UNIVERSITY OF TECHNOLOGY
(YATANARPON CYBER CITY)
FACULTY OF ELECTRONIC ENGINEERING

IMAGE CLASSIFICATION OF GOODS AT SHOP FOR VISUALLY IMPAIRED PERSON

BY

MYAT HSU KHAING

B.E (EC) GRADUATION THESIS

FEBRUARY, 2023

UNIVERSITY OF TECHNOLOGY
(YATANARPON CYBER CITY)
FACULTY OF ELECTRONIC ENGINEERING

IMAGE CLASSIFICATION OF GOODS AT SHOP FOR VISUALLY IMPAIRED PERSON

BY

MYAT HSU KHAING

A partial dissertation submitted to the University of Technology, Yatanarpon Cyber City in fulfillment of the requirement for the degree of

Bachelor of Engineering

(Electronics)

February, 2023

DECLARATION

I hereby declare that I carried out the research work reported in this thesis in the Faculty of Electronic Engineering, University of Technology (Yatanarpon Cyber City), under the supervision of Daw Kyi Pyar Zaw. I solemnly declare that to the best of my knowledge, no part of this thesis has been submitted here or elsewhere in a previous application for award of a degree. All sources of knowledge used have been duly acknowledged.

……

8th February, 2023

MYAT HSU KHAING

6EcE-10R

APPROVAL

This is to certify that the graduation thesis entitled "**IMAGE CLASSIFICATION OF GOODS AT SHOP FOR VISUALLY IMPAIRED PERSON**" carried out by **(MA MYAT HSU KHAING**, **6EcE-10R)** has been read and approved for meeting part of the requirements and the regulations governing the award of the Bachelor of Engineering (Electronics), Faculty of Electronic Engineering, degree of University of Technology (Yatanarpon Cyber City), Myanmar.

1. Dr. Atar Mon
   Professor and Head
   Faculty of Electronic Engineering ……
   (Chairman)

2. Daw Kyi Pyar Zaw
   Lecturer
   Faculty of Electronic Engineering ……
   (Supervisor)

3. Dr.Atar Mon
   Professor and Head
   Faculty of Electronic Engineering ……
   (Co-Supervisor)

4. Daw Hnin Pwint Phyu
   Lecturer
   Faculty of Electronic Engineering ……
   (Member)

5. Daw Aye Theingi Oo
   Lecturer
   Faculty of Electronic Engineering …….
   (Member)

ACKNOWLEDGEMENTS

First of all, I would like to express my special thanks to Dr. Aung San Lin, Rector (Acting), University of Technology (Yatanarpon Cyber City), for initiating the Bachelor programme at the University of Technology (Yatanarpon Cyber City).

I would like to express my gratitude to Dr. Thu Zar Aung, Pro-rector, University of Technology (Yatanarpon Cyber City), for her kind permission to complete this graduation thesis. I am thankful to her for giving enough consideration, ideas and views.

I would like to express my grateful thanks to Dr. Atar Mon, Professor and Head of Faculty of Electronic Engineering, University of Technology (Yatanarpon Cyber City), for her kind guidance, encouragement and supervision and patience in making my thesis to complete successfully.

I am very grateful to my supervisor, Daw Kyi Pyar Zaw, Lecturer, Faculty of Electronic Engineering, University of Technology (Yatanarpon Cyber City), for her kind guidance and encouragement. She has been very supportive in this graduation thesis, and also guides a lot, particularly, at the level of quality of presentation and technical knowledge sharing and guiding in this thesis.

I would like to give special thanks to Daw Hnin Nu Wai, Associate Professor, English Department, University of Technology (Yatanarpon Cyber City) for her valuable supports from the language point of view in my graduation thesis work.

I would like to thank a lot to Daw Hnin Pwint Phyu and Daw Aye Theingi Oo for their mentoring, encouragement, and recommending this dissertation.

Finally, I am grateful to my parents and senior friend, Htin Aung Lu who specially offered strong technical, moral and physical support, care and kindness, during the year of my graduation thesis study.

ABSTRACT

In the current digital era, AI has become an emerging trend and this powerful technology has transformed ordinary operations into intelligent automation process. Computer vision is a branch of AI that allows to extract useful information from photos, videos and other visual inputs. The proposed system intends to classify thirty different goods using Convolutional Neural Network (CNN). This system will be helpful for visually impaired person in shopping. There are two stages to implement the system. The first one is "Implementation of Classifier Model" using Convolutional Neural Network and the second is "Deploying the Classifier Model with Webcam" and the classified result will be transformed from text to speech. The system is implemented by using PyTorch framework. Then VGG-16 CNN model is applied for image classification and a 1080p webcam is used for image acquisition. The good images dataset is taken from supermarket and Google images. The classifier model is trained on a dataset that includes 4000 images of goods. The overall accuracy of the system is approximately 88% in real-time testing with 1080p webcam and this depends on the resolution of the camera. The system is trained with 0.001 learning rate by 30 epochs. In summary, there are many people who are visually impaired in the world and this system intends for them to be more independent and to feel like a normal person in their daily lives. AI has been widely used in the developed countries instead of workers in many places. Among the useful applications of AI, applying AI to help a disable person is the most beautiful part of AI.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

Nowadays, people are applying many computer vision and deep learning applied systems in various ways such as movie recommendation, auto reply chat bots, face recognition system, malware checking system and human aid applications. Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised. Machine learning algorithms almost always require structured data, while deep learning networks rely on layers of Artificial Neural Network (ANN). Moreover, the help of technology especially for impaired person is the most beautiful part of technology. As a technology terms, computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual can do.

There are many people who are visually impaired in the world. They cannot see and read anything. They usually guess what the thing is by their sense. This system is to develop a deep learning classifier model to help the visually impaired person in shopping. This system helps the user to know easily the name of good that is input to the camera especially which are not easy to guess by the sense such as plastic packed vegetables, meat, etc. Being the proposed system is part of computer vision technology, this system use Webcam for good capturing and a custom Deep Convolutional Neural Network (CNN) based model VGG-16 is used for classification process. By considering the needs of visually impaired individuals and other marginalized communities, technology can be created that is not only innovative and effective, but also inclusive and empowering for all individuals.

Image processing is a type of signal processing in which input is an image and output may be image or characteristics and features associated with that image. Image processing services combine advanced algorithmic technology with machine learning and computer vision to process large volumes of pictures easily and quickly. Some of the important applications of image processing in the field of science and technology include computer vision, remote sensing, face detection, recognition, medical image

analysis, image search: to search for similar images, which can be used in search engines, e-commerce platforms, and social media, agriculture: to classify and identify crops, pests, and weeds, quality control: to detect defects and anomalies in manufacturing processes. Overall, image classifiers are an essential tool for visual recognition tasks and are widely used across many industries and domains.

1.1 Aim and Objectives

The main aim of the system is:

- To assist visually impaired person in shopping for classification of Goods in Supermarkets

The objectives are defined as follows:

- To learn Deep Learning and Artificial Neural Network
- To build an image classifier model with PyTorch
- To train the model for classification of thirty different of goods
- To develop a command line application for system implementation
- To deploy the classifier model with webcam in local machine
- To transform the classified result text to speech

1.2 Overview of the System

The main task of the system is to classify the input goods in real-time. In this system, a custom architecture CNN model is used to classify the goods and for image feature extraction pretrained, VGG-16 convolutional base layer is used. The system works by taking an input image and passing it through the VGG16 model to extract features at different levels of abstraction. The output of the model is then passed through a fully connected layer and a Softmax activation function to generate a probability distribution over the different classes of goods. The dataset for this system only contains thirty different classes of good from the supermarket and some are collected from Google. So, the system shows-one of these goods. The goods in the dataset are as follows: apple, cherry tomato, grape, chilli, pumpkin, watermelon, mixed fruit, jack fruit, bitter gourd, chicken wings, chicken drumette, cabbage, dragon fruit, strawberry, etc. For the training dataset, there are 100 images for each class and 20 images for each of test set. Totally, there are 4000 images in the dataset and with the image dimensions of 224x224. The system to classify different goods with

VGG16 model is a powerful computer vision application that can help visually impaired person at the identification and tracking of goods in shopping.

1.3 Scope of Thesis

Image classifier model is implemented as below:

- Collecting thirty different good images dataset from market and online
- Image classifier model is built and trained with training dataset to the minimum error state by PyTorch framework
- Class prediction process take places by the trained model
- 8.1m 1080p webcam is used for input good images
- Command line application is developed with two usage: train.py & predict.py
- Create a suitable scripting environment for prediction in local machine
- The most likelihood predicted goods are shown as a result
- Predicted top one class is output by speech

1.4 Procedure of Thesis

This thesis is implemented according to the following programs:

- Learning theoretical background required for the system such as Deep learning and Artificial Neural Network
- Collecting the dataset for training, validation and testing processes
- Building suitable neural network architecture design for classifier model with different numbers of hidden layers
- Training the classifier model with suitable epochs for the most accurate result
- Evaluating the model with validation dataset to be able to estimate the performance of the trained model, calculating metrics such as accuracy, precision, recall
- Testing the trained model to classify the goods by accessing the images from test dataset
- Setting up the classifier model in command line terminal of local machine with an environment that is created in Conda
- Testing the model in real-time with webcam and the result is output with speech

1.5 Outline of Thesis

The image classification of goods system is analyzed according to the following chapter plans.

- ❖ Chapter 1 introduces the introduction and overview of the classifier system.
- ❖ Chapter 2 describes the theoretical background related to this proposed system with necessary block diagrams.
- ❖ Chapter 3 demonstrates system design and implementation.
- ❖ Chapter 4 includes testing results with sample tables for good classification.
- ❖ Chapter 5 concludes the summary, limitations, recommendations and further extensions for this system.

CHAPTER 2
THEORETICAL BACKGROUND

In this chapter, theoretical background of the proposed system will be discussed. This system is intended to capture the goods by using Webcam and to predict their names by using Convolutional Neural Network. The different types of Convolutional Neural Network architectures are explained in this chapter. This chapter also discusses about Convolutional Neural Network layers, tensors and the related coding implementations for theoretical background.

2.1 History of Convolutional Neural Network

Convolutional Neural Networks (CNNs) have a relatively short but influential history in the field of artificial intelligence and machine learning. In [5], David Hubel and Torsten Wiesel described “simple cells” and “complex cells” in the human visual cortex. The authors proposed that both kinds of cells are used in pattern recognition. A “simple cell” responds to edges and bars of particular orientations. Figure 2.1 shows the example of sample cell.

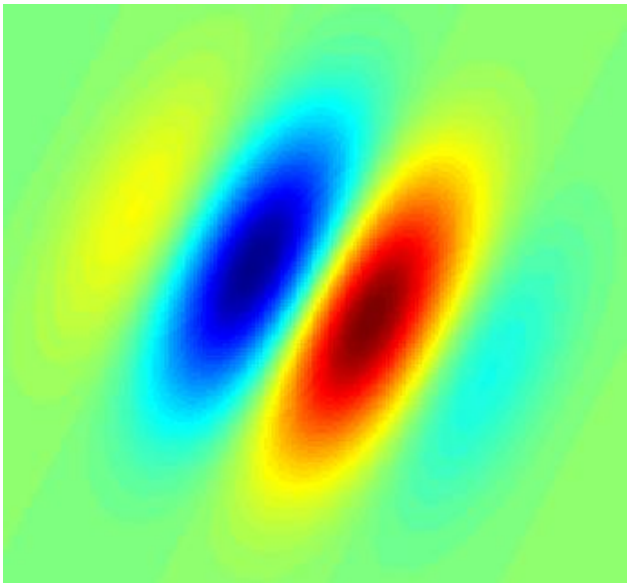


Figure 2.1 Example of Sample Cell

A “complex cell” also responds to edges and bars of particular orientations, but it is different from a simple cell in that these edges and bars can be shifted around the scene and the cell will still respond. For instance, a simple cell may respond only

to a horizontal bar at the bottom of an image, while a complex cell might respond to horizontal bars at the bottom, middle, or top of an image. This property of complete cell is termed as "spatial invariance". Hubel and Wiesel proposed in [5] that complex cells achieve spatial invariance by "summing" the output of several simple cells that all prefer the same orientation (e.g., horizontal bars) but different receptive fields (e.g., bottom, middle, or top of an image). By collecting information from a bunch of simple cell minions, the complex cells can respond to horizontal bars that occur anywhere. This concept – those simple detectors can be "summed" to create more complex detectors – is found throughout the human visual system, and is also the fundamental basis of convolution neural network models.

In the [6], Dr. Kunihiko Fukushima was inspired by Hubel and Wiesel's work on simple and complex cells [5], and proposed the "neocognitron" model. The neocognitron model includes components termed "S-cells" and "C-cells." These are not biological cells, but rather mathematical operations. The "S-cells" sit in the first layer of the model, and are connected to "C-cells" which sit in the second layer of the model. The overall idea is to capture the "simple-to-complex" concept and turn it into a computational model for visual pattern recognition.

Convolutional Neural Networks (CNNs) have become popular especially in computer vision in the last few years because of outstanding performance on different tasks, such as image classifications. Convolutional neural networks are a category of Neural Networks that have been proven very effective in areas such as image recognition and classification. CNN have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.

The first work on modern convolutional neural networks (CNNs) occurred in the [12], inspired by the neocognitron. Yann LeCun et al., in their paper "GradientBased Learning Applied to Document Recognition" demonstrated that a CNN model which aggregates simpler features into progressively more complicated features can be successfully used for handwritten character recognition. Specifically, LeCun trained a CNN using the MNIST database of handwritten digits (MNIST pronounced "EMnisst"). MNIST is a now-famous data set that includes images of handwritten digits paired with their true label of 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Sample of MNIST dataset is shown in Figure 2.2. A CNN model is trained on MNIST by giving it an example image, asking it to predict what digit is shown in the image, and then updating the model settings based on whether it predicted the digit identity correctly

or not. State-of-the-art CNN models can today achieve near-perfect accuracy on MNIST digit classification.

Figure 2.2 Example of MNIST Dataset

2.2 Convolutional Neural Networks (CNNs)

In around 1960 [5], Hubel and Wiesel designed the Convolutional Neural Network (CNN) architecture for visual cortexes recognition contain neurons that can individually respond to small regions of the visual field contents. Actually, Convolutional Neural Network (CNN) is similar with Artificial Neural Networks (ANN) but the architecture of CNN contains convolution operation that can more extract visual features than ANN. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters and characteristics.

In classification problem, the structure of a convolutional neural network is the series of computational blocks (or layers) stacking together and ending with a classifier, that makes a labeled class prediction about an input image. Typical layers are: Convolution, Activation function, Pooling, Normalization, and Classifier. All these computational layers simply take inspiration to the functionality of the visual cortex in human brain. Figure 2.3 shows the sequence architecture of a CNN model.

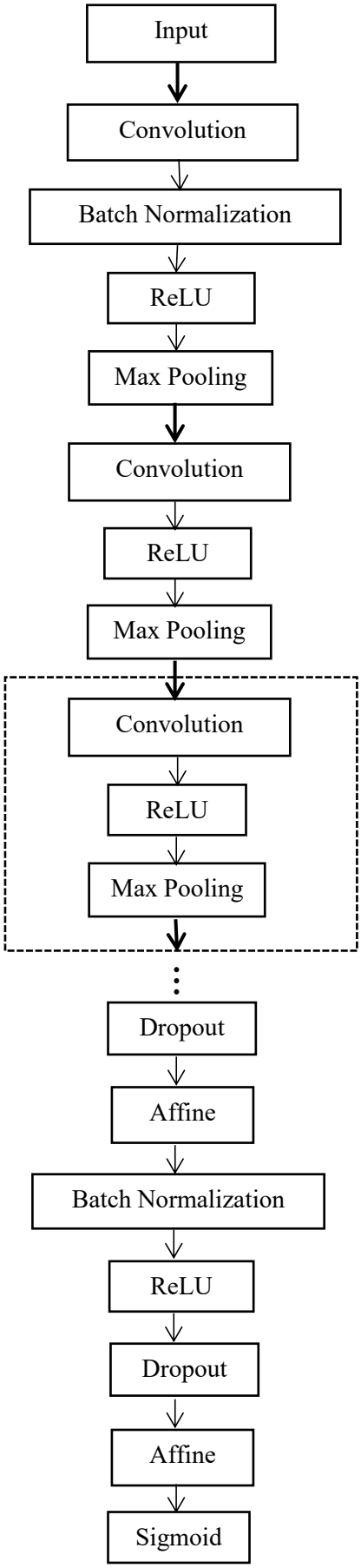


Figure 2.3 A CNN Model Sequence Architecture

The input layer receives the input image and passes it to the next layer. Convolutional layer applies a set of learnable filters or kernels to the input image to extract features. Each filter detects specific patterns or features in the input image, such as edges or corners. The output of this layer is a feature map that highlights the presence of these patterns in the input image. Activation layer applies a non-linear activation function to the output of the convolutional layer to introduce non-linearity and improve the model's ability to learn complex relationships in the data. Pooling layer reduces the spatial dimensions of the feature maps generated by the convolutional layer, helping to reduce the computational cost of the model and prevent overfitting. Dropout layer randomly drops out a fraction of the neurons in the previous layer during training, which helps to prevent overfitting. The purpose of the affine layer is to apply a learned linear transformation to the input data, allowing the network to learn non-linear relationships between the features. Specifically, the weights in the affine layer are learned during training via backpropagation and gradient descent, optimizing the network to minimize a loss function. Convolutional Neural Networks (CNNs) [10] are popular especially in computer vision applications such as face recognition, pose estimation, object detection, image recognition.

The networks are built from individual parts approximating neurons, typically called units or simply "neurons." Each unit has some number of weighted inputs. These weighted inputs are summed together (a linear combination) then passed through an activation function to get the unit output.

2.3 Neuron Model

Biological neurons are the basic unit of the nervous system and are responsible for processing and transmitting information throughout the body. It can receive and propagate signals from its connections. They consist of three main parts: the dendrites, the cell body, and the axon. In particular, a neuron presents many input connections (dendrites) and a single output (axon) that can be connected to other neurons or to biological tissue. The dendrites receive signals from other neurons, which are then processed in the cell body and transmitted down the axon to other neurons. The strength of the signal depends on the number and frequency of incoming signals. The signals propagate from one neuron cell to another via synaptic process. The ensemble of cells connection from a neural network. Synaptic signals may be excitatory or

inhibitory. If the input excitations are large enough, an impulse signal, is also called action potential, activate the neuron cell and propagates through the axon. By inspiring a biological neuron from the nervous system, a neuron or perceptron is designed [7]. Figure 2.4 shows the sample biological neuron in human brain system.

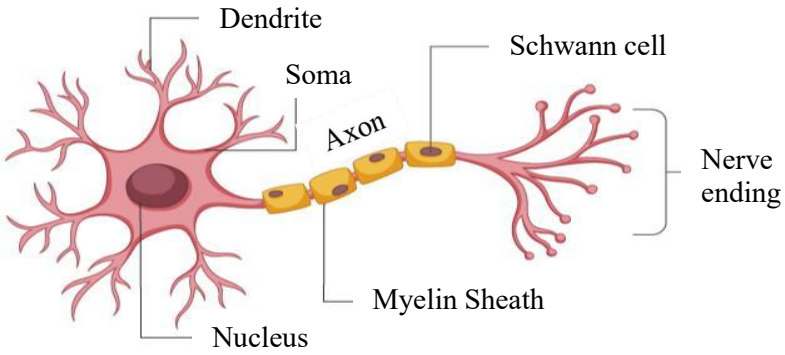


Figure 2.4 Biological Neuron

Perceptrons, on the other hand, are a simplified model of a biological neuron used in artificial neural networks. The perceptrons or neurons is the basic building block for neural network [7]. A perceptron can be viewed as a cell with many inputs and a single output. Each input has specific signal called weights. When the sum of inputs reaches some threshold of the activation function, the output of the neuron is active and can propagate to other neurons. So, a single perceptron work is the sum of products with different weights, apply bias, apply activation function for non-linearity and product output or propagate the result to another perceptron as input. The bias input is like an offset, and can be present or not in a neural network model. The designed and inspired perceptron is shown in Figure 2.5.

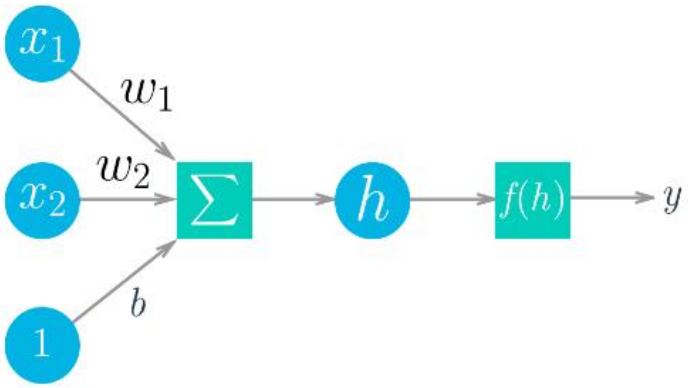


Figure 2.5 Designed and Inspired Perceptron

In Mathematical Expression :

$$h = w_1 x_1 + w_2 x_2 + b \tag{2.1}$$

The inputs $x_i$of a neuron are combined using a weighted linear combination,

$$h = \sum_{i=1}^{n} (w_i * x_i + b) \tag{2.2}$$

where n is number of inputs, x_i is i^{th} x input. With vectors this is the dot-inner product of two vectors:

$$h = [x_1 x_2 \dots x_n] \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{2.3}$$

The result is then modified by a non-linear activation function φ that generates the output y,

$$y = \varphi(g(x)) \tag{2.4}$$

φ is activation function and φ can be tanh, sigmoid, ReLU, or other activation functions. Figure 2.6 shows the different activation functions.

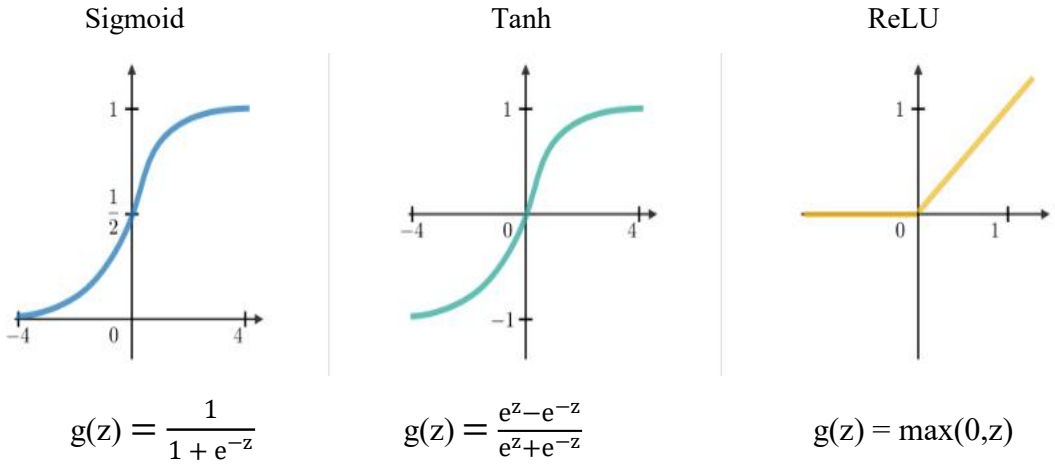
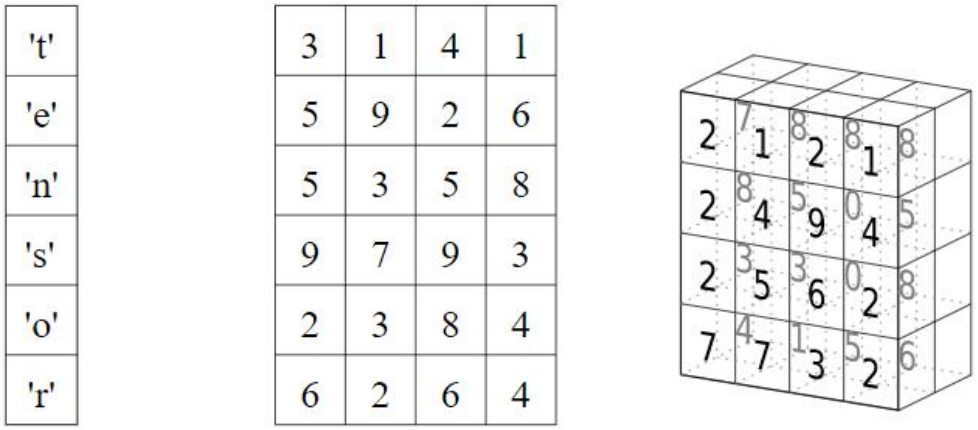


Figure 2.6 Different Activation Functions

The difference between biological neurons and perceptrons is that biological neurons are much more complex and dynamic than perceptrons. Biological neurons can change the strength of their connections over time through a process called synaptic plasticity, which allows them to learn and adapt to changing environments. Perceptrons, on the other hand, have fixed weights that are set during training and do not change during inference. Biological neurons can perform a wide range of complex computations, while perceptrons are limited to simple linear functions. Overall, while biological neurons and perceptrons are related to the field of neural networks, they have important differences in terms of their complexity, dynamics, and computational capabilities. Perceptrons are simple but still useful type of neural network. While they may not be suitable for more complex tasks, they can still be valuable in a range of applications, particularly those where speed and real-time predictions.

2.4 Tensors

Neural network computations are just a bunch of linear algebra operations on tensors, a generalization of matrices. A vector is a 1-dimensional tensor, a matrix is a 2-dimensional tensor, an array with three indices is a 3-dimensional tensor (RGB color images for example). The fundamental data structure for neural networks are tensors and PyTorch (as well as pretty much every other deep learning framework) is built around tensors.



Tensor Dimension [6,1] Tensor Dimension [6,4] Tensor Dimension [4,4,2]

Figure 2.7 Basic Diagram of Tensor

2.5 Multilayer Perceptron (MLP)

A multi-layer perceptron (MLP) is a type of artificial neural network that consists of multiple layers of neurons. MLP is a class of feed-forward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to any feed-forward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptron. The real power of this algorithm happens while stacking these individual units into layers and then the stacks of layers are added into a network of neurons. The input layer of an MLP receives input data, which is then passed through the next hidden layers, where the data is processed and transformed before producing an output. The output layer produces the final output of the network, which can be a classification, regression, or other type of prediction. The output of one layer of neurons becomes the input for the next layer. The hidden layers of an MLP allow it to model more complex functions than a perceptron, making it capable of handling more

sophisticated tasks such as image and speech recognition. The number of hidden layers and the number of neurons in each layer are hyper parameters that can be tuned to optimize the performance of the network.

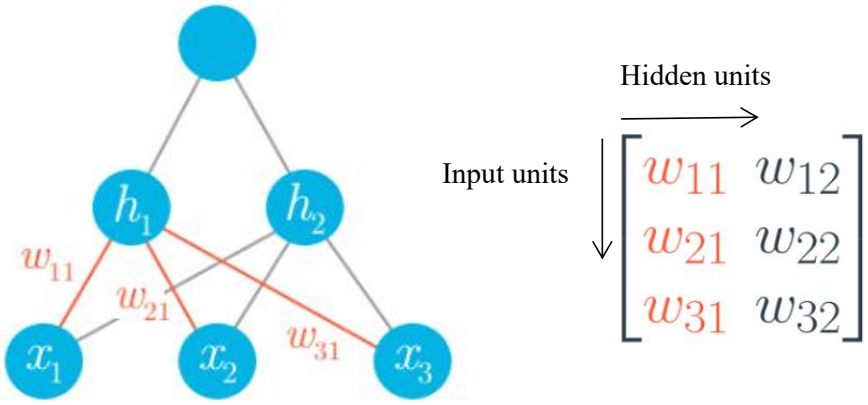


Figure 2.8 Calculation of the Output of a Neuron

The first layer shown on the bottom here are the inputs, understandably called the input layer. The middle layer is called the hidden layer, and the final layer (on the right) is the output layer. It can express that this network in mathematically with matrices again and use the matrix multiplication to get linear combinations for each unit in one operation. For example, the hidden layer (h_1& h_2) here) can be calculated as follow:

$$\vec{h} = [h_1 h_2] = [x_1 x_2 \ldots x_n]. \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ \vdots & \vdots \\ w_{n_1} & w_{n_2} \end{bmatrix} \tag{2.5}$$

The output for this small network is found by treating the hidden layer as inputs for the output unit. The network output is expressed simply as:

$$y = f_2(f_1(\vec{x}\, W_1)W_2) \tag{2.6}$$

Multilayer Perceptron (MLP) is used to apply in computer vision, now succeeded by Convolutional Neural Network (CNN). MLP is now deemed insufficient for modern advanced computer vision tasks. The training of an MLP involves adjusting the weights of the connections between the neurons to minimize a cost function, using an optimization algorithm such as back-propagation. During training, the network learns to recognize patterns and generalizes to new data by adjusting the weights in a way that minimizes the error between the predicted output and the actual output. Disadvantage is that the number of total parameters can grow to

very high. One limitation of MLPs is that they may overfit the training data which means that they may perform well on the training data but poorly on new, unseen data. This can be addressed through regularization techniques such as dropout, which randomly drops out some of the neurons during training to prevent overfitting. The sample multilayer perceptron is shown in Figure 2.9 and the number of perceptron in layer 1 is multiplied by number of perceptron in layer 2, so total parameters are grow and it costs a lot of computation power. This is inefficient because there is redundancy in such high dimensions. Another disadvantage is that it disregards spatial information. It takes flattened vectors as inputs.

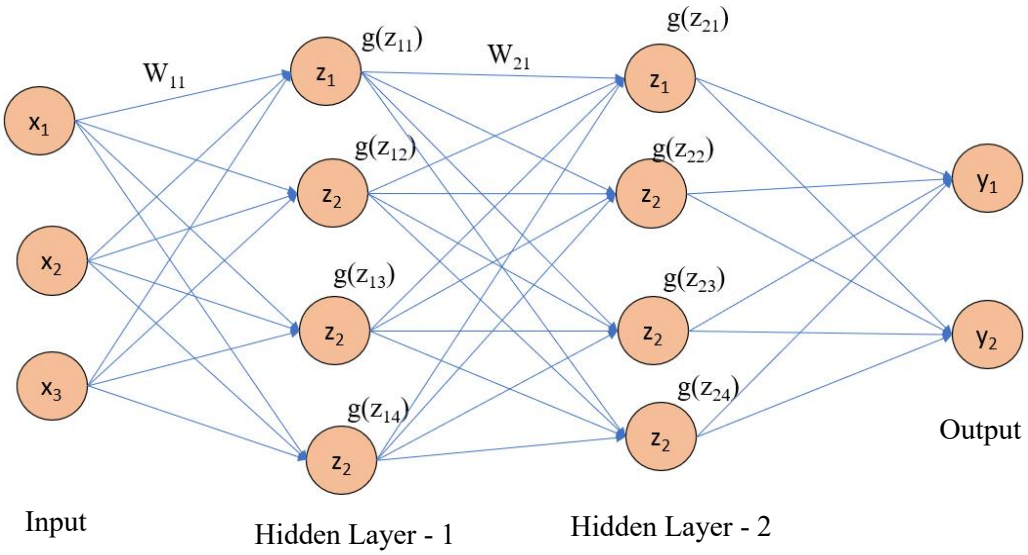


Figure 2.9 Simple Multilayer Perceptron Model

On the other hand, CNN is the incumbent, current most popular of computer vision algorithms, winner of multiple ImageNet competitions. It can account for local connectivity (each filter is panned around the entire image according to certain size and stride, allows the filter to find and match patterns no matter where the pattern is located in a given image). The weights are smaller, and shared less wasteful, easier to train than MLP. More effective too. Can also go deeper. Layers are sparsely connected rather than fully connected. It takes matrices as well as vectors as inputs. The layers are sparsely connected or partially connected rather than fully connected. Every node does not connect to every other node.

The panning of filters in CNN [10] essentially allows parameter sharing, weight sharing so that the filter looks for a specific pattern, and is location invariant can find the pattern anywhere in an image. This is very useful for object detection.

Patterns can be discovered in more than one part of the image. Simple Convolutional Neural Network is shown in Figure 2.10.

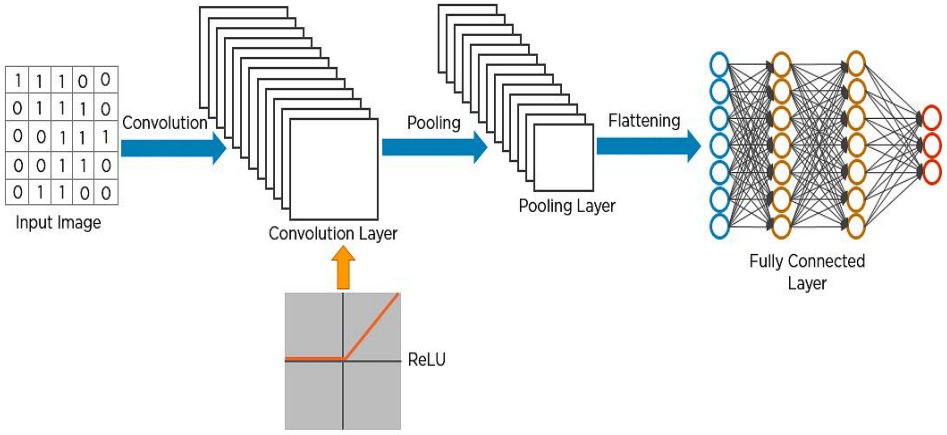


Figure 2.10 Simple Convolutional Neural Network Model

2.5.1 Convolutional Layer

Convolutional neural networks apply a filter to an input to create a feature map that summarizes the presence of detected features in the input. The convolutional operation is the basic function that brought to the development of the convolutional neural network. It works on not only black and white image but also RGB (Red, Green, Blue) image [10]. Not like MLP, convolution is not vectorizing an image. The convolution operation is a sum of dot product in space between an input volume and a kernel, made by a cubic matrix of weights for RGB image, that slides along the input volume to produce an output feature map. The kernel works like a filter of the input and extracts some visual information f (w, x) from the image. Figure 2.11 shows simple convolution operation.

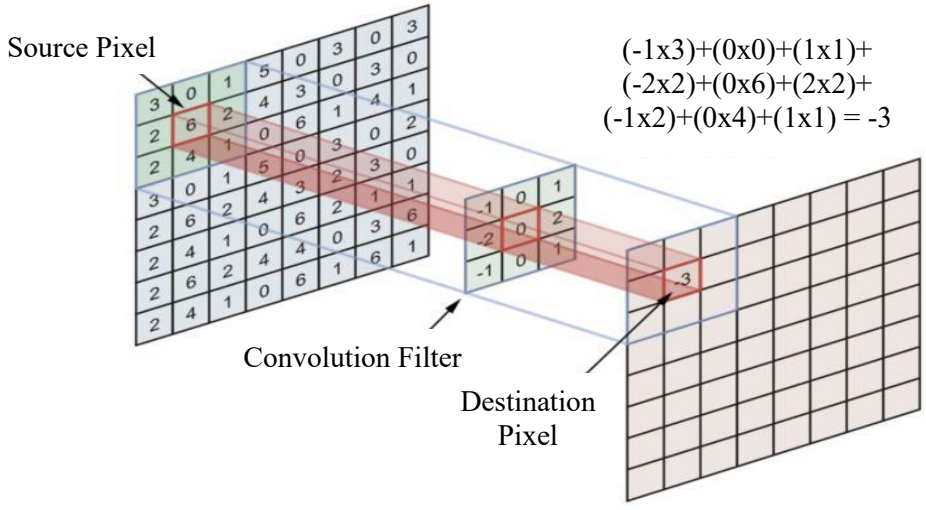


Figure 2.11 Simple Convolution Operation

The kernel is defined as K*K size and that kernel is used for all image channel for an RGB image, the colour depth is 3 so the kernel size will be (K*K*3). The weights present in a neural network are the parameters to be learned, and, at the end of a training process, these learned parameters will be characterized the network for a computer vision problem for a specific training database. In convolutional neural network the values inside the kernel are the weights(w) that filter the input simply multiplying them with the respective value of the image. The values of the input image have to be set in a range of 8 bit (0-255, or zero mean -128, +127). In the convolution process, input image values are multiplied with the kernel weights, sun the multiplied result and produce as output feature map. In Figure 2.12, (6*6) input image is multiplied with the weight kernel and result feature map is produced.

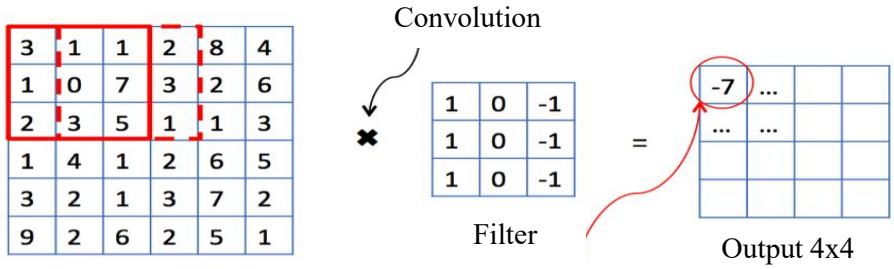


Figure 2.12 Simple of Element-Wise Product

In Mathematical expression :

$$\boldsymbol{f(w,x) = \sum_{i=1}^{k} \sum_{j=1}^{k} (w_{ij} * x_{i+p,j+q})} \qquad (2.7)$$

where w is K*K kernel

K*K is the kernel size.

p is row index of input image (p^{th} row of an image).

q is column index of input image (q^{th} column of an image).

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving a network that has a wholesome understanding of images in the dataset, similar to how

a human vision would. In the convolution process, there are three parameters to tune to get the effective model, they are stride, padding and kernel dimension.The stride determines how many positions the kernel shifts along the image in horizontal and in vertical way. Typical values are [1 1], [2 2], [4 4]. There are two types of results to the operation, one in which the convoluted feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in the case of the former, or Same Padding in the case of the latter.

Padding means adding zero values on the sides of an image to cover the losing information from corners of the input image. As an example, when kernel size is smaller than the input image size, the output feature map will be smaller than the input image size and the information from the corners can be lost.

To cover this problem, adding the zero values on the size of an image is a solution. Figure 2.13 shows sample zero padding to get same output size.

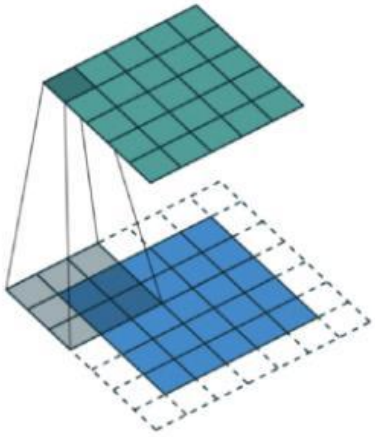


Figure 2.13 Sample Zero Padding

The kernel dimension is also need to choose in design CNN architecture. But there are some commonly use kernel size are (3*3), (5*5) and (7*7). But in this CNN architecture, only use (3*3) kernel size is used.

2.5.2 Pooling Layer

A pooling layer is an essential component of a Convolutional Neural Network (CNN) architecture. The main function of a pooling layer is to reduce the spatial size of the input feature map, while preserving the important information. It is used in convolutional neural network to extract clear parts of the feature map. Pooling layer usually follows by the convolutional layers. The pooling layer is typically applied

after the convolutional layer and activation function, and its main purpose is to reduce the computation and memory requirements of the network, and to introduce some amount of spatial invariance to the network. The spatial invariance means that the network can still recognize the same pattern, even if it is shifted slightly in the image. There are three types of pooling function max pooling, average pooling and min pooling. Figure 2.14 shows average pooling, max pooling and min pooling. But in deep learning models, most are used max pooling. Max pooling is extracting the maximum value for the pooling kernel and average pooling is getting average of the values in kernel. Max pooling sample is shown in Figure 2.15.

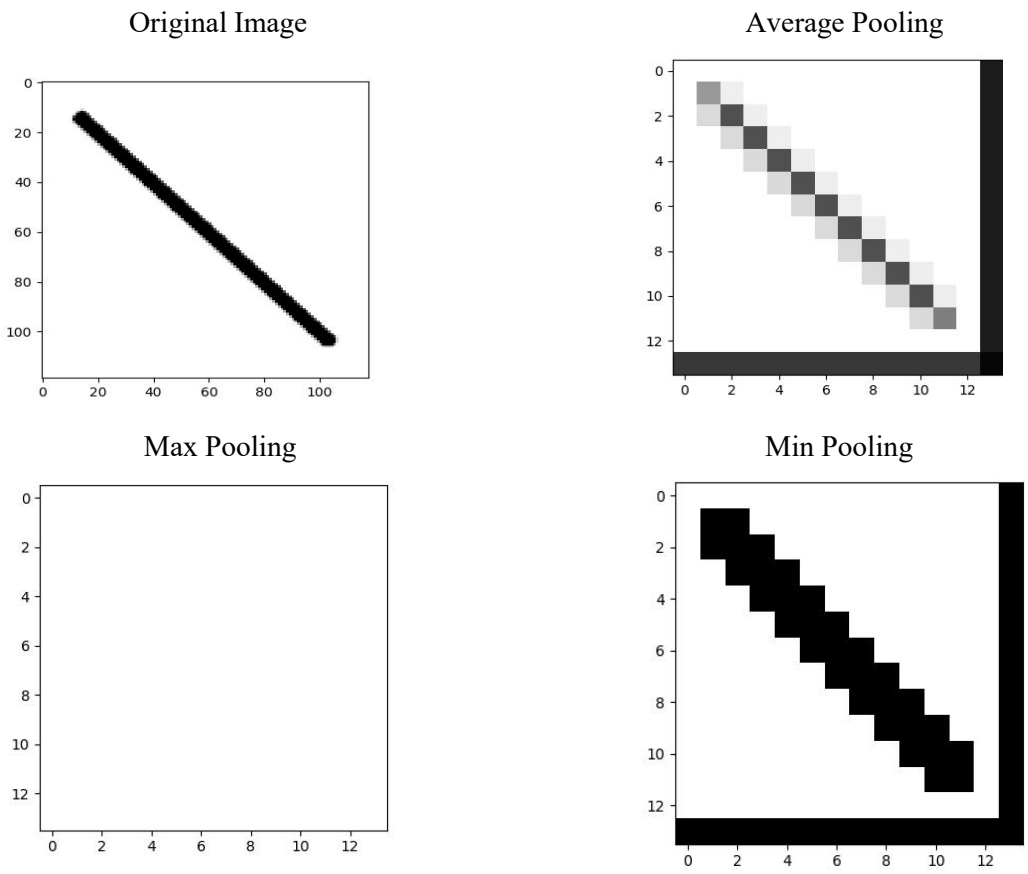


Figure 2.14 Average Pooling, Max Pooling and Min Pooling

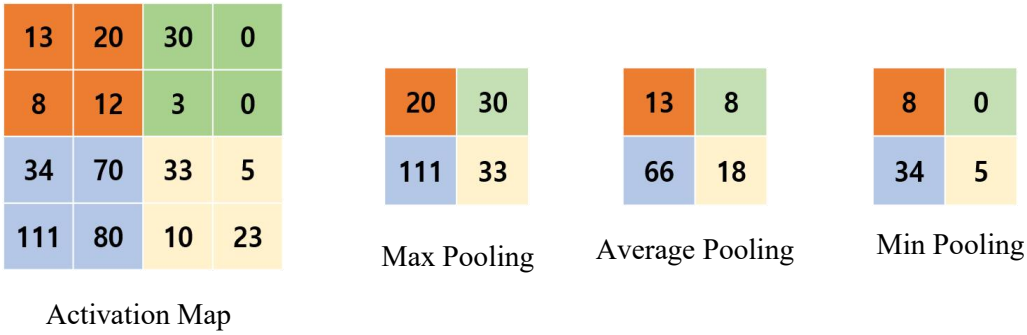


Figure 2.15 Max Pooling, Average Pooling, Min Pooling Sample

2.5.3 Flatten Layer

Flatten layer is converting the feature map data into a 1-dimensional array for inputting it to the next layer. It flatting the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer. In other words, it put all the pixel data in one line and make connections with the fully-connected layers to make classification. Sample of flatten layer is shown in Figure 2.16.

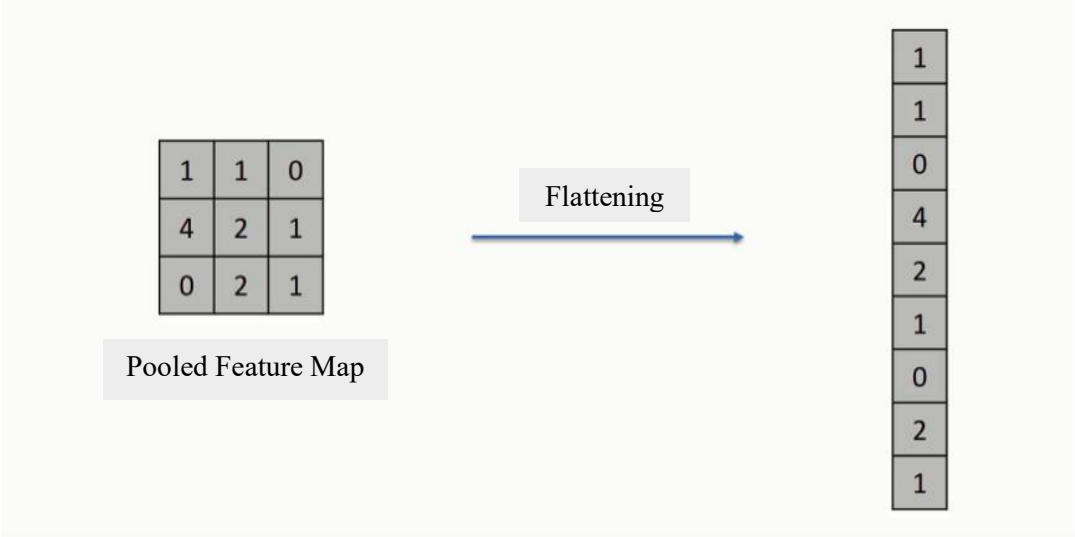


Figure 2.16 Sample of Flatten Layer

2.5.4 Fully-Connected Layer

Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the convolutional network. In fully-connected or dense networks, each unit in one layer is connected to each unit in the next layer. In fully-connected networks, the input to each layer must be a one-dimensional vector (which can be stacked into a 2D tensor as a batch of multiple examples). However, our images are 28x28 2D tensors, so we need to convert them into 1D vectors. Thinking about sizes, we need to convert the batch of images with shape (64, 1, 28, 28) to a have a shape of (64, 784), 784 is 28 times 28. This is typically called flattening, we flattened the 2D images into 1D vectors. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer. The CNN process begins with convolution and pooling, breaking down the image into features, and analyzing them independently. After passing through the fully connected layers, the

final layer uses the SoftMax activation function (instead of ReLU) which is used to get probabilities of the input being in a particular class (classification).

2.5.5 SoftMax Layer

The softmax function [4] is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1. The softmax function is sometimes called the softargmax function, or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used for multiclass classification, and its formula is very similar to the sigmoid function which is used for logistic regression. The softmax function can be used in a classifier only when the classes are mutually exclusive. Many multi-layer neural networks end in a penultimate layer which outputs real-valued scores that are not conveniently scaled and which may be difficult to work with. Here the softmax is very useful because it converts the scores to a normalized probability distribution, which can be displayed to a user or used as input to other systems. For this reason, it is usual to append a softmax function as the final layer of the neural network.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.8}$$

where all the z_i values are the elements of the input vector and can take any real value. Softmax activation function coding implementation is shown in Figure 2.17.

```
def softmax(x):
    return torch.exp(x)/torch.sum(torch.exp(x),dim=1).view(-1,1)

# Here, out should be the output
probabilities = softmax(out)

# Does it have the right shape? Should be (64, 10)
print(probabilities.shape)
# Does it sum to 1?
print(probabilities.sum(dim=1))
```

```
torch.Size([64, 10])
tensor([ 1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
         1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
         1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
         1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
         1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
         1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,  1.0000,
```

Figure 2.17 Softmax Activation Function in Pytorch Coding

2.5.6 Rectified Linear Unit

ReLU stands for Rectified Linear Unit [1], and is a type of activation function. ReLU has become very popular in convolutional neural networks. Mathematically, ReLU is defined as :

$$y = max(0,x) \tag{2.9}$$

ReLU activation function is shown in Figure 2.18.

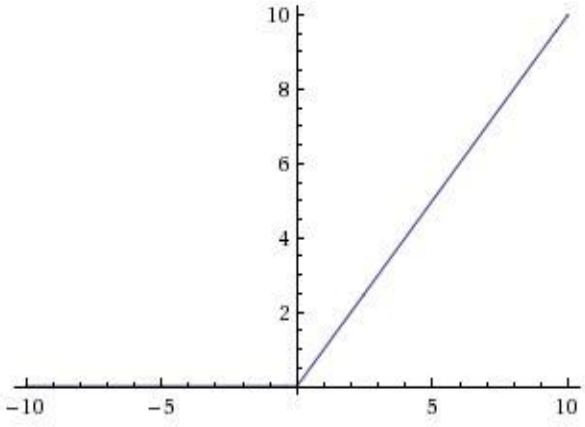


Figure 2.18 ReLU Activation Function

ReLU is linear (identity) for all positive values, and zero for all negative values. The advantages of ReLU are as follow:

- ReLU is cheap to compute as there is no complicated math. The model can therefore take less time to train or run.
- It converges faster. Linearity means that the slope does not plateau, or “saturate,” when x gets large. It does not have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh.

It is sparsely activated. Since ReLU is zero for all negative inputs, it is likely for any given unit to not activate at all.

2.5.7 Leaky Rectified Linear Unit

The Leaky Rectified Linear Unit (Leaky ReLU) is a type of activation function commonly used in artificial neural networks. It is similar to the Rectified Linear Unit (ReLU) activation function, but with a small modification that addresses the "dying ReLU" problem. The "dying ReLU" problem occurs when the input to a ReLU unit is negative, causing the unit to output zero, and effectively "die". This can happen during training when the weights are updated in a way that causes the ReLU

units to become stuck at zero. When a large number of ReLU units "die", the network's performance can suffer. To address this issue, the Leaky ReLU function introduces a small slope for negative input values. Specifically, the Leaky ReLU function outputs x when x is positive, and a small negative slope (typically 0.01) times x when x is negative. For example, leaky ReLU may have $y = 0.01x$ when $x < 0$. The sample of leaky ReLU is shown in Figure 2.19.

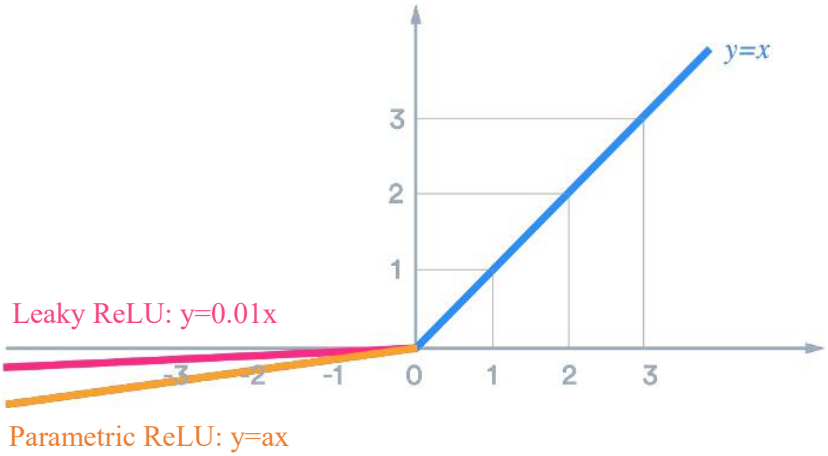


Figure 2.19 Leaky ReLU Activation Function

Leaky ReLU has two benefits:

- ❖ It fixes the “dying ReLU” problem, as it does not have zero-slope parts.
- ❖ It speeds up training. There is evidence that having the “mean activation” be close to 0 makes training faster. (It helps keep off-diagonal entries of the Fisher information matrix small, but the user can safely ignore this.) Unlike ReLU, leaky ReLU is more “balanced,” and may therefore learn faster. But Leaky ReLU is not always superior to plain ReLU, and should be considered only as an alternative.

2.6 Types of Loss functions

Machines learn by means of a loss function. It is a method of evaluating how well specific algorithm models the given data. If predictions deviate too much from actual results, loss function would cough up a very large number. Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction. But there is no one-size-fits-all loss function to algorithms in machine learning. There are various factors involved in choosing a loss function for specific problem such as type of machine learning algorithm chosen, ease of calculating the

derivatives and to some degree the percentage of outliers in the data set. Broadly choosing a loss function can be classified into two major categories depending upon the type of learning task, regression losses or classification losses.

2.6.1 Mean Square Error (MSE)

Mean Square Error (MSE) is measured as the average of squared difference between predictions and actual observations. MSE is mostly used for regression problems. It is only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions. Moreover, MSE has nice mathematical properties which makes it easier to calculate gradients [8].

$$\text{MSE} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n} \tag{2.10}$$

where y_i is actual label output of i^{th} label,

$\hat{y}_i$ is predicted output of i^{th} label

2.6.2 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) measures the average of sum of absolute differences between predictions and actual observations. Like MSE, this as well measures the magnitude of error without considering their direction. Unlike MSE, MAE needs more complicated tools such as linear programming to compute the gradients. MAE is more robust to outliers since it does not make use of square [8].

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{A} \tag{2.11}$$

where y_i is actual label output of i^{th} label,

$\hat{y}_i$ is predicted output of i^{th} label

2.6.3 Multi class SVM Loss

Multi class SVM Loss is a type of classification loss and it calculates the score of correct categories should be greater than sum of scores of all incorrect categories by some safety margin. And hence hinge loss is used for maximum-margin classification, most notably for support vector machines. Although not differentiable, it is a convex function which makes it easy to work with usual convex optimizers.

$$SVM_{loss} = \sum_{j \neq y_i} max(0, S_j - S_{y_i} + 1) \tag{2.12}$$

This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label. When the actual label is 1 (y(i) = 1), second half of function disappears whereas in case actual label is 0 (y(i) = 0) first half is dropped off.

The loss function is just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are confident but wrong.

$$\text{Cross Entropy loss} = -(y_i log(\hat{y}_i) + (1 - y_i)\, log(1 - \hat{y}_i)) \qquad (2.13)$$

2.7 Types of CNN Architectures

ImageNet has been running an annual competition in visual recognition (ImageNet Large Scale Vision Recognition Challenge-ILSVRC) where participants are provided with 1.4 million of images. Some popular CNN architecture won in ILSVRC competitions are LeNet-5, AlexNet, GoogLeNet, VGGNet and ResNet.

2.7.1 LeNet-5

LeNet-5 [2] is one of the simplest architectures. It has 2 convolutional and 3 fully-connected layers. The average-pooling layer was called a sub-sampling layer and it had trainable weights (which isn't the current practice of designing CNNs nowadays). This architecture has about 60,000 parameters. LeNet-5 is shown in Figure 2.20.

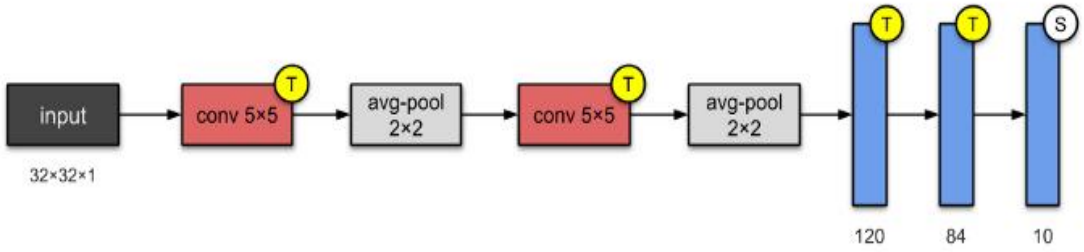


Figure 2.20 LeNet-5 Architecture

2.7.2 AlexNet

With 60M parameters, AlexNet [2] has 8 layers, 5 convolutional and 3 fullyconnected. AlexNet just stacked a few more layers onto LeNet-5. At the point of publication, the authors pointed out that their architecture was "one of the largest

convolutional neural networks to date on the subsets of ImageNet". The network had a very similar architecture as LeNet by Yann LeCun et al [10] but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. Figure 2.21 shows AlexNet architecture.

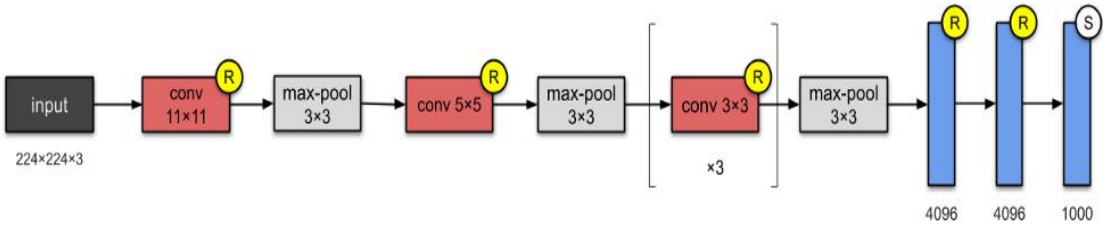


Figure 2.21 AlexNet Architecture

2.7.3 VGG-16 (2014)

The folks at Visual Geometry Group (VGG) invented the VGG-16 [2][9] which has 13 convolutional and 3 fully-connected layers, carrying with them the ReLU tradition from AlexNet. This network stacks more layers onto AlexNet, and use smaller size filters (2×2 and 3×3). VGG-16 consists of 138M parameters and takes up about 500MB of storage space. It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor. Figure 2.22 shows VGG-16 architecture.

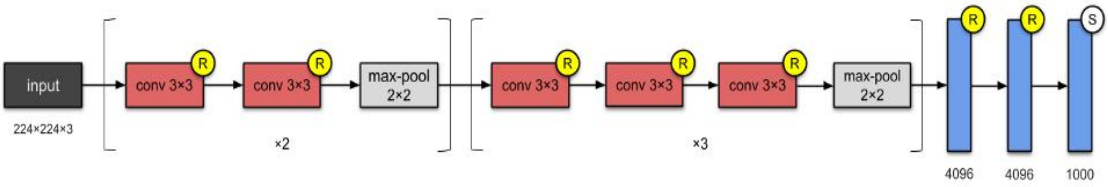


Figure 2.22 VGG-16 Architecture

2.7.4 VGG-19

VGG19 is a deeper version of VGG16 where the number 19 represents the number of trained layers of which 16 are convolutional and 3 are fully connected. It is composed of approximately 144 million parameters while VGG16 has a total of 138 million parameters. The VGG19 model was introduced in the 2014 ImageNet

competition, where it achieved state-of-the-art performance on image classification tasks. Compared with VGG16, VGG19 is slightly better but requests more memory. The VGG19 is one of the most popular pre-trained models for image classification. Figure 2.23 shows VGG-19 architecture.

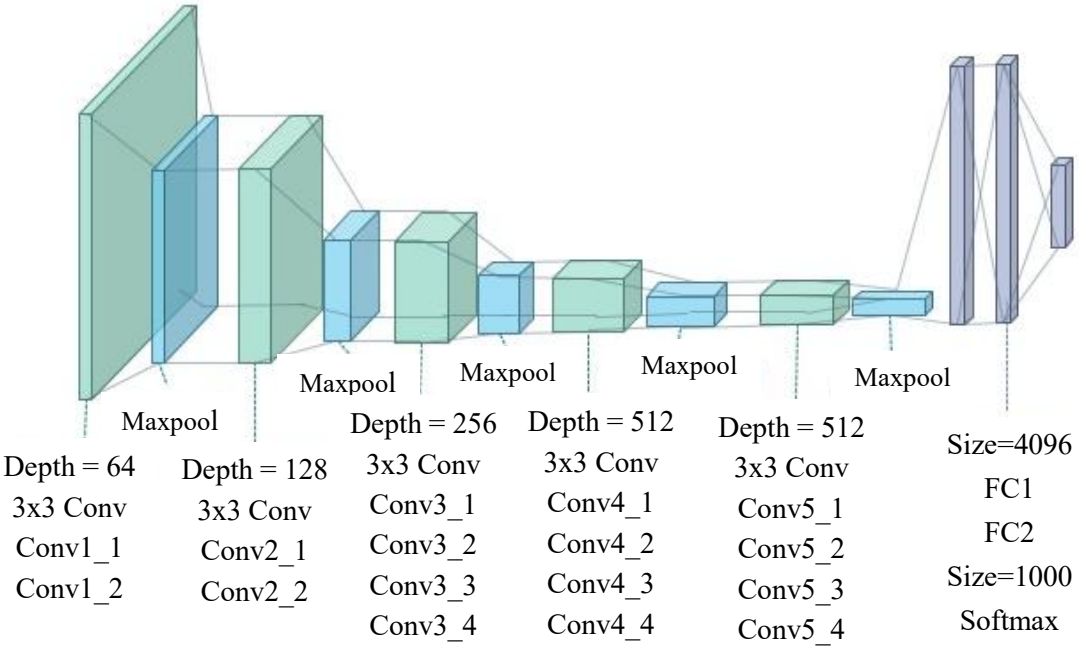


Figure 2.23 VGG-19 Architecture

2.7.5 Inception-V1

Inception-V1 [2] is 22-layers architecture with 5M parameters convolutional neural network. This is done by means of Inception modules. An Inception Module is an image model block that aims to approximate an optimal local sparse structure in a CNN. Put simply, it allows to use multiple types of filter size, instead of being restricted to a single filter size, in a single image block in which the user then concatenate and pass onto the next layer. The design of the architecture of an Inception module is a product of research on approximating sparse structures. Having parallel towers of convolutions with different filters, followed by concatenation, captures different features at 1×1, 3×3 and 5×5, thereby 'clustering' them. 1×1 convolutions are used for dimensionality reduction to remove computational bottlenecks. Due to the activation function from 1×1 convolution, its addition also adds non-linearity. This idea is based on the Network in Network paper.

The authors also introduced two auxiliary classifiers to encourage discrimination in the lower stages of the classifier, to increase the gradient signal that

gets propagated back, and to provide additional regularization. The auxiliary networks (the branches that are connected to the auxiliary classifier) are discarded at inference time. Inception-V1 architecture is shown in Figure 2.24.

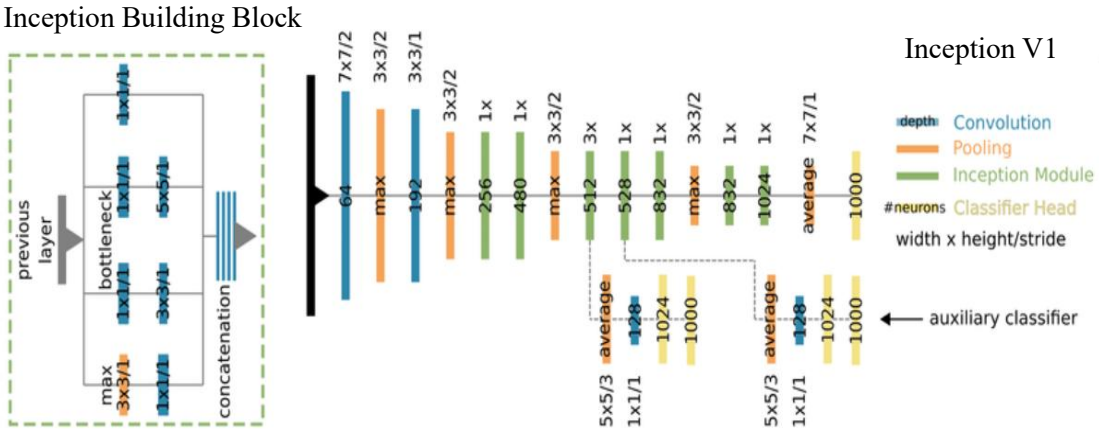


Figure 2.24 Inception-V1 Architecture

2.7.6 ResNet-50

ResNet (short for "Residual Network") is a type of neural network architecture that was introduced in 2015 by researchers at Microsoft Research. It is a deep convolutional neural network that is designed to address the problem of vanishing gradients in very deep networks, which can make training difficult. From the past few CNNs, the developers have seen nothing but an increasing number of layers in the design, and achieving better performance. But with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. The folks from Microsoft Research addressed this problem with ResNet [2][9] using skip connections (a.k.a. shortcut connections, residuals), while building deeper models.

Residual Networks is a classic neural network used as a backbone for many computers vision tasks. This model was the winner of ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed them to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training very deep neural networks was difficult due to the problem of vanishing gradients.

However, increasing network depth does not work by simply stacking layers together. Deep Networks are hard to train because of the notorious vanishing gradient problem as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. ResNet is one of the early

adopters of batch. The basic building block for ResNets are the conv and identity blocks. Figure 2.25 shows ResNet-50 architecture.

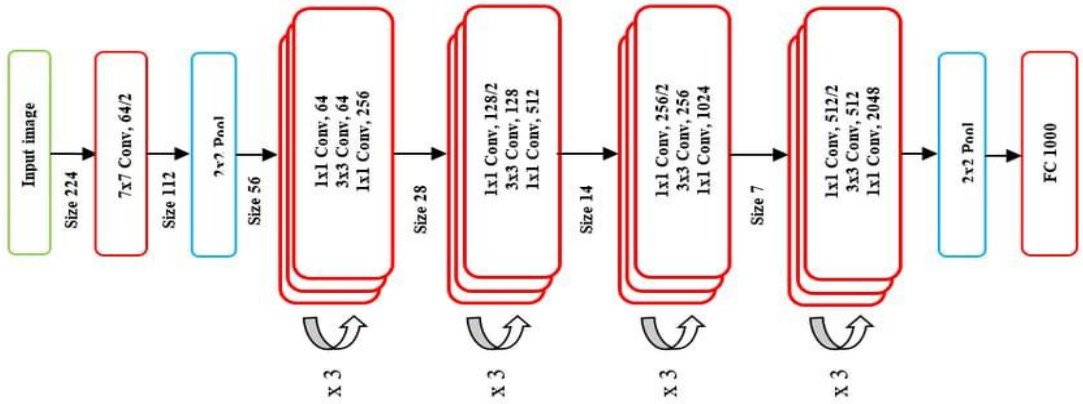


Figure 2.25 ResNet-50 Architecture

2.7.7 Xception (2016)

Xception is an adaptation from Inception, where the Inception modules have been replaced with depth wise separable convolutions. It has also roughly the same number of parameters as Inception-v1 (23M). With a modified depth wise separable convolution, it is even better than Inception-v3 [2] (also by Google, 1st Runner Up in ILSVRC 2015) for both ImageNet ILSVRC and JFT datasets. Figure 2.26 shows xception architecture.

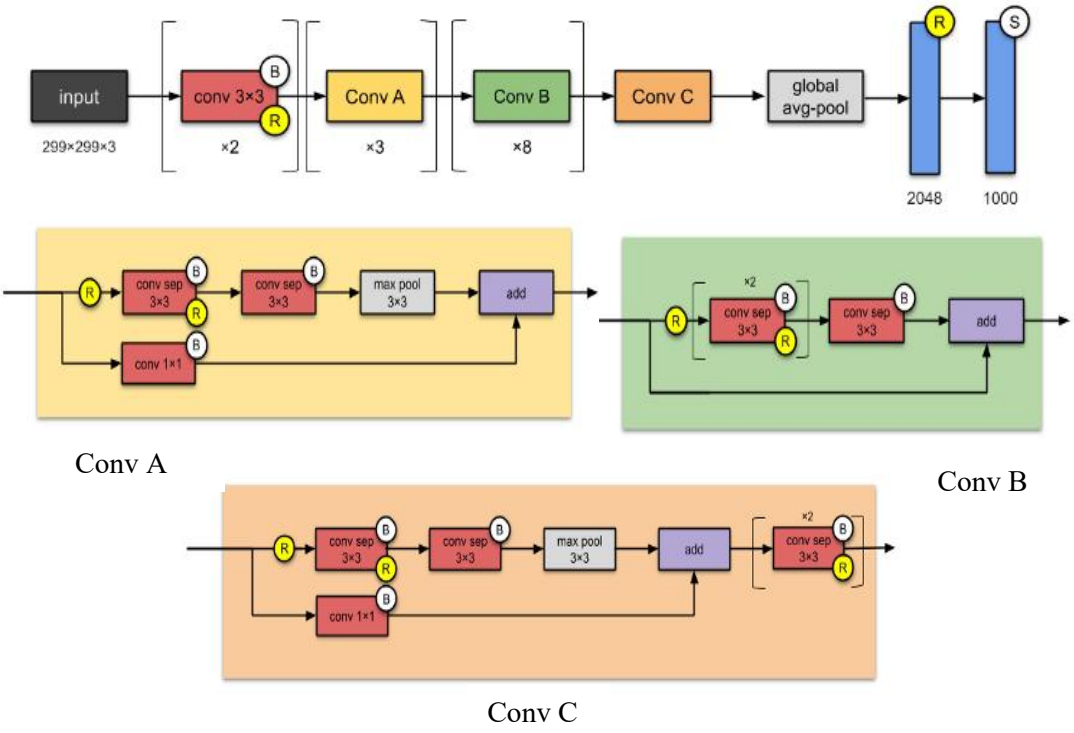


Figure 2.26 Xception Architecture

2.8 Building A Network with PyTorch

Building a Convolutional Neural Network (CNN) is a process that involves defining the architecture of the network, and then training the network on a large dataset to learn patterns and make predictions. The goal of building a CNN with PyTorch is to train the network to recognize patterns in images and make predictions about their class labels. PyTorch provides a module nn that makes building networks much simpler. Building a network for example with 784 inputs, two hidden units (128 & 64), 10 output units and a Softmax output is shown in Figure 2.27. Implementation of the network with Pytorch is shown in Figure 2.28. The following processes are the essential steps to build a CNN classifier model using PyTorch, but there are many details and variations that can be added, depending on the specific problem and data.

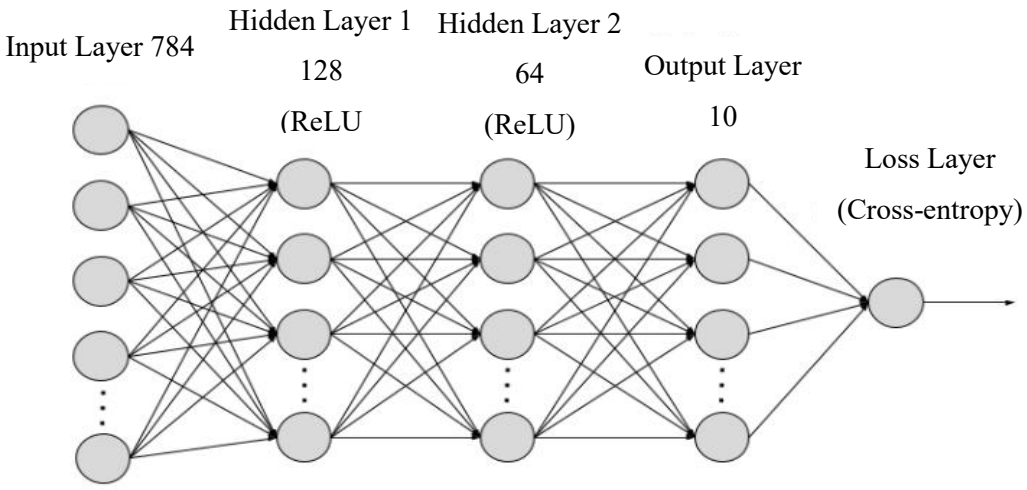


Figure 2.27 A Network with Two Hidden Layers

```
from torch import nn
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784,128)
        self.fc2 = nn.Linear(128,64)
        self.fc3 = nn.Linear(64,10)
    def forward (self,x):
        #x = self.fc1(x)
        x = F.relu(self.fc1(x))
        #x = self.fc2(x)
        x = F.relu(self.fc2(x))
        #x = self.fc3(x)
        x = F.softmax(self.fc3(x),dim = 1)
        return x
model = Network()
model
```

```
Network(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
```

Figure 2.28 Building a Network in Pytorch Coding Example

2.8.1 Forward Pass

After building a network, forward propagation will take place when the input image is passed. Figure 2.29 shows the coding implementation of forward propagation.

```
# Grab some data
data = ./path/to/train_dataset
images, labels = data.next()

# Resize images into a 1D vector, new shape is (batch size, color channels, image pixels)
images.resize_(64, 1, 784)
# or images.resize_(images.shape[0], 1, 784) to automatically get batch size

# Forward pass through the network
img_idx = 0
ps = model.forward(images[img_idx,:])

img = images[img_idx]
helper.view_classify(img.view(1, 28, 28), ps)
```

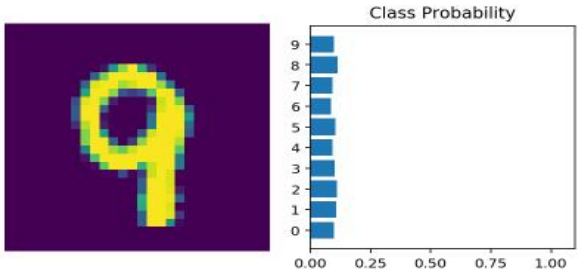


Figure 2.29 Coding Example for Forward Propagation Process

In the output Figure above, the network has basically no idea what this digit is. It is because it have not trained yet, all the weights are random. So the next step is training the network.

2.9 Training Neural Network

At first the network is naive, it does not know the function mapping the inputs to the outputs. Training the network by showing it examples of real data, then adjusting the network parameters such that it approximates this function as shown in Figure 2.30 is the first step in training process.

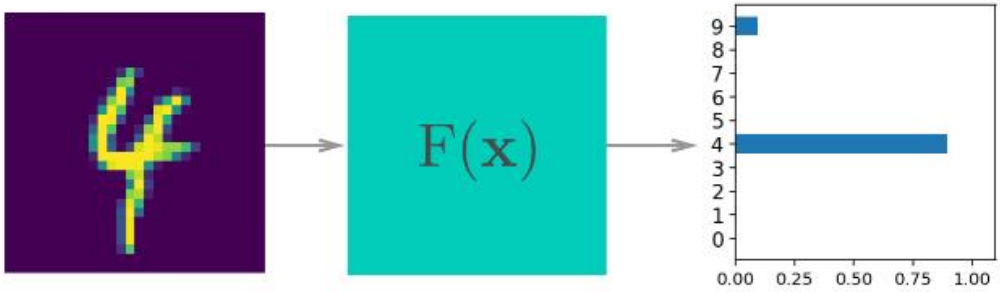


Figure 2.30 Class Prediction Example for Handwritten Digit

To find the most appropriate parameters for network, it needs to know that how poorly the network is predicting the real outputs. For this, calculation of a loss function (also called the cost) results prediction error. For example, the mean squared loss is often used in regression and binary classification problems.

$$loss, \ell = \frac{1}{2n}\sum_i^n (y_i - \hat{y}_i)^2 \tag{2.14}$$

where n is the number of training examples, y_i are the true labels, and $\hat{y}_i$ are the predicted labels.

By minimizing this loss with respect to the network parameters, configurations where the loss is at a minimum can be executed and the network is able to predict the correct labels with high accuracy. We find this minimum using a process called gradient descent. The gradient is the slope of the loss function and points in the direction of fastest change. To get to the minimum in the least amount of time, the gradient will be needed to follow downwards.

2.9.1 Backpropagation

For single layer networks, gradient descent is straightforward to implement. However, it is more complicated for deeper, multilayer neural networks like the one which have been built. Complicated enough that it took about 30 years before researchers figured out how to train multilayer networks. Training multilayer networks is done through backpropagation which is really just an application of the chain rule from calculus. The backpropagation algorithm consists of the four main steps. At the first step, forward pass, the input data is fed into the neural network, and the output is computed by propagating the activations through the network. The second step is error computation. The error between the predicted output and the actual output is computed using a loss function. The most commonly used loss function for classification tasks is cross-entropy loss. At the third step, backward pass, the error is backpropagated through the network, computing the gradients of the weights with respect to the error using the chain rule of calculus. After that, updating the weights of each neuron in hidden layers process is followed. The gradients are used to update the weights of the network using an optimization algorithm such as stochastic gradient descent (SGD). For the highest accuracy score and minimum error between predicted and actual outputs, steps 1-4 are repeated for multiple epochs until the network converges to a satisfactory solution. By minimizing the error, the network is able to learn the underlying patterns in the data, allowing it to make accurate

predictions on new unseen data. It is easiest to understand if it is converted to a two layer network into a graph representation as shown in Figure 2.31.

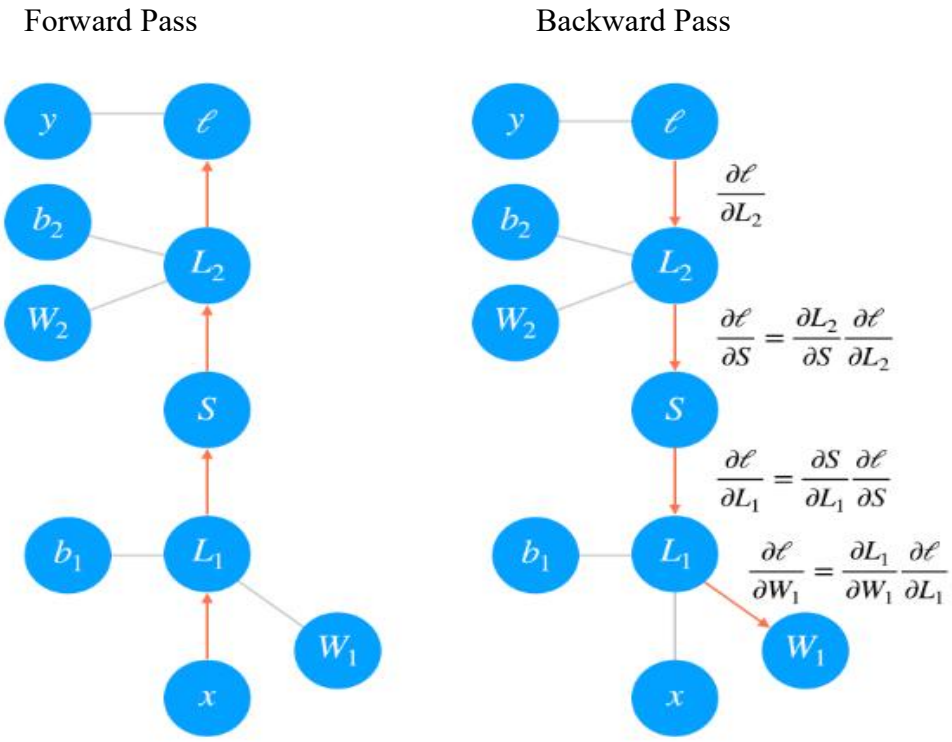


Figure 2.31 Forward and Backward Propagation General Diagram

In the forward pass through the network, the data and operations go from bottom to top here. And then pass the input $\boldsymbol{x}$ through a linear transformation $\boldsymbol{L_1}$ with weights W_1 and biases b_1. The output then goes through the sigmoid operation $\boldsymbol{S}$ and another linear transformation L_2**.** Finally the loss can be calculated by differentiating between output and target. The loss value can be used as a measure of how bad the network's predictions are. The goal then is to adjust the weights and biases to minimize the loss. To train the weights with gradient descent, propagate the gradient of the loss backwards through the network. Each operation has some gradient between the inputs and outputs. As the gradients are sent backward, the incoming gradient and the past gradient can be multiplied for the operation. In mathematically, this is just calculating the gradient of the loss with respect to the weights using the chain rule as shown in Equation 2.15.

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial L_1}{\partial W_1} \cdot \frac{\partial S}{\partial L_1} \cdot \frac{\partial L_2}{\partial S} \cdot \frac{\partial \ell}{\partial L_2} \tag{2.15}$$

Updating the weights using this gradient with some learning rate α is as shown in Equation 2.16.

$$W_1' = W - \alpha \frac{\partial \ell}{\partial W_1} \tag{2.16}$$

where the learning rate α is set such that the weight update steps are small enough and the iterative method settles in a minimum.

2.9.2 Error Optimization

There is one last piece that is needed for training is an optimizer that will be used to update the weights with the gradients. It can get from PyTorch optim package. For example, using stochastic gradient descent can be written as optim.SGD. There are many kind of optimization algorithms to minimize the error step by step. Gradient descent algorithm is used in the proposed system. Figure 2.32 shows the general concept of gradient decent algorithm.

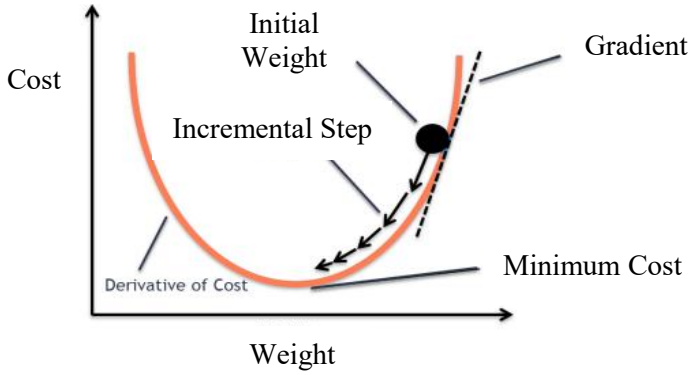


Figure 2.32 Adding an Optimizer in Training Process

2.10 Transfer Learning

Transfer learning is using pre-trained networks to solved challenging problems in computer vision. Specifically, using networks trained on ImageNet available from torchvision. ImageNet is a massive dataset with over 1 million labeled images in 1000 categories. It is used to train deep neural networks using an architecture called convolutional layers. Once trained, these models work astonishingly well as feature detectors for images they were not trained on. Using a pre-trained network on images not in the training set is called transfer learning.

There are two main parts in pre-trained models : the features and the classifier. The features part is a stack of convolutional layers and overall works as a feature detector that can be fed into a classifier. The classifier part is a single fully-connected layer. This classifier layer was trained on the ImageNet dataset, so it will not work for specific problem. That means the pre-trained model needs to replace the classifier,

but the features will work perfectly on their own. In general, pre-trained networks are amazingly good feature detectors that can be used as the input for simple feed-forward classifiers. With “torchvision.models” these pre-trained networks can be downloaded and use them in many applications. Figure 2.33 shows the general concept of transfer learning.

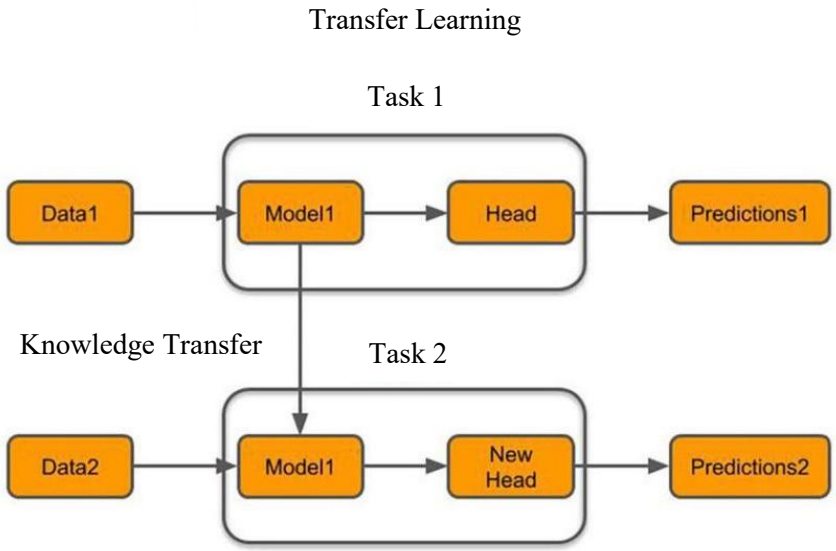


Figure 2.33 Transfer Learning General Block Diagram

2.11 Summary

In this chapter, the related theoretical background of the system and the methods used to develop a classifier model are described. In the next chapter, the system design and programming framework for the system and implementation of CNN based image classifier are explained with the relevant figures.

CHAPTER 4
TEST AND RESULTS

This chapter describes the testing results of design and implementation of Image Classifier for Goods. It includes the comparison of training and validation accuracy, average accuracy calculation, likelihood rank calculation for different classes and classification result of good for every classes. Input image acquisition is taken by webcam.

4.1 Inference and Validation

Now the trained model can be used for making predictions. This is typically called inference, a term borrowed from statistics. However, neural networks have a tendency to perform too well on the training data and are not able to generalize to data that has not been seen before. This is called overfitting and it impairs inference performance. To test for overfitting while training, the measurement of performance on data which is not in the training set is called the validation set. To avoid overfitting through regularization, the process called dropout is added while monitoring the validation performance during training. So, the model will be needed to test with validation dataset first. Figure 4.1 shows the loss comparison graph of training and validation processes after 30 epochs.

```
Epoch25/30..Train loss: 0.363..valid lossS: 1.151..valid accuracy : 0.816..
Epoch26/30..Train loss: 0.653..valid lossS: 1.035..valid accuracy : 0.833..
Epoch27/30..Train loss: 0.226..valid lossS: 0.890..valid accuracy : 0.861..
Epoch28/30..Train loss: 0.311..valid lossS: 0.937..valid accuracy : 0.878..
Epoch29/30..Train loss: 0.038..valid lossS: 1.205..valid accuracy : 0.840..
Epoch30/30..Train loss: 0.284..valid lossS: 1.465..valid accuracy : 0.812..

Out[6]: <matplotlib.legend.Legend at 0x7fd4cac2a828>
```

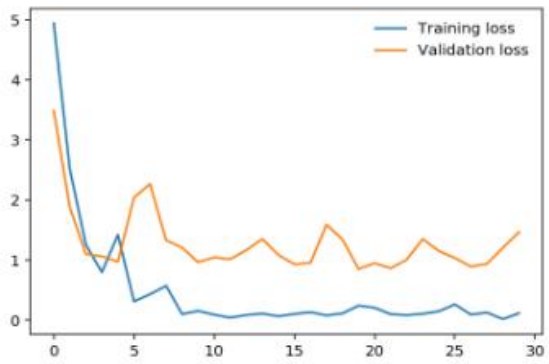


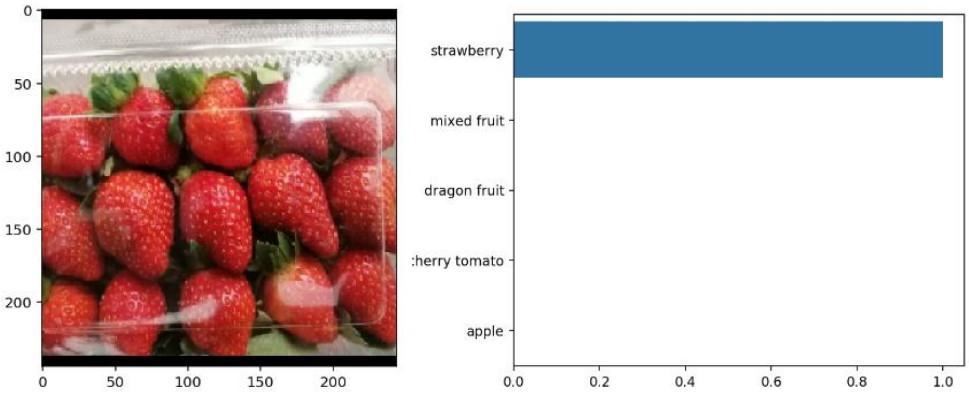
Figure 4.1 Training and Validation Loss Comparison

4.2 Testing the Classifier with Different Classes of Goods

After testing the model performance on validation dataset, it can be seen that the result meets the criterion. So then, it is ready to start the prediction of Goods for different classes. The following figures show the likelihood percentage for different classes.

4.2.1 Testing Result for Strawberry

The following Figure 4.2(a) is the input image to classifier model and the Figure 4.2(b) is the likelihood result plot of classification for the input image. In Figure 4.2(b), the likelihood percentage for strawberry is almost 100% and there is no likelihood percentage for other classes.

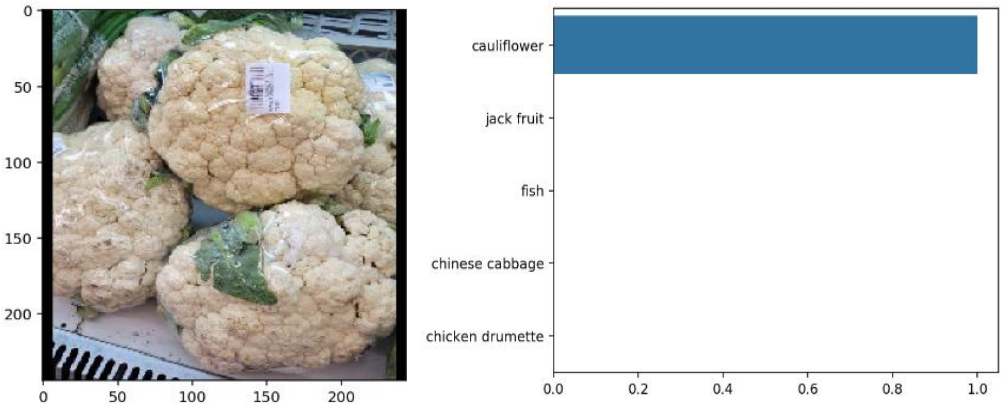


(a) **(b)**

Figure 4.2 Classification Result for Strawberry

4.2.2 Testing Result for Cauliflower

The following Figure 4.3(a) is the input image to classifier model and the Figure 4.3(b) is the likelihood result plot of classification for the input image.

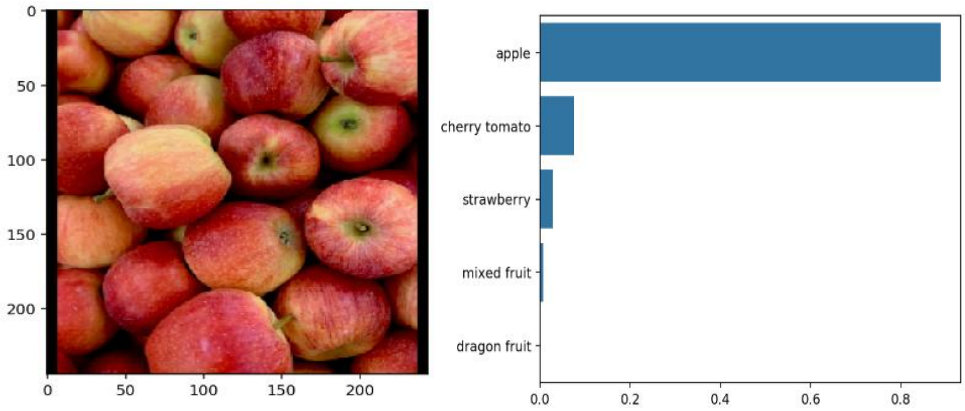


(a) **(b)**

Figure 4.3 Classification Result for Cauliflower

4.2.3 Testing Result for Apple

The following Figure 4.4(a) is the input image to classifier model and the Figure 4.4(b) is the likelihood result plot of classification for the input image. In Figure 4.4(b), the likelihood percentage for apple is over 90% and approximately 6% likelihood for cherry tomato.

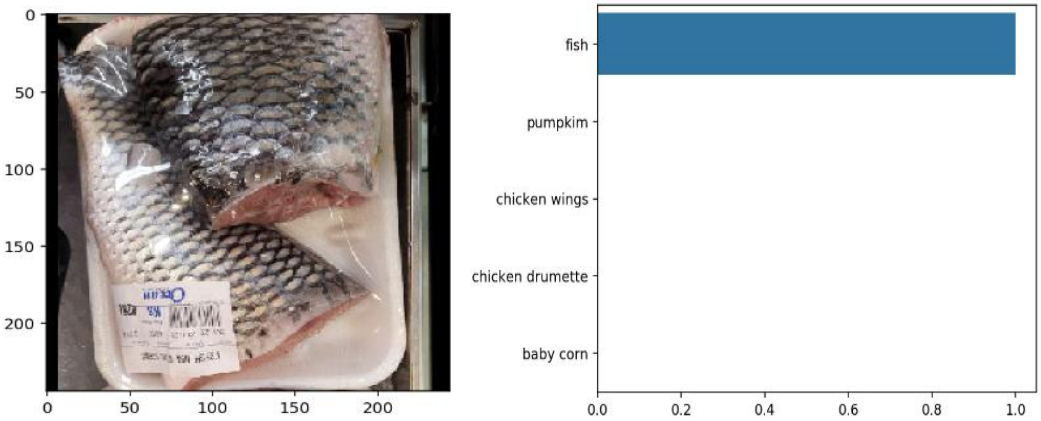


(a) **(b)**

Figure 4.4 Classification Result for Apple

4.2.4 Testing Result for Fish

The following Figure 4.5(a) is the input image to classifier model and the Figure 4.5(b) is the likelihood result plot of classification for the input image. In Figure 4.5(b), the likelihood percentage for fish is almost 100% and there is no likelihood percentage for other classes.



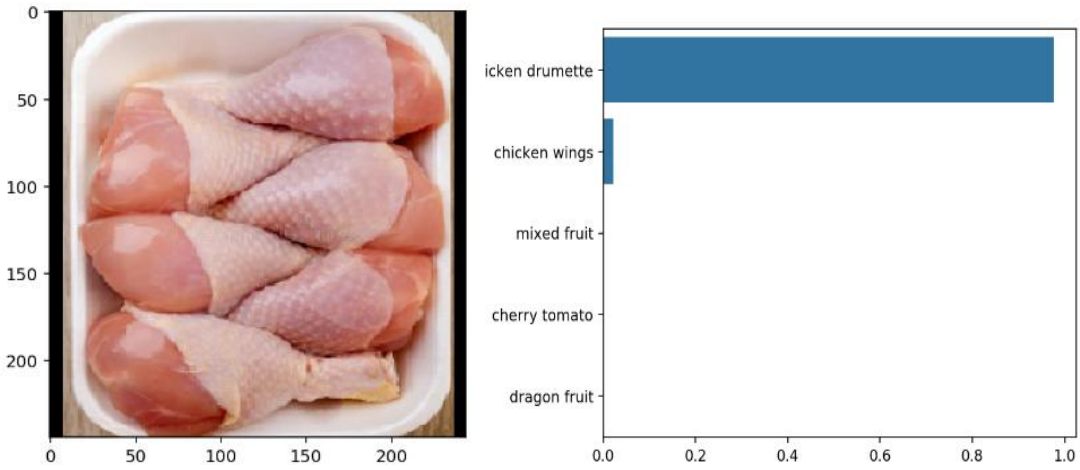
(a) **(b)**

Figure 4.5 Classification Result for Fish

4.2.5 Testing Result for Chicken Drumette

The following Figure 4.6(a) is the input image to classifier model and the Figure 4.6(b) is the likelihood result plot of classification for the input image. In

Figure 4.6(b), the likelihood percentage for chicken drumette is over 90% and chicken wings likelihood is approximately 5%.

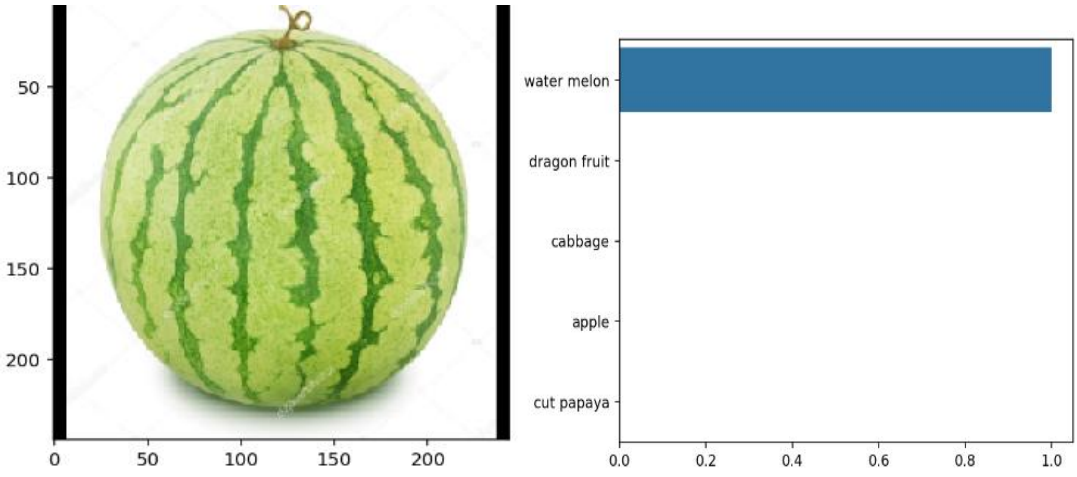


(a) **(b)**

Figure 4.6 Classification Result for Chicken Drumette

4.2.6 Testing Result for Watermelon

The following Figure 4.7(a) is the input image to classifier model and the Figure 4.7(b) is the likelihood result plot of classification for the input image. In Figure 4.7(b), the likelihood percentage for watermelon is almost 100% and there is no likelihood percentage for other classes.



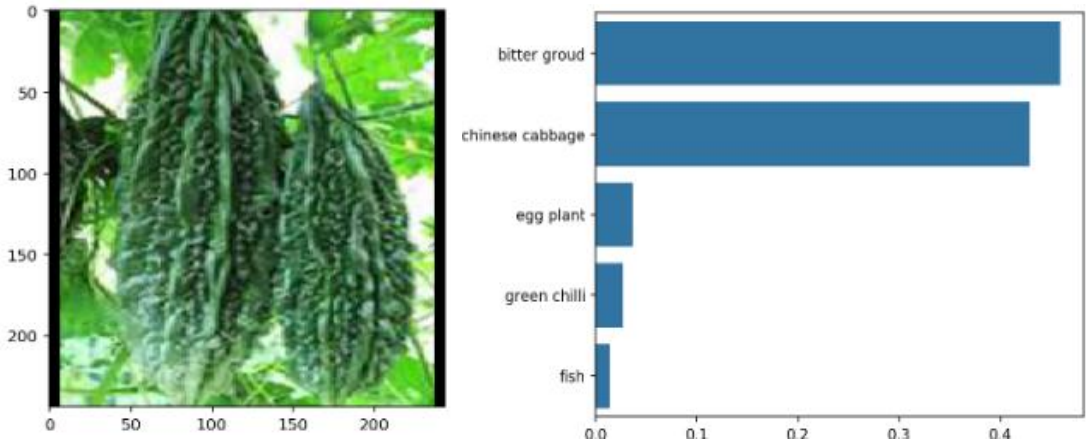
(a) **(b)**

Figure 4.7 Classification Result for Watermelon

4.2.7 Testing Result for Bitter Gourd

The following Figure 4.8(a) is the input image to classifier model and the Figure 4.8(b) is the likelihood result plot of classification for the input image. In Figure 4.7(b), the likelihood percentage for watermelon is approximately 50%,

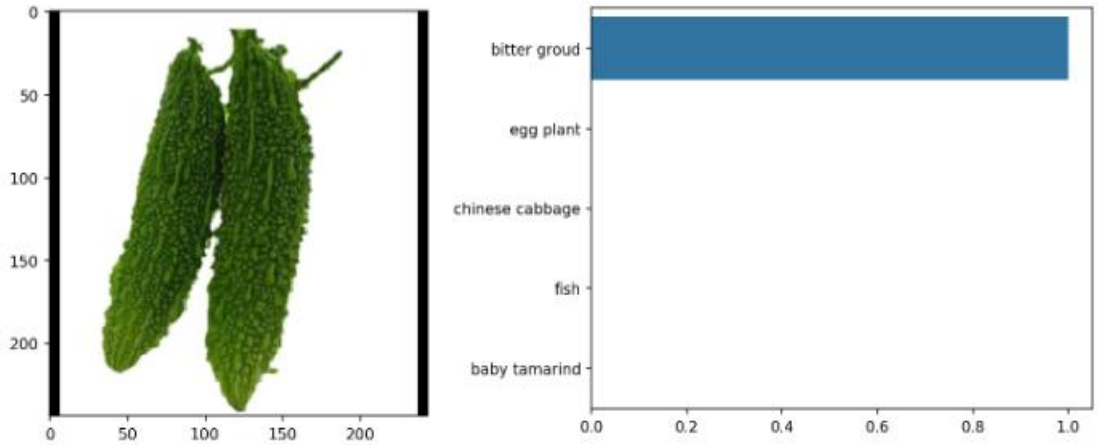
chinese cabbage likelihood is over 40% and egg plant likelihood is nearly 10%. Although the top most likelihood is bitter gourd, the likelihood percentage for true class is slightly insignificant while the other testing results are over 90%.



(a) **(b)**

Figure 4.8 Classification Result for Bitter Gourd

This occurs because of the complex background. When the capture region has a clear background and the goods image is captured clearly, the classification result percentage is significant. This is demonstrated in Figure 4.9.



(a) **(b)**

Figure 4.9 Classification Result for Bitter Gourd in Clear Background

4.3 Setting up the Classifier Model in Terminal (Command Prompt)

To perform this process, the local machine will be needed to build a scripting environment with essential libraries for Python program operation. After that, it is almost ready to implement with webcam for input image acquisition and output the top class's text to speech in the terminal workspace. Figure 4.10 shows the sample output result in command line application. The predicted result in speech is played

with the device's MP3 player after the text result is executed. Basic Usage for prediction in cmd workspace is python predict.py /image_path

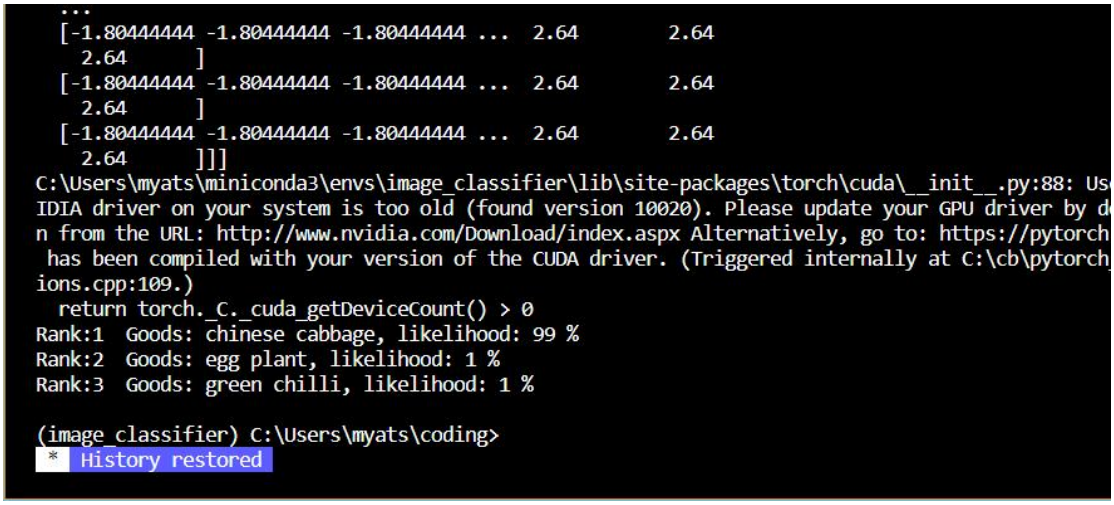


Figure 4.10 Classification Result in Command Line Interface

4.4 Performance Evaluation

In this section, overall accuracy of this system is calculated. In the first stage, training is performed. The train and test dataset are shown in Table 4.1.

The equations of performance evaluation are as follows:

$$\text{Accuracy} = \frac{\text{correctly predicted class}}{\text{total testing class}} \times 100\% \tag{4.1}$$

$$\text{Precision(Rank)} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \times 100\% \tag{4.2}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \times 100\% \tag{4.3}$$

For 50 test images, confusion matrix table is as follows:

Table 4.1 Training Dataset and Testing Dataset

	Strawberry	Corn	Chilli	Chicken wing	Jack fruit
Testing set	10	10	10	10	10
Training set	100	100	100	100	100

Table 4.2 Confusion Matrix for Five Class of Goods

	Predicted Class						
Actual Class		Strawberry	Corn	Cherry	Chicken wing	Jack fruit	Accuracy
	Strawberry	9	0	1	0	0	90%
	Corn	0	10	0	0	0	100%
	Cherry	2	0	8	0	0	80%
	Chicken wing	0	0	0	8	0	80%
	Jack fruit	0	0	0	0	9	90%

4.5 Summary

In this model, the dataset is trained using a custom CNN, VGG-16 pre-trained base model with a new created classifier within 30 epochs and got the average accuracy of over 88%. The developer used adaptive learning rate optimization algorithm with initial learning rate of 0.001 and the Stochastic Gradient Descent (SGD) algorithm is used for error optimization in this system.

CHAPTER 5
DISCUSSION AND CONCLUSION

Classification of Goods for visually impaired person system have significant value of both academic and useful perspectives. This proposed system can be used in the daily life of a visually impaired person to be more independent to others. This system is implemented to be a great help to visually impaired person and can be upgraded with many more further functions.

The accuracy of each good is different according to their features in the dataset. It is designed to recognize the name of goods and purposed for visually impaired person. Generally, the failure to verify an image was due to poor image quality and high similarity between thirty types of goods. The overall performance and accuracy rate is about 80% according to the 100 test images and 20% is error rate. This system is implemented with python programming and it is used pytorch, open source deep learning frame work in building model. For camera streaming input and image loading, the system is used OpenCV, image processing frame work. The three benefits for this system are as follows:

- The user can classify the goods that are not easy to guess by the sense.
- Because of this system, the impaired person will be more independent in daily life.
- The impaired person will be more easy to do things as normal people when the system has upgraded with text to speech functions and so on.

5.1 Limitation

There are some limitations in this system. This system is designed to classify and recognize only thirty types of goods: Apple, cherry tomato, grape, chilli, pumpkin, watermelon, mixed fruit, jack fruit, bitter gourd, chicken wings, chicken drumette, cabbage, dragon fruit, strawberry, etc which can be found generally in supermarkets. Therefore, it cannot output the valid result of other type of goods. If the user inputs other types of goods, the system will assume those tests as unknown goods and it may not guarantee the accuracy of the result. Another limitation is it can't not classify the goods which are in similar shape or packed in similar box shape and colour. Moreover,

the image quality must also be good enough to see the good clearly. If the capture region has complex background, the system cannot find the correct good and cannot identify the type correctly. Sometimes, the system is unable to produce the correct results for the images which are captured from a long distance and in light reflected conditions.

During the designing of the CNN architecture, the system also considers the model speed and performance. To be real-time processing, the model needs to light weight deep learning model. So, to be light weight model, the accuracy can drop.

5.2 Conclusion

In conclusion, Classification of Goods for visually impaired person has demonstrated its capability in accurately identifying thirty different goods in images. The overall average accuracy is 88% in real-time testing with webcam by 30 epochs trained VGG-16 model. Moreover, this system could provide vital information to help visually impaired individuals navigate their environment and identify objects they encounter. This can be particularly useful in day-to-day activities such as grocery shopping, cooking and navigating unfamiliar spaces. This would greatly enhance the independence and quality of life for visually impaired individuals.

5.3 Further Extensions

For further extension, this system can be used to classify various types of objects in reality. The system can be accessed more resources to achieve better performance such as training on a larger dataset could achieve higher accuracy and precision. Classification can be advanced by adding some algorithms such as You Only Look Once (YOLO) for different objects detection in real-time with superb processing speed. It has the potential to significantly various industries and business by automating the process of goods identification and categorization, thus saving time and human resources. Moreover, if the system is upgraded for recognizing texts on goods, the limitation for not classifying accurately in similar shape goods will be solved. In summary, this system can be extended to provide information of goods, reading the text on papers, objects and anything that need to be classified in daily life of people.

CHAPTER 3
SYSTEM DESIGN AND IMPLEMENTATION

This chapter describes overall design and implementation of classifier model for prediction of different goods. Python programming language and PyTorch framework is used for system implementation. This system can be used in supermarkets for visually impaired person. In this system, Thirty different class of goods can be classified using VGG-16 CNN model.

3.1 System Implementation

In this section, system implementation of VGG-16 model will be described completely. The proposed system is trained base on VGG-16 model which is a custom Convolutional Neural Network (CNN) model. The training dataset contains good images. This dataset is created by collecting good images from google and supermarket and then passed through the pretrained VGG-16 convolutional base layer to generate the features of the image as shown in Figure 3.1. After that, the dataset is trained using custom CNN.

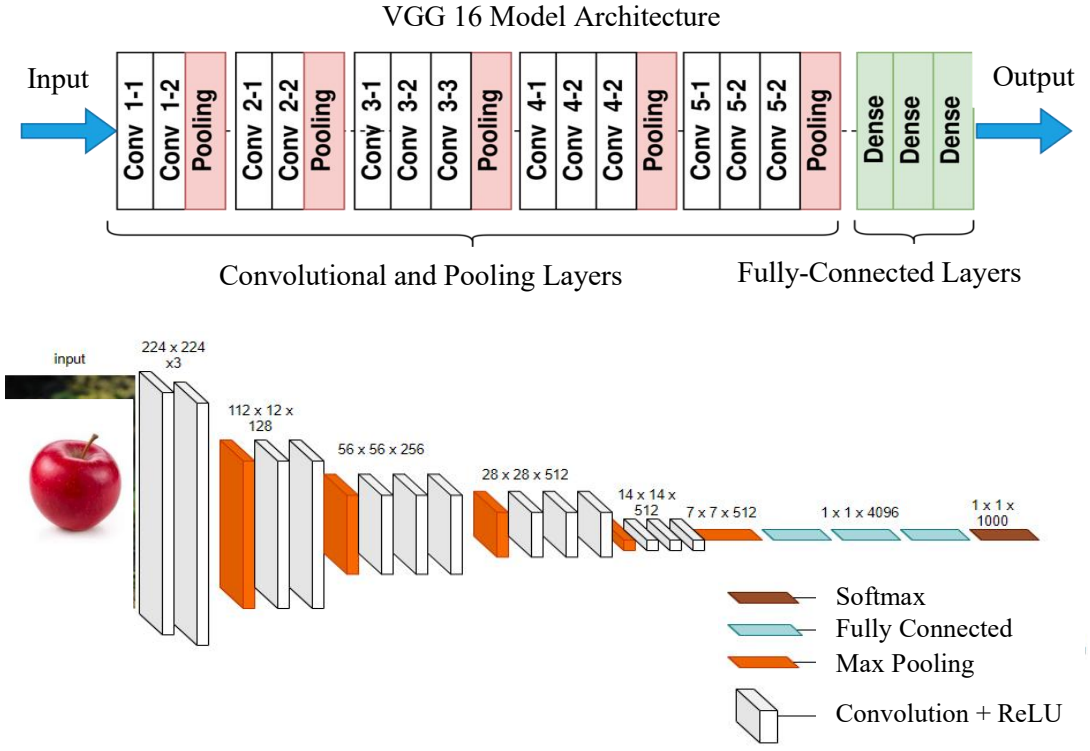


Figure 3.1 Sequence of VGG16 Layers for Class Prediction

3.2 Overall System Design

There are two stages in the system : Training and Testing the classifier model. The block diagram design of the system is shown in Figure 3.2.

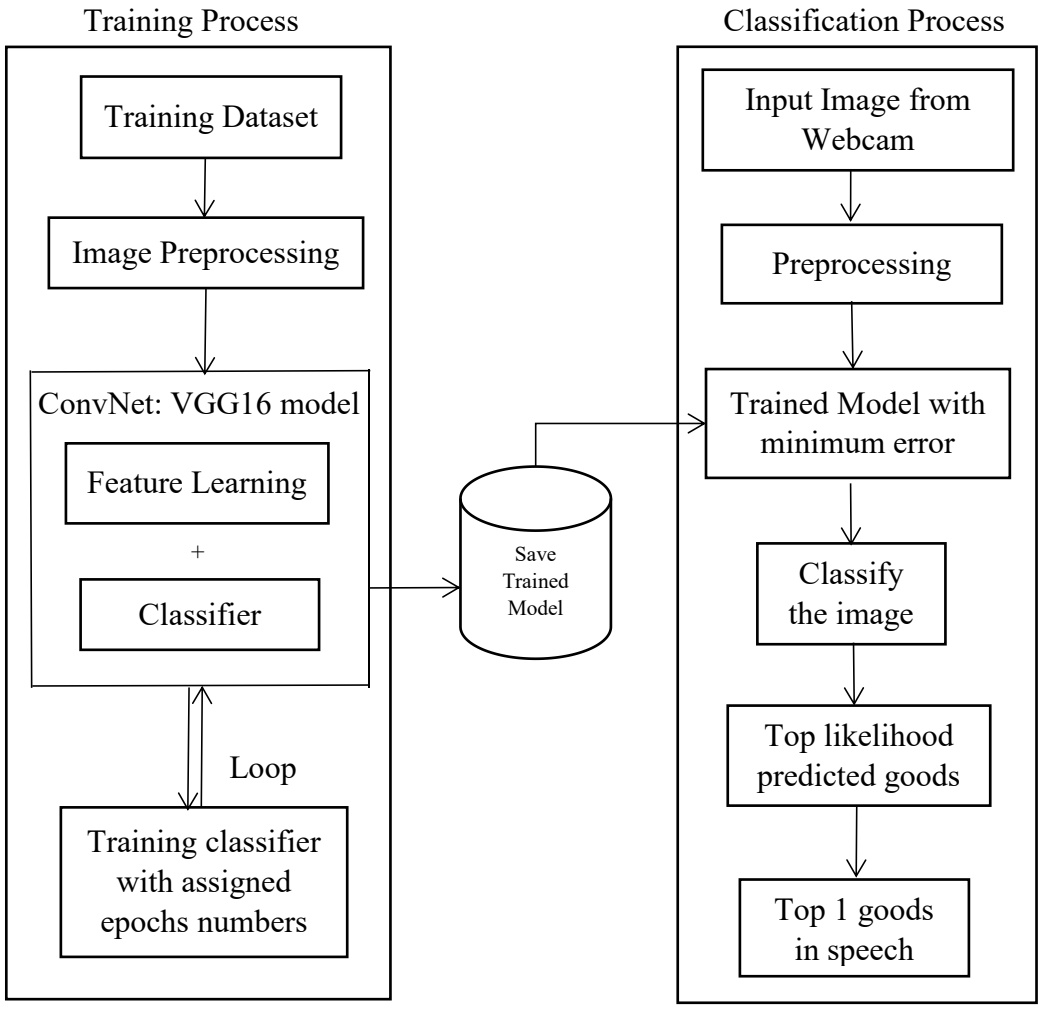


Figure 3.2 Overall System Design

3.3 Training Process

In the training process, the following steps and tasks are included.

- ❖ Creating Dataset : It is collected by taking images from market and online. After that, split the dataset of 80% for training, 10% for evaluating and 10% for testing process.
- ❖ Image Preprocessing : It sets images to 'RGB' mode and resize to 224x224 pixels as required by network then, normalize images to be suitable for VGG16 model to understand. And it transforms the pixel data to tensor form.

- ConvNet-VGG16 model : VGG16 feature layer is used for image feature extraction by transfer learning. And then, classifier network with 3 fully connected layers is built for image classification.
- Model Training : The parameters of pre-trained feature network are left static during training and the classifier of VGG-16 model is trained by multiple forward and backward iterations, epochs.
- Saving the Model : The model with target accuracy checkpoints is saved as a dictionary file (eg.checkpoint.pth).

3.4 Classification Process

In the classification process, the following steps and tasks are included for a complete application.

- Input Image : In the command prompt of local machine, the input image captured by webcam can be passed to classifier model or the input image can be accessed by manual that is giving the path directory of image.
- Image Preprocessing : The preprocessing step is similar to task of training process.
- Classification : The input image tensor form is accessed to the trained classifier model for classification.
- Predict the class of image : The system will result top five likelihood probabilities of predicted goods with descending rank value.
- Output : Finally, the system will output top-1 class text to speech.

Example demonstration for output testing process is shown in the following Figure 3.3 and Figure 3.4.

Figure 3.3 Example Testing Image_33

```
root@6662e91e90ae:/home/workspace/ImageClassifier# python predict.py --image ./goods/
08/image_33.png --checkpoint checkpoint.pth --top_k 5
Rank:1  Goods: pumpkim, liklihood: 72 %
Rank:2  Goods: jack fruit, liklihood: 25 %
Rank:3  Goods: dragon fruit, liklihood: 3 %
Rank:4  Goods: cut papaya, liklihood: 1 %
Rank:5  Goods: mixed fruit, liklihood: 1 %
root@6662e91e90ae:/home/workspace/ImageClassifier#
```

Figure 3.4 Output Rank for Predicted Goods

3.5 Model Implementation and Training

This section discusses about the training process. The detection device, webcam is used for this system. And then, Convolutional Neural Network model (CNN) is used for classification. The CNN model contains input layer, convolutional layers, max-pooling layers, dropout layers, flatten layer, fully connected layers and output layers. There are 4-convolution layers, 3-maxpooling layers and 3-fully connected layers in the model. When the model loaded successfully, the model summary will show in background. The model summary is shown in Figure 3.5 and the user can estimate the model processing time for single image by checking the trainable parameters. The model loading can take about 10s to 15s. The total trainable parameter amount is 45,457,541.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 300, 300, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 150, 150, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 150, 150, 32)      4640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 150, 150, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 75, 75, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 87616)             0
_________________________________________________________________
dense (Dense)                (None, 512)               44859904
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 2565
=================================================================
Total params: 45,457,541
Trainable params: 45,457,541
Non-trainable params: 0
```

Figure 3.5 Model Architecture Summary

This system trained the model with pytorch framework, the model file name shall be "*.pth". That means it is stored both the model weights and model architecture into a single file. Because of this file structure, the user can change the model architecture, loss functions, activation functions and learning rate after the system is launched. The VGG-16 CNN model for convolutional base layer is used for image feature extraction and the flowchart for the training process is described as follows.

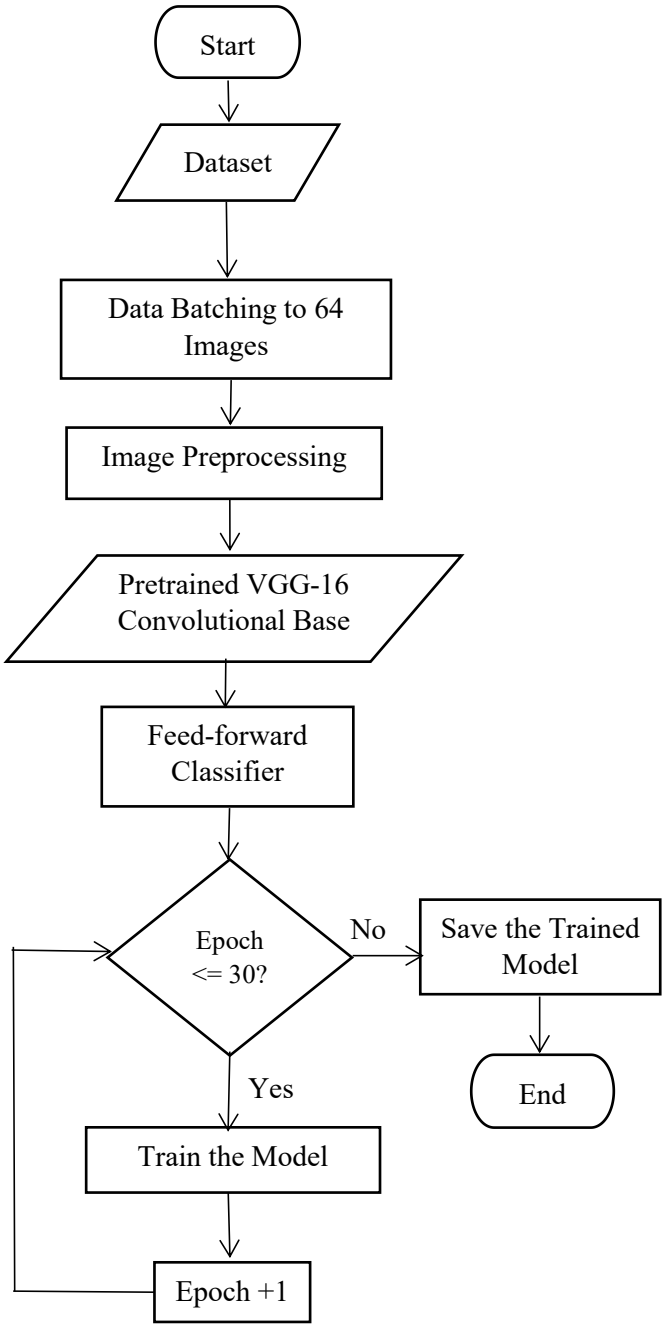


Figure 3.6 Training Process Flowchart

To build and train a new feed-forward classifier using pretrained convolutional based feature extraction layers, the following processes will be needed to take:

- ❖ Define a new, untrained feed-forward network as a classifier, using ReLU activation and dropout
- ❖ Make a forward pass through the network
- ❖ Use the network output to calculate the loss
- ❖ Perform a backward pass through the network with loss.backward()
- ❖ Calculate the gradient for error optimization process
- ❖ Take a step with the optimizer to update the weights
- ❖ Track the loss and accuracy on the validation set to determine the best hyper parameters

Figure 3.7 shows the coding implementation of building a classifier network.

```
def initial_model(architecture):
    model = models.vgg16(pretrained = True)
    model.name = 'vgg16'
    print(model.name)
    for param in model.parameters():
        param.requires_grad = False
    return model

def initial_classifier(model,hidden_units):
    from collections import OrderedDict
    classifier = nn.Sequential(OrderedDict([
        ('fc1',nn.Linear(25088,2000)),
        ('relu1',nn.ReLU()),
        ('dropout',nn.Dropout(0.3)),
        ('fc2',nn.Linear(2000,1000)),
        ('relu2',nn.ReLU()),
        ('dropout',nn.Dropout(0.3)),
        ('fc3',nn.Linear(1000,30)),
        ('output',nn.LogSoftmax(dim=1))]))
    model.classifier = classifier
    print(model)
    print("No. of hidden units(fc1) :" , hidden_units)
    return classifier
```

Figure 3.7 Building a Classifier Network in Pytorch

After that, an error optimization algorithm is added for training process and backward propagation algorithm is added as shown in coding implementation Figure 3.7. Looping this algorithm for one pass through entire dataset is called an epoch. For each batch of epoch, a training forward pass to calculate the loss, do a backwards pass, and update the weights. The training process will take place till the number of assigned epochs is completed. At the end of assigned epochs, the training loss, validation loss and the progress of accuracy at each epoch will be printed out with a comparison graph.

The coding implementation for training process is shown in Figure 3.8.

```
def network_training (model,trainloader,validloader):
    epoch = 50
    running_loss = 0
    print("Number of epochs :", epoch)
    print('Training process initializing...\n')
    for e in range(epoch):
        for images,labels in trainloader:
            steps += 1
            images,labels = images.to(device),labels.to(device)
            optimizer.zero_grad()
            logps = model.forward(images)
            loss = criterion(logps,labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if steps % print_every == 0:
                valid_loss,accuracy = 0,0
                model.eval()
```

Figure 3.8 Coding Implementation for Training Process

3.6 Saving the Model

When the network has trained to a minimum error value, the model is saved with a dictionary file (checkpoint.pth), so it can be loaded later for making predictions. To save the trained model in Pytorch, torch.save function is used as shown in Fig 3.9.

```
model.class_to_idx = train_dataset.class_to_idx
torch.save({'state_dict': model.state_dict(),
            'drop_out':0.3,
            'epoch':30,
            'classifier': model.classifier,
           'class_to_idx': model.class_to_idx,
           'optimizer_dict': optimizer.state_dict()},'checkpoint2.pth')
```

Figure 3.9 Coding Implementation for Model Saving

3.7 Loading the Trained Model for Prediction

A function to load checkpoints is shown in Figure 3.10.

```
checkpoint = torch.load('checkpoint2.pth',map_location='cpu')
model.classifier = checkpoint['classifier']
model.class_to_idx = checkpoint['class_to_idx']
model.load_state_dict(checkpoint['state_dict'])
model.eval()
```

Figure 3.10 Coding Implementation for Loading Trained Model

3.8 Testing the Model for Prediction

After training the model for a specific purpose, the model requires to be tested if the accuracy meets the criteria for application. Testing a trained neural network model is an important step in the development process, as it provides insight into the model ability to generalize to new, unseen data. Figure 3.10 shows the flowchart for prediction process.

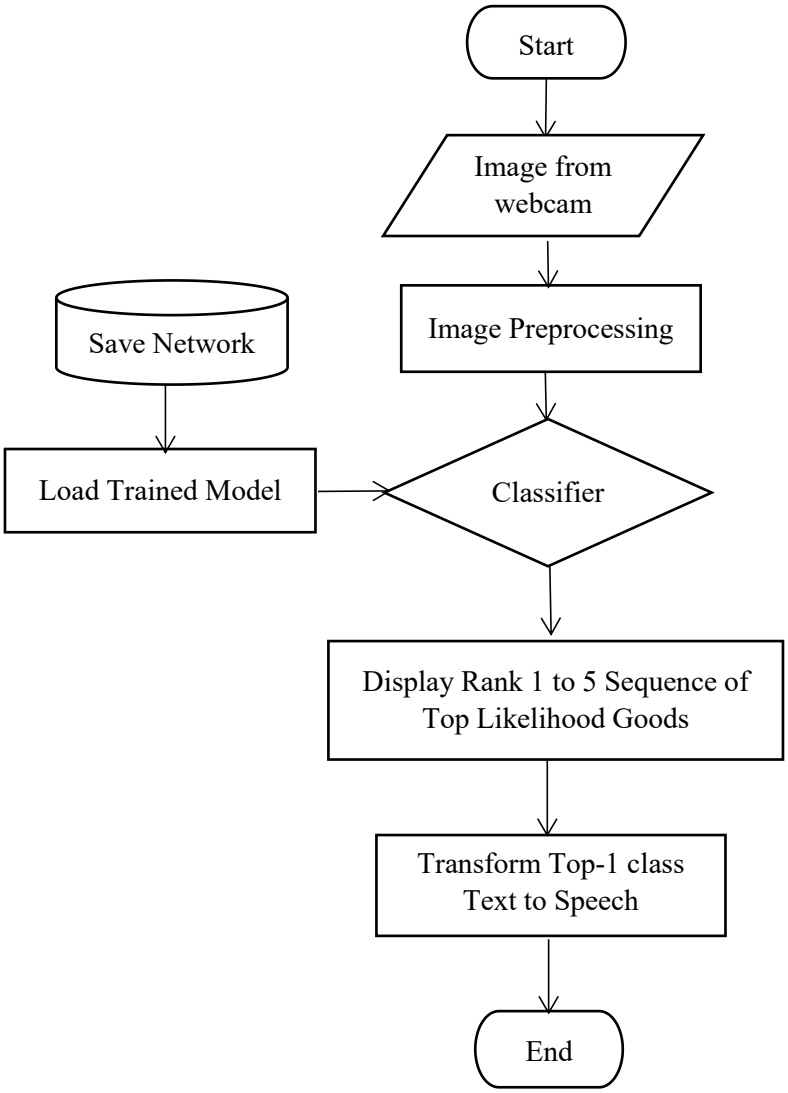


Figure 3.11 System Flowchart For Prediction Process

3.9 Summary

The model implementation and training process take place at the cloud workspace with GPU. Therefore, command line application is developed for real-time testing with webcam in the local machine. The test and result of this classification system will be described in Chapter 4.

REFERENCES

[1] Abien Fred Agrap, “ReLU Activation Function”, De La Salle University, Manila, Metro Manila, Philippines. Deep Learning using Rectified Linear Units ReLU (accessed March, 2018)

[2] Adrian Rosebrock, “ImageNet: VGGNet, ResNet, Inception, and Xception with Keras” Imagenet-vggnet-resnetinception-xception-keras (accessed March 20, 2017).

[3] Binghui Chen, “Softmax Activation Function”, School of Information and Communication Engineering (SICE), Beijing University of Posts and Telecommunications (accessed July, 2017).

[5] David H. Hubel, and Torsten Wiesel, “Simple Cells and Complex Cells” presented at the Harvard University, Cambrigde, MA, USA, 1959.

[6] Dr. Kunihiko Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition”, NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan, Biol. Cybernetics 36, 193 202 (1980).

[7] Jaspreet, “History of Neural Network”, Data Scientist, Paris, (published in Criteo R&D Blog, Oct 6,2020).

[8] Jason Brownlee, “CNN training” (accessed May 17, 2019).

[9] Sumit Sahin, “VGGNet vs ResNet”, Senior at NSIT, New Delhi. Ex SDE Intern at Western Digital (accessed May 3, 2020)

[10] Yann LeCun et al., “Gradient-Based Learning Applied to Document Recognition”, AT & T Bell Laboratories, Holmdel, N. J. 07733.