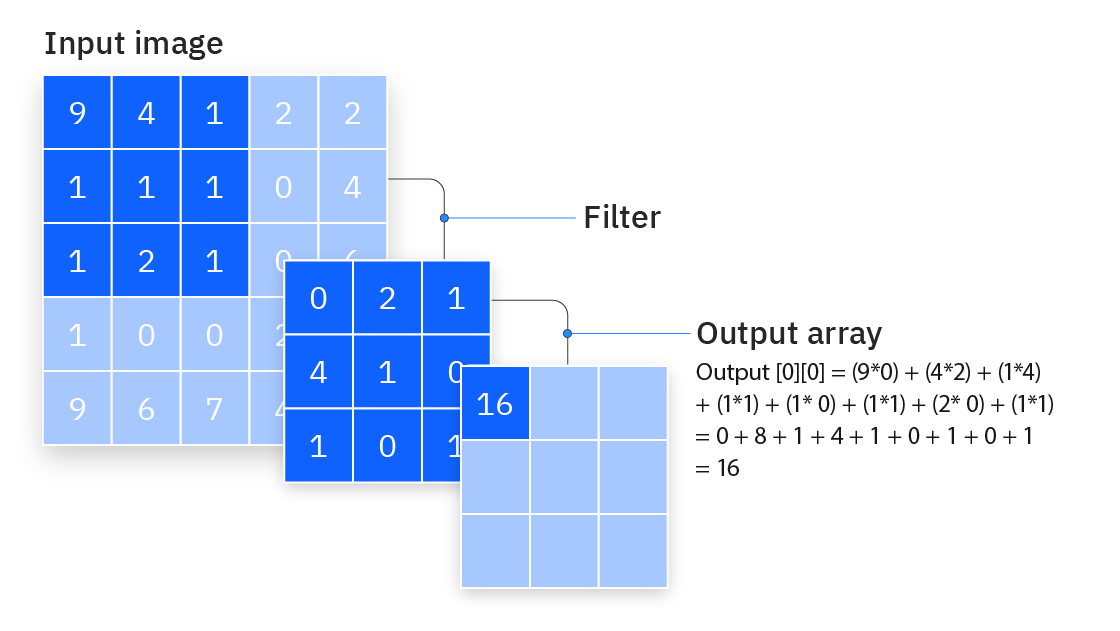


Figure 6: The Process of Convolution

### **3.1.5 Pooling layer**

Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

* Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
* Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

### **3.1.6 Fully-connected layer**

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

### **3.2 CNN Models Architecture**

Now let’s look at what we do. We create three models (VGG16, ResNet50, and a Custom model)

### **3.2.1 VGG16[18]**

VGG is a deep convolutional neural network that was proposed by Karen Simonyan and Andrew Zisserman [19]. VGG is an acronym for their group name, Visual Geometry Group, from the Oxford University. This model secured 2nd place in the ILSVRC-2014 competition where 92.7% classification performance was achieved. The VGG model investigates the depth of layers with a very small convolutional filter size (3 × 3) to deal with large-scale images. The authors released a series of VGG models with different layer lengths, from 11 to 19, which is presented in *table.1*.

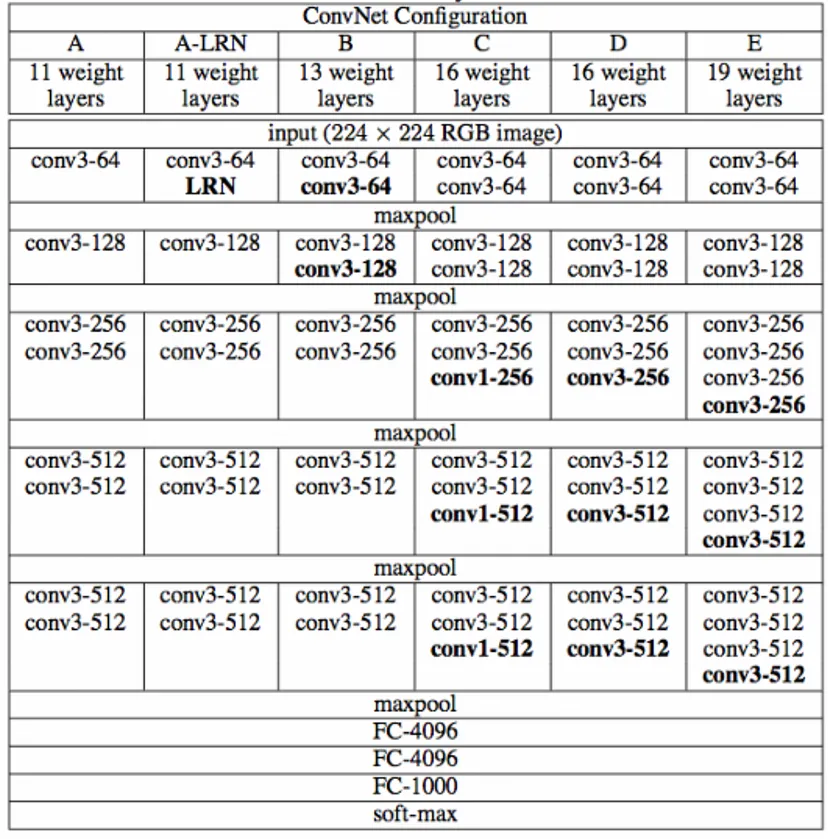


Table 1: Different configurations of VGG

The structure of VGG16 is described by the following figure:

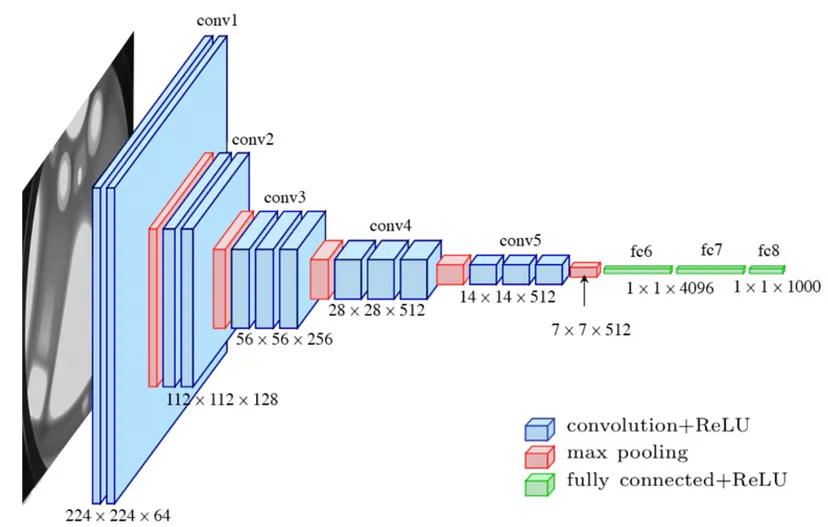


Figure 7: The Architecture of VGG16

VGG16 is composed of 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. Therefore, the number of layers having tunable parameters is 16 (13 convolutional layers and 3 fully connected layers). That is the reason why the model’s name is VGG16. The number of filters in the first block is 64, then this number is doubled in the later blocks until it reaches 512. This model is finished by two fully connected hidden layers and one output layer. The two fully connected layers have the same neuron numbers which are 4096. The output layer consists of 1000 neurons corresponding to the number of categories of the ImageNet dataset.

Now, let’s see the detailed information in each layer of the model:

**vgg16\_model = VGG16()**

**vgg16\_model.summary()**

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
conv2d (Conv2D) (None, 224, 224, 64) 1792   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 224, 224, 64) 36928   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 112, 112, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_2 (Conv2D) (None, 112, 112, 128) 73856   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_3 (Conv2D) (None, 112, 112, 128) 147584   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 56, 56, 128) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_4 (Conv2D) (None, 56, 56, 256) 295168   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_5 (Conv2D) (None, 56, 56, 256) 590080   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_6 (Conv2D) (None, 56, 56, 256) 590080   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_2 (MaxPooling2 (None, 28, 28, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_7 (Conv2D) (None, 28, 28, 512) 1180160   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_8 (Conv2D) (None, 28, 28, 512) 2359808   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_9 (Conv2D) (None, 28, 28, 512) 2359808   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_3 (MaxPooling2 (None, 14, 14, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_10 (Conv2D) (None, 14, 14, 512) 2359808   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_11 (Conv2D) (None, 14, 14, 512) 2359808   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_12 (Conv2D) (None, 14, 14, 512) 2359808   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_4 (MaxPooling2 (None, 7, 7, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten (Flatten) (None, 25088) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 4096) 102764544   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 4096) 16781312   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_1 (Dropout) (None, 4096) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 1000) 4097000   
=================================================================  
Total params: 138,357,544  
Trainable params: 138,357,544  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

As the number of filters increases following the model depth, hence the number of parameters increases significantly in the later layers. Especially, the parameter number in the two fully connected hidden layers is very large, with 102, 764, 544, and 16, 781, 312 parameters, respectively. It accounts for 86.4% parameters of the whole model.

### **3.2.2 ResNet[19]**

The Residual networks (ResNet in short )architecture is one of the most well-known and popular networks in the literature, introduced by Microsoft Research in 2015 in the paper written by He. et. al[20]. The residual networks proposed in this paper won the first places on ILSVRC 2015 classification task, ImageNet detection, ImageNet localization, COCO detection and COCO segmentation task.

Deeper networks produce higher training and test error than shallower counterparts caused by degradation problem. To address this problem residual networks use identity mapping to reference layer input. Then ResNets can gain accuracy by increasing depth and easy to optimize. To understand how it does you should read the paper.

#### **3.2.2.1 Residual Blocks**

Residual building block is the fundamental component of residual networks with reference to the layer input, in other words identity mapping as you can see in the following figure.

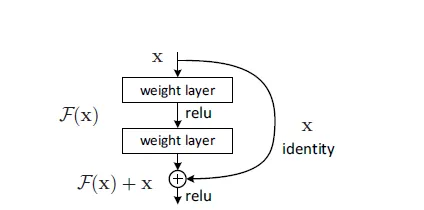


Figure 8: Residual Building Block

If desired an underlying mapping to be fitted by some stack of layers is H(x), the residual nets reformulated it to F(x)=H(x) — x. They have solved the degradation problem with this trick. The considered formula for identity mapping defined as:

(1)

Here x and y are the layers input and output respectively and F(, {}) presents residual mapping to be learned. It is obvious that the x and F dimensions should be equal in *Eqn.(1)*. When changing the input/output channels we can apply a linear projection Ws to the shortcut connection to match the dimensions.

(2)

Identity mapping is sufficient for addressing the degradation problem and economical, so that linear projection (Ws) is only used when dimension matching is needed.

#### **3.2.2.2 ResNet50**

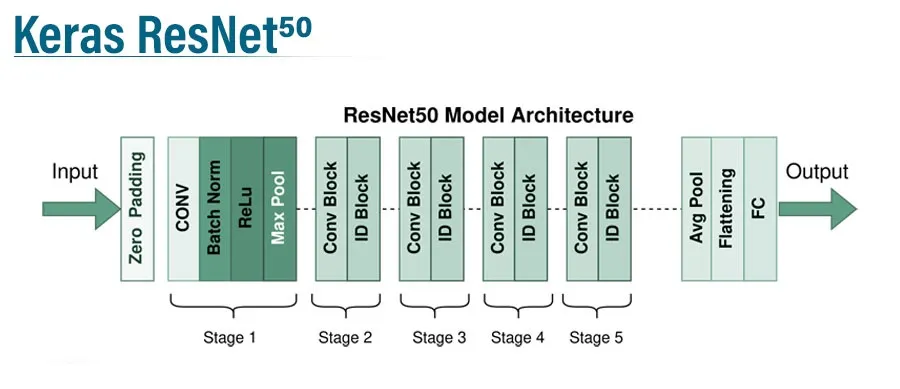


Figure 9: ResNet-50 architecture took from https://www.educba.com/keras-resnet50/

ResNet architecture combines four modules, likewise GoogleNet, each of which uses several residual blocks with the same output channels, but in 50-layer ResNet the 2-layer block in 34-layer ResNet replaced with the 3-layer bottleneck block. *Table.2* demonstrates the different ResNet models.

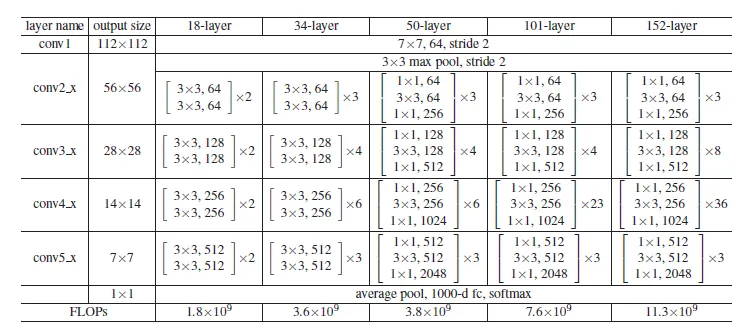


Table 2: Different ResNet Architecture from https://medium.com/@arashserej/resnet-50-83b3ff33be7d

As we show we implemented the ResNet50 model and the summary of this architecture as below:

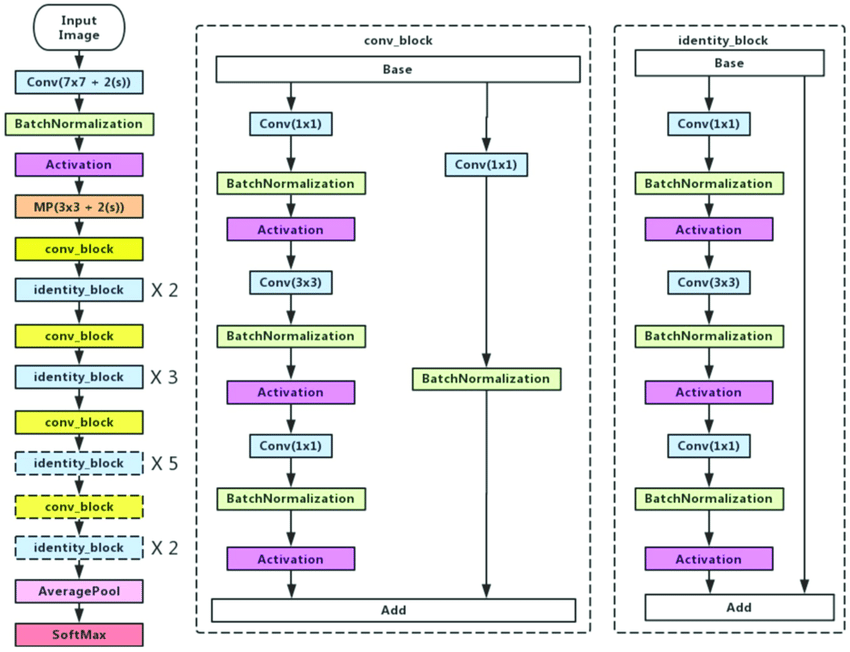


Figure 10: ResNet-50 Architecture Cited from [21]

### ***3.2.3 Custom Model***

We create CNN architecture that utilizes convolutional layers to extract features from the images, followed by pooling layers for dimensionality reduction. Dropout layers help prevent overfitting, and the final fully connected layers perform classification, predicting the most likely disease affecting the plant in the image. Now Let's discuss each part in detail.

The model can be segmented into three main parts:

1. Convolutional Layers
2. Dropout Layer
3. Fully Connected Layers

The model architecture is shown in ***figure.11***. and the model summary shown below:

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 128, 128, 32) 896

conv2d\_1 (Conv2D) (None, 128, 128, 32) 9248

max\_pooling2d (MaxPooling) (None, 64, 64, 32) 0

conv2d\_2 (Conv2D) (None, 64, 64, 64) 18496

conv2d\_3 (Conv2D) (None, 62, 62, 64) 36928

max\_pooling2d\_1 (MaxPoolin (None, 31, 31, 64) 0

conv2d\_4 (Conv2D) (None, 31, 31, 128) 73856

conv2d\_5 (Conv2D) (None, 29, 29, 128) 147584

max\_pooling2d\_2 (MaxPoolin) (None, 14, 14, 128) 0

conv2d\_6 (Conv2D) (None, 14, 14, 256) 295168

conv2d\_7 (Conv2D) (None, 12, 12, 256) 590080

max\_pooling2d\_3 (MaxPoolin (None, 6, 6, 256) 0

conv2d\_8 (Conv2D) (None, 6, 6, 512) 1180160

conv2d\_9 (Conv2D) (None, 4, 4, 512) 2359808

max\_pooling2d\_4 (MaxPoolin (None, 2, 2, 512) 0

dropout (Dropout) (None, 2, 2, 512) 0

flatten (Flatten) (None, 2048) 0

dense (Dense) (None, 1500) 3073500

dropout\_1 (Dropout) (None, 1500) 0

dense\_1 (Dense) (None, 38) 57038

=================================================================

Total params: 7842762 (29.92 MB)

Trainable params: 7842762 (29.92 MB)

Non-trainable params: 0 (0.00 Byte)

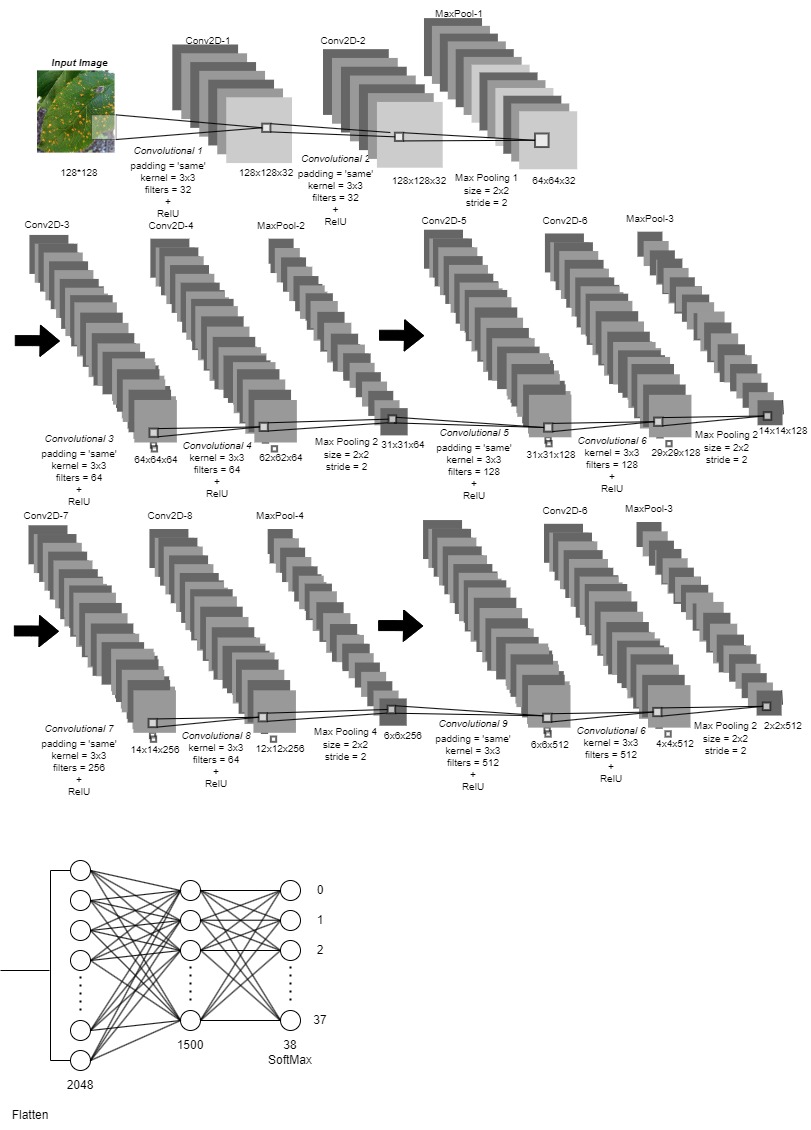


Figure 11: Custom CNN Architecture

#### **3.2.3.1 Convolutional Layers**

The core of the model consists of multiple convolutional layers. Our model contains 10 convolutional layers, these layers are responsible for extracting features from the input images. Each layer uses multiple filters (32, 64, 128, 256, 512 in this model) to learn different patterns within the images. The kernel size defines the area of the image considered by each filter, in our case we set it to 3x3. We used the padding *‘same’* which ensured that the output remains the same size as the input for each convolutional layer. Finally, the activation function in our model is the *‘ReLU’, this* activation introduces non-linearity, allowing the model to learn more complex relationships between features.

The convolutional layers are followed by a pooling layer, these layers downsample the feature maps produced by the convolutional layers, reducing the data size and computational cost. They also help the model focus on the most relevant features. we add five max-pooling layers, and this process allows it to learn increasingly complex features from the images, progressively extracting higher-level information.

#### **3.2.3.2 Dropout Layer**

A dropout layer is introduced before the fully connected layer to help prevent overfitting. During training, a random subset of neurons is temporarily dropped out, forcing the model to rely on different features in each training iteration. This helps improve generalization and reduce the model's sensitivity to specific training data.

#### **Fully Connected Layers**

1. **Flatten Layer:** This layer transforms the multi-dimensional feature maps from the convolutional layers into a single one-dimensional vector.
2. **Dense Layer:** This layer performs classification. It has 1500 neurons with a 'ReLU' activation function, followed by another dropout layer (0.5) for further regularization.
3. **Output Layer:** The final layer has 38 neurons (corresponding to the 38 disease classes) with a 'softmax' activation function. Softmax ensures the output probabilities sum to 1, indicating the likelihood of each disease class for a given image.

Overall, this CNN architecture utilizes convolutional layers to extract features from the images, followed by pooling layers for dimensionality reduction. Dropout layers help prevent overfitting, and the final fully connected layers perform classification, predicting the most likely disease affecting the plant in the image.

## **Chapter 4**

Experiment Setup

In this chapter we will mention the training parameters used for each model (e.g., learning rate, optimizer), and the hardware/software environment used for training (e.g., GPU type, libraries).

### **4.1 Training Parameters**

#### **4.1.1 Learning Rate**

The learning rate is a crucial hyperparameter in training a Convolutional Neural Network. It controls the size of the steps the model takes during optimization, impacting how quickly it learns and converges to a good solution.

we set it to 0.0001, it’s small and will making the learning process slower but potentially leading to better convergence and avoiding overfitting.

#### **4.1.2 Optimizer**

An optimizer is a critical component in training a Convolutional Neural Network. It essentially guides the model towards better performance by adjusting the weights and biases based on the error (difference between predicted and actual labels) during training. In our models we used the ‘Adam’ (Adaptive Moment Estimation) optimizer. This combines the benefits of Stochastic Gradient Descent (SGD) and Root Mean Square Prop (RMSprop). It maintains exponentially decaying averages of past gradients and squared gradients, dynamically adjusting the learning rate for each parameter.

### **4.2 hardware/software environment**

We deal with Google Colab Pro as an environment for our work, we used the L4 GPU with 15GB of GPU RAM, 12GB of Memory, and all the files upload and download into Drive.

## **Chapter 5**

Results and Discussion

In this chapter we will show the performance metrics for each model on the training, validation, and test sets. This includes Accuracy, Loss, Precision, Recall, F1-score, and show the Confusion Matrix. Compare the performance of the three architectures, highlighting the strengths and weaknesses of each and analyze the validation loss compared to training loss to assess overfitting. Finally, discuss the factors contributing to the performance of the Custom CNN model.

Each model has the same data fitted into it, and the same number of epochs which is 10, except VGG16 the number of epochs was 100 because we apply the EarlyStopping technique and set a 100 step per epoch.

### **5.1 Model Performance**

here we will show the performance for each model VGG16, ResNet50, and Custom.

**5.1.1 VGG16 Performance**

##### ***5.1.1.1 Accuracy and Loss***

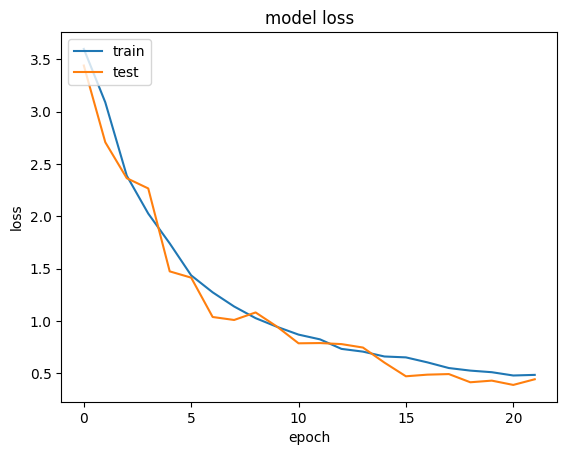


Figure 13: VGG16 Loss for Train & Valid

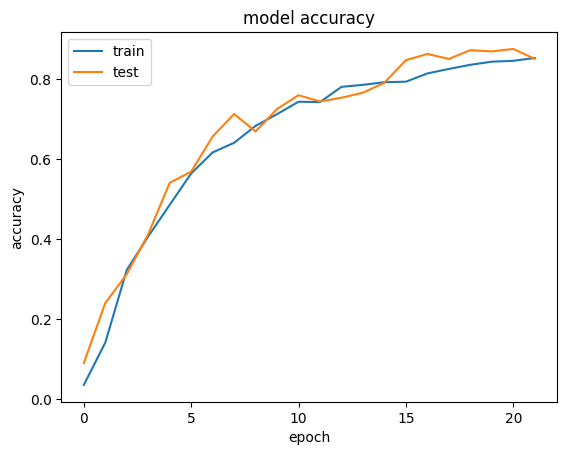


Figure 12: VGG16 Accuracy for Train & Valid

##### ***5.1.1.2 Precision, Recall, F1-score for each Class***

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | Recall | f1-score |
| Apple scab | 0.87 | 0.87 | 0.87 |
| Apple Black rot | 0.95 | 0.95 | 0.95 |
| Apple Cedar apple rust | 0.86 | 0.90 | 0.88 |
| Apple healthy | 0.80 | 0.86 | 0.83 |
| Blueberry healthy | 0.80 | 0.93 | 0.86 |
| Cherry Powdery mildew | 0.93 | 0.95 | 0.94 |
| Cherry healthy | 0.81 | 0.98 | 0.89 |
| Corn Cercospora leaf spot gray leaf spot | 0.94 | 0.70 | 0.81 |
| Corn Common rust | 0.97 | 1.00 | 0.98 |
| Corn Northern Leaf Blight | 0.77 | 0.97 | 0.86 |
| Corn healthy | 0.96 | 1.00 | 0.98 |
| Grape Black rot | 0.88 | 0.96 | 0.92 |
| Grape Esca (Black Measles) | 0.99 | 0.90 | 0.94 |
| Grape Leaf blight (Isariopsis Leaf Spot) | 0.99 | 0.97 | 0.98 |
| Grape healthy | 0.89 | 0.97 | 0.93 |
| Orange Haunglongbing | 0.94 | 0.91 | 0.93 |
| Peach Bacterial spot | 0.89 | 0.83 | 0.86 |
| Peach healthy | 0.95 | 0.91 | 0.93 |
| Pepper, bell Bacterial spot | 0.77 | 0.92 | 0.84 |
| Pepper, bell healthy | 0.88 | 0.83 | 0.85 |
| Potato Early blight | 0.82 | 0.95 | 0.88 |
| Potato Late blight | 0.95 | 0.67 | 0.78 |
| Potato healthy | 0.96 | 0.77 | 0.85 |
| Raspberry healthy | 0.95 | 0.83 | 0.89 |
| Soybean healthy | 0.96 | 0.88 | 0.92 |
| Squash Powdery mildew | 0.95 | 0.95 | 0.95 |
| Strawberry Leaf scorch | 0.93 | 0.90 | 0.92 |
| Strawberry healthy | 0.93 | 0.96 | 0.95 |
| Tomato Bacterial spot | 0.72 | 0.95 | 0.82 |
| Tomato Early blight | 0.67 | 0.71 | 0.69 |
| Tomato Late blight | 0.83 | 0.68 | 0.5 |
| Tomato Leaf Mold | 0.79 | 0.87 | 0.83 |
| Tomato Septoria leaf spot | 0.90 | 0.42 | 0.58 |
| Tomato Spider mites Two-spotted spider mite | 0.85 | 0.75 | 0.80 |
| Tomato Target Spot | 0.70 | 0.79 | 0.75 |
| Tomato Yellow Leaf Curl Virus | 0.92 | 0.88 | 0.90 |
| Tomato mosaic virus | 0.90 | 0.92 | 0.91 |
| Tomato healthy | 0.90 | 0.95 | 0.92 |

Table 3: VGG16 Precision, Recall, and F1-Score

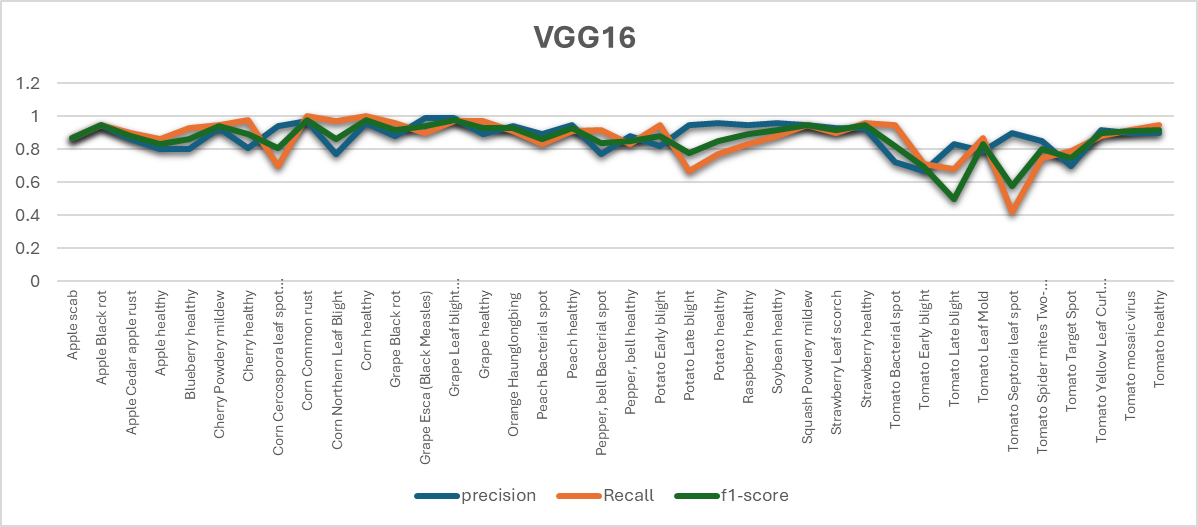


Figure 14: Precision, Recall, and F1-Score

***5.1.1.3 Confusion Matrix***

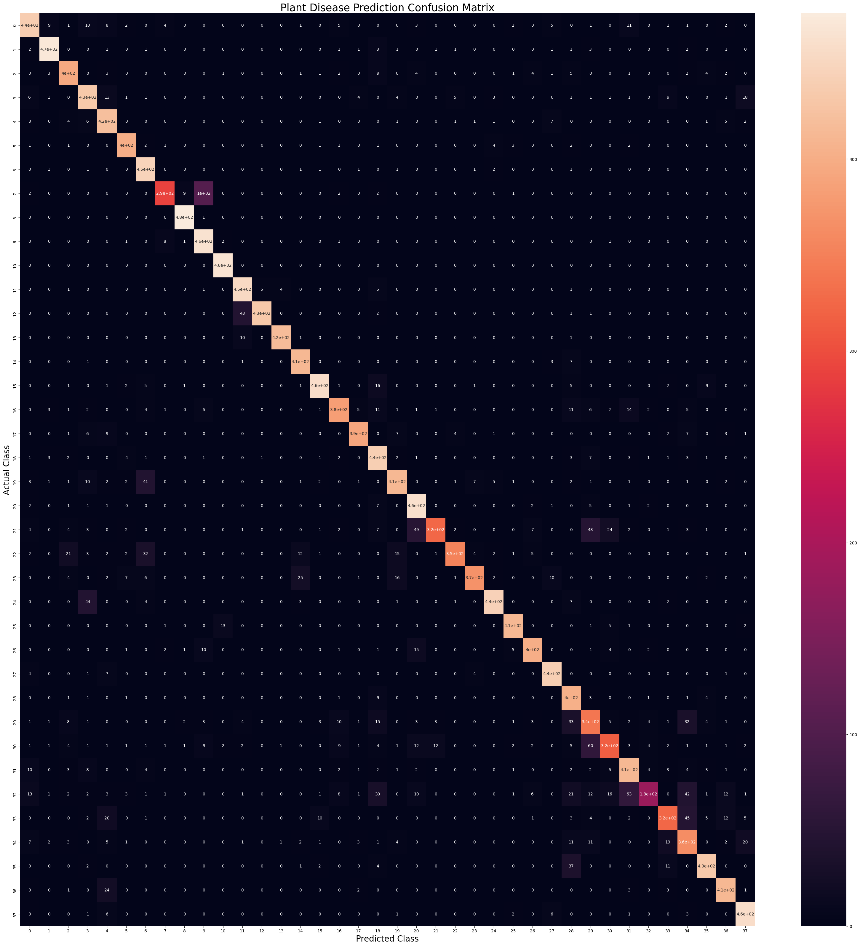


Figure 15: VGG16 Confusion Matrix

#### **5.1.2 ResNet Performance**

##### ***5.1.2.1 Accuracy and Loss***

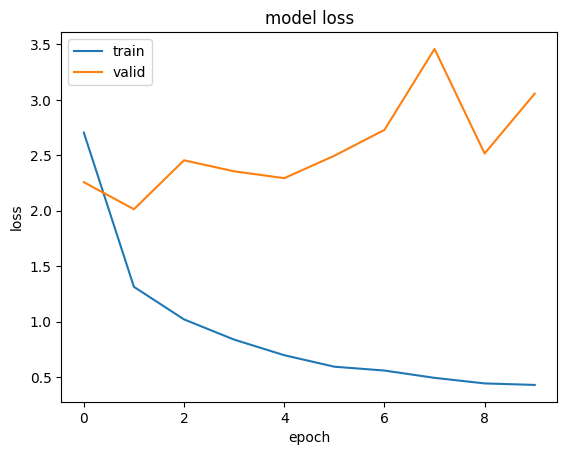


Figure 17: ResNet50 Loss for Train & Valid

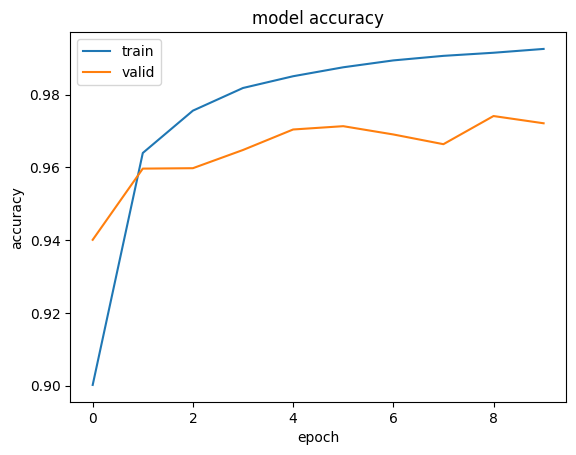


Figure 16: ResNet50 Accuracy for Train & Valid

##### ***5.1.2.2 Precision, Recall, F1-score for each Class***

##### 

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | Recall | f1-score |
| Apple scab | 1.00 | 0.97 | 0.98 |
| Apple Black rot | 0.99 | 0.99 | 0.99 |
| Apple Cedar apple rust | 0.99 | 0.99 | 0.99 |
| Apple healthy | 0.99 | 0.99 | 0.99 |
| Blueberry healthy | 0.97 | 1.00 | 0.98 |
| Cherry Powdery mildew | 0.99 | 1.00 | 0.99 |
| Cherry healthy | 1.00 | 0.99 | 0.99 |
| Corn Cercospora leaf spot gray leaf spot | 0.98 | 0.87 | 0.92 |
| Corn Common rust | 1.00 | 0.99 | 0.99 |
| Corn Northern Leaf Blight | 0.88 | 0.98 | 0.93 |
| Corn healthy | 1.00 | 1.00 | 1.00 |
| Grape Black rot | 0.90 | 1.00 | 0.95 |
| Grape Esca (Black Measles) | 1.00 | 0.90 | 0.95 |
| Grape Leaf blight (Isariopsis Leaf Spot) | 1.00 | 1.00 | 1.00 |
| Grape healthy | 1.00 | 1.00 | 1.00 |
| Orange Haunglongbing | 0.99 | 1.00 | 0.99 |
| Peach Bacterial spot | 1.00 | 0.99 | 0.99 |
| Peach healthy | 0.98 | 1.00 | 0.98 |
| Pepper, bell Bacterial spot | 0.98 | 0.98 | 0.98 |
| Pepper, bell healthy | 0.98 | 0.97 | 0.97 |
| Potato Early blight | 1.00 | 0.96 | 0.98 |
| Potato Late blight | 0.98 | 0.94 | 0.96 |
| Potato healthy | 0.96 | 0.99 | 0.98 |
| Raspberry healthy | 1.00 | 1.00 | 1.00 |
| Soybean healthy | 1.00 | 0.99 | 1.00 |
| Squash Powdery mildew | 1.00 | 0.99 | 1.00 |
| Strawberry Leaf scorch | 1.00 | 0.99 | 1.00 |
| Strawberry healthy | 1.00 | 1.00 | 1.00 |
| Tomato Bacterial spot | 1.00 | 0.96 | 0.98 |
| Tomato Early blight | 0.92 | 0.91 | 0.91 |
| Tomato Late blight | 0.92 | 0.94 | 0.93 |
| Tomato Leaf Mold | 0.96 | 0.95 | 0.96 |
| Tomato Septoria leaf spot | 0.89 | 0.97 | 0.93 |
| Tomato Spider mites Two-spotted spider mite | 0.97 | 0.87 | 0.92 |
| Tomato Target Spot | 0.90 | 0.91 | 0.91 |
| Tomato Yellow Leaf Curl Virus | 0.91 | 1.00 | 0.95 |
| Tomato mosaic virus | 0.99 | 0.99 | 0.99 |
| Tomato healthy | 0.98 | 0.98 | 0.98 |

Table 4: Precision, Recall, and F1-Score

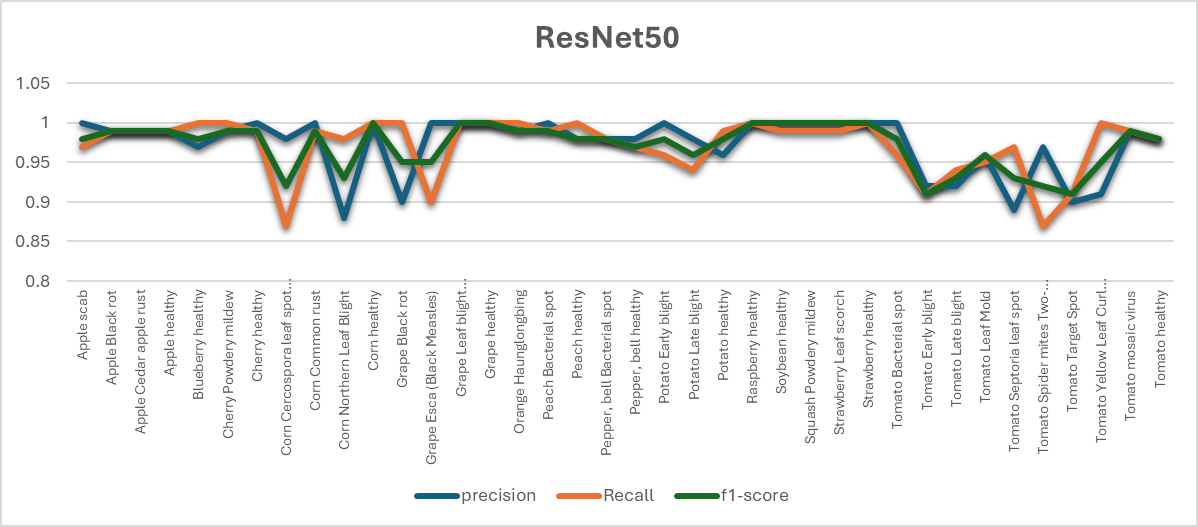


Figure 18: Precision, Recall, and F1-Score

***5.1.2.3 Confusion Matrix***

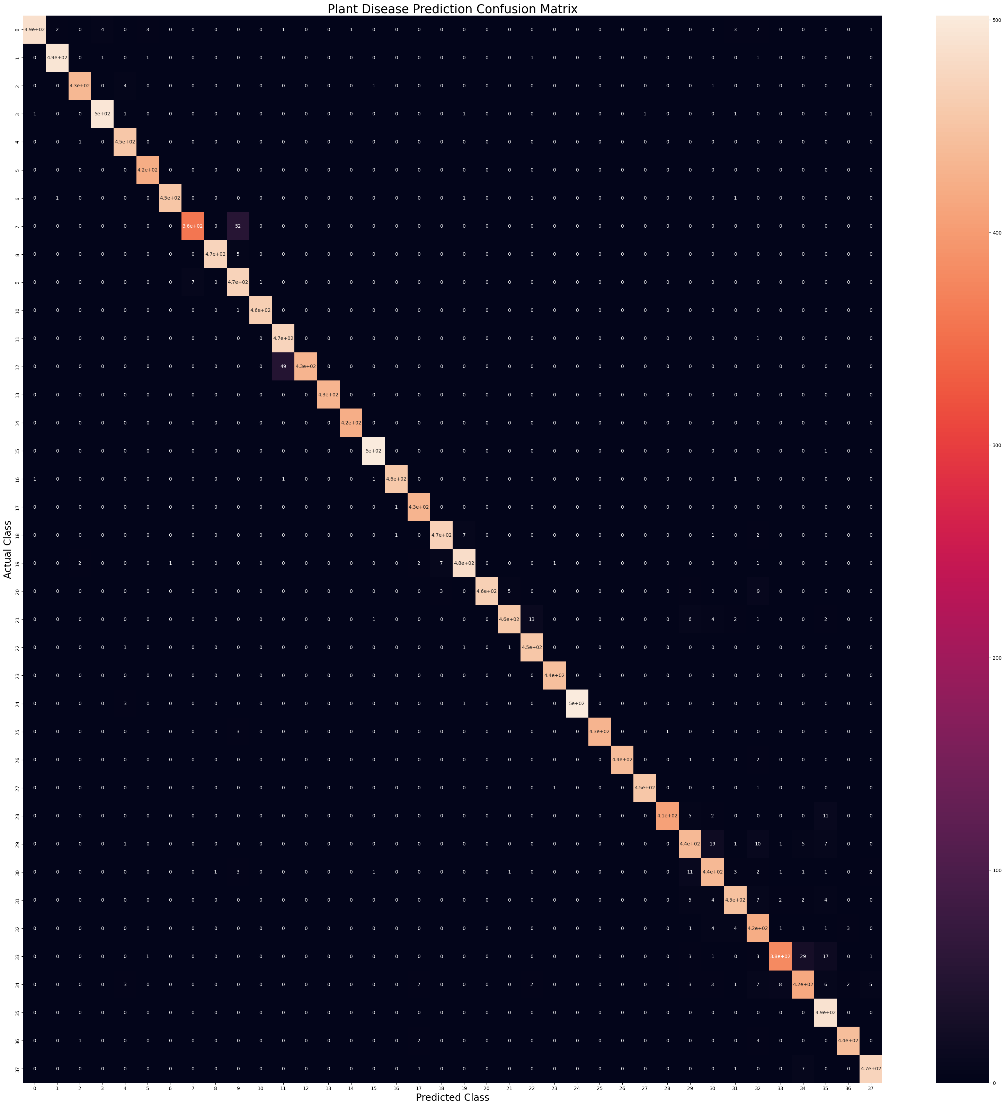


Figure 19: ResNet50 Confusion Matrix

#### **5.1.3 Custom Model Performance**

***5.1.3.1 Accuracy and Loss***

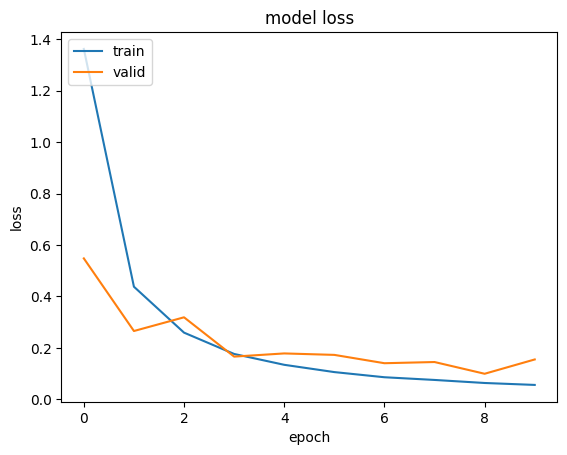


Figure 21: Custom Model Loss for Train & Valid

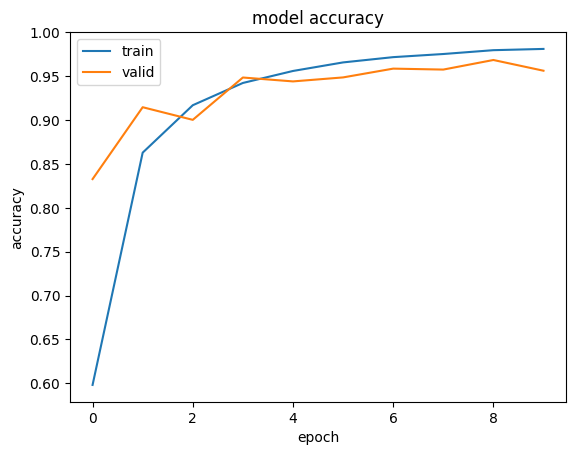


Figure 20: Custom Model Accuracy for Train & Valid

##### ***5.1.3.2 Precision, Recall, F1-score for each Class***

|  |  |  |  |
| --- | --- | --- | --- |
|  | precision | Recall | f1-score |
| Apple scab | 0.98 | 0.88 | 0.93 |
| Apple Black rot | 1.00 | 0.96 | 0.98 |
| Apple Cedar apple rust | 0.99 | 0.93 | 0.95 |
| Apple healthy | 0.94 | 0.96 | 0.95 |
| Blueberry healthy | 0.94 | 0.98 | 0.96 |
| Cherry Powdery mildew | 0.96 | 0.98 | 0.97 |
| Cherry healthy | 0.96 | 0.99 | 0.98 |
| Corn Cercospora leaf spot gray leaf spot | 0.97 | 0.87 | 0.91 |
| Corn Common rust | 1.00 | 0.99 | 1.00 |
| Corn Northern Leaf Blight | 0.95 | 0.94 | 0.94 |
| Corn healthy | 1.00 | 1.00 | 1.00 |
| Grape Black rot | 0.99 | 0.97 | 0.98 |
| Grape Esca (Black Measles) | 0.99 | 0.99 | 0.99 |
| Grape Leaf blight (Isariopsis Leaf Spot) | 0.98 | 1.00 | 0.99 |
| Grape healthy | 1.00 | 0.98 | 0.99 |
| Orange Haunglongbing | 0.96 | 0.98 | 0.97 |
| Peach Bacterial spot | 0.98 | 0.94 | 0.96 |
| Peach healthy | 0.99 | 0.98 | 0.99 |
| Pepper, bell Bacterial spot | 0.99 | 0.98 | 0.94 |
| Pepper, bell healthy | 0.94 | 0.95 | 0.95 |
| Potato Early blight | 1.00 | 0.94 | 0.97 |
| Potato Late blight | 0.93 | 0.97 | 0.95 |
| Potato healthy | 0.97 | 0.93 | 0.95 |
| Raspberry healthy | 0.90 | 0.99 | 0.94 |
| Soybean healthy | 0.99 | 0.96 | 0.97 |
| Squash Powdery mildew | 0.87 | 1.00 | 0.93 |
| Strawberry Leaf scorch | 0.98 | 0.95 | 0.96 |
| Strawberry healthy | 0.90 | 1.00 | 0.95 |
| Tomato Bacterial spot | 0.99 | 0.95 | 0.97 |
| Tomato Early blight | 0.93 | 0.92 | 0.92 |
| Tomato Late blight | 0.94 | 0.89 | 0.92 |
| Tomato Leaf Mold | 0.98 | 0.98 | 0.98 |
| Tomato Septoria leaf spot | 0.84 | 0.94 | 0.89 |
| Tomato Spider mites Two-spotted spider mite | 0.81 | 1.00 | 0.89 |
| Tomato Target Spot | 0.95 | 0.84 | 0.89 |
| Tomato Yellow Leaf Curl Virus | 0.98 | 0.98 | 0.98 |
| Tomato mosaic virus | 0.98 | 0.98 | 0.98 |
| Tomato healthy | 0.99 | 0.98 | 0.98 |

Table 5: Precision, Recall, and F1-Score

Figure 22:Precision, Recall, and F1-Score

***5.1.3.3 Confusion Matrix***

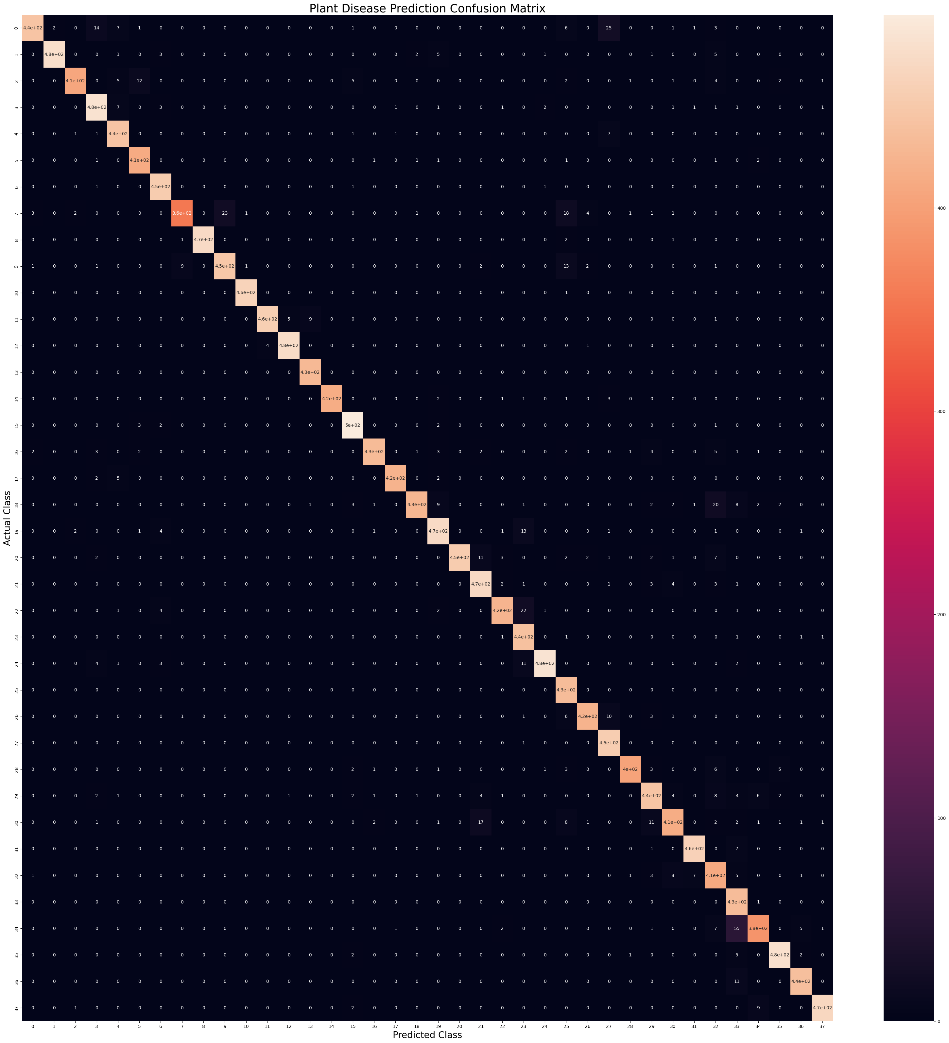


Figure 23: Custom Model Confusion Matrix

### **5.2 Discussion and Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Loss | Val\_Loss | Val\_Acc |
| VGG16 | 83% | 0.48 | 0.51 | 85% |
| ResNet50 | 99% | 0.4291 | 3.0567 | 97% |
| Custom | 98% | 0.05 | 0.15 | 95% |

Table 6: Summary of Result

As we saw previously each of the three model have a good result, but let’s have a closer look at each of them. VGG16 provided us with good and stable performance, good performance on various classes with relatively balanced precision, recall, and F1-scores. The difference between training loss (0.48) and validation loss (0.51) is minimal (0.03). This small difference indicates that the model has a good balance between fitting the training data and generalizing to the validation data, exhibits the least overfitting. This model is well-regularized and generalizes effectively to unseen data. But on the other hand, it also has a Lower overall accuracy compared to ResNet50 and Custom Model. Some classes like "Tomato Early blight" and "Tomato Late blight" have significantly lower scores.

The ResNet50 gives us high accuracy and performance across almost all classes. Superior generalization with high precision, recall, and F1-scores. But at the same time, the substantial gap between training and validation loss suggests overfitting. The model performs very well on the training data but fails to generalize equally well to the validation data. Despite high validation accuracy, the high validation loss is a critical indicator of overfitting.

The custom model provides us with everything we need. High accuracy and performance with low validation loss indicating good generalization. Balanced precision, recall, and F1-scores across most classes. The training loss is extremely low at 0.05, and the validation loss is 0.15, with a difference of 0.10. The difference between training and validation loss is modest. While there is a slight indication of overfitting, the validation loss remains reasonably low, suggesting the model maintains good generalization capabilities. The custom model achieves a good trade-off between training performance and validation generalization.

The Custom CNN model's performance can be attributed to several factors, ranging from architectural design choices to training strategies. Here, we explore these factors in detail:  
  
1. Architectural Design

* Depth and Complexity: The custom model consists of 10 convolutional layers and 5 max-pooling layers. This depth allows the model to capture complex features from the images, which is crucial for distinguishing between 38 different classes.
* Convolutional Layers: Convolutional layers are adept at capturing spatial hierarchies in the data. Having multiple convolutional layers helps the model to learn various levels of abstraction, from edges and textures in the early layers to more complex shapes and objects in the deeper layers.
* Max-Pooling Layers: Max-pooling layers help in downsampling the feature maps, reducing the computational load and the number of parameters. This also helps in making the representations invariant to small translations, enhancing generalization.
* Dropout: Dropout is a regularization technique that helps prevent overfitting. By randomly dropping units during training, dropout forces the network to learn more robust features that are not reliant on specific neurons. This contributes to better generalization on the validation set.

2. Input Image Size

* Image Size (124x124): While the custom model uses a smaller input size (124x124) compared to VGG16 and ResNet50 (224x224), this reduces computational complexity and speeds up training. It also helps in scenarios where high-resolution detail may not be as crucial, thus balancing between detail and efficiency.

3. Training and Validation Strategies

* Balanced Training and Validation Loss: The custom model achieves a training loss of 0.05 and a validation loss of 0.15. This small difference indicates effective learning without significant overfitting. This balance suggests that the model is generalizing well to unseen data, which is crucial for robust performance.
* Epochs and Early Stopping: Although trained for 10 epochs (less than VGG16), the effective use of early stopping might have ensured that the model stopped training once the performance on the validation set stopped improving, preventing overfitting.

## **Conclusion**

The custom CNN model's performance is a result of a well-designed architecture tailored to the specific task of plant disease classification. Its depth and complexity are sufficient to capture detailed features, while regularization techniques like dropout help in preventing overfitting. The smaller input size balances computational efficiency with sufficient detail extraction. Effective training strategies, including data augmentation, further enhance its generalization capabilities. Together, these factors contribute to the custom model's robust performance in classifying plant diseases.

# **References**

1. <https://www.scientificamerican.com/article/global-population-growth-is-slowing-down-heres-one-reason-why/> *(Last check 19/may)*
2. <https://humanprogress.org/trends/global-death-rate-is-falling/> *(Last check 19/may)*
3. <https://foodplantsinternational.com/> *(Last check 19/may)*
4. <https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset> *(Last check 19/may)*
5. <https://uci.edu/> *(Last check 19/may)*
6. <https://datasetsearch.research.google.com/> *(Last check 19/may)*
7. <https://pandas.pydata.org/docs/> *(Last check 19/may)*
8. <https://www.tensorflow.org/learn> *(Last check 19/may)*
9. <https://pypi.org/project/opencv-python/> *(Last check 19/may)*
10. <https://albumentations.ai/docs/images/introduction/image_augmentation/augmentation.jpg> *(Last check 19/may)*
11. Hu, B., Lei, C., Wang, D., Zhang, S., & Chen, Z. (2019). A preliminary study on data augmentation of deep learning for image classification. *arXiv preprint arXiv:1906.11887*.
12. Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., & Shen, F. (2022). Image data augmentation for deep learning: A survey. *arXiv preprint arXiv:2204.08610*.
13. Avidan, S., & Shamir, A. (2023). Seam carving for content-aware image resizing. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2* (pp. 609-617).3
14. Saponara, S., & Elhanashi, A. (2021, September). Impact of image resizing on deep learning detectors for training time and model performance. In *International Conference on Applications in Electronics Pervading Industry, Environment and Society* (pp. 10-17). Cham: Springer International Publishing.
15. Mingxing et al [13] proposed a technique for neural network design architecture for object detection and suggested several ways of optimization to enhance efficiency of neural network models.
16. Saponara, S., & Elhanashi, A. (2021, September). Impact of image resizing on deep learning detectors for training time and model performance. In *International Conference on Applications in Electronics Pervading Industry, Environment and Society* (pp. 10-17). Cham: Springer International Publishing.
17. <https://www.ibm.com/topics/convolutional-neural-networks> *(Last check 19/may)*
18. <https://lekhuyen.medium.com/an-overview-of-vgg16-and-nin-models-96e4bf398484> *(Last check 19/may)*
19. <https://medium.com/@arashserej/resnet-50-83b3ff33be7d> *(Last check 19/may)*
20. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
21. Ji, Q., Huang, J., He, W., & Sun, Y. (2019). Optimized deep convolutional neural networks for identification of macular diseases from optical coherence tomography images. *Algorithms*, *12*(3), 51.
22. <https://colab.research.google.com/> *(Last check 19/may)*