VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



**A MODEL FOR CLASSIFICATION OF WASTE BASED ON CNN AND IMAGE PROCESSING TECHNIQUES**

By

PHAM CONG TUAN

A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of   
Bachelor of Information Technology/Computer Science/Computer Engineering

Ho Chi Minh City, Vietnam  
Year 2022

**A MODEL FOR CLASSIFICATION OF WASTE BASED ON CNN AND IMAGE PROCESSING TECHNIQUES**

APPROVED BY: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,   
Nguyen Van A, Ph.D., Chair (*Example*)   
\*(*Type Committee names beneath lines*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
(*Typed Committee name here*)

THESIS COMMITTEE   
(Whichever applies)

# ACKNOWLEGMENTS

It is with deep gratitude and appreciation that I acknowledge the professional guidance of Dr. Nguyen Van Sinh. His constant encouragement and support helped me to achieve my goal.

My gratitude goes to the other members of the laboratory, Mr. Pham Quoc Son Lam and Mr. Tran Thanh Tung. Their technical help and good humor made these years a great learning experience. I am grateful to the faculty of the School of Computer Science of the International University, especially Dr. Nguyen Van Sinh for his support since the beginning of my studies. Gratitude is also expressed to the members of my reading and examination committee.

*Note: Paper A4, Top: 2.5cm; Bottom: 2cm; Left: 3cm; Right: 2cm*

**Table of Contents**

[ACKNOWLEGMENTS 4](#_Toc123900880)

[LIST OF FIGURES 7](#_Toc123900881)

[LIST OF TABLES 8](#_Toc123900882)

[ABSTRACT 9](#_Toc123900883)

[CHAPTER 1: INTRODUCTION 10](#_Toc123900884)

[1.1. Background 10](#_Toc123900885)

[1.2. Problem Statement 10](#_Toc123900886)

[1.3. Scope and Objectives 11](#_Toc123900887)

[1.4. Assumption and Solution 12](#_Toc123900888)

[1.5. Structure of thesis 13](#_Toc123900889)

[CHAPTER 2: LITURATURE REVIEW/RELATED WORK 15](#_Toc123900890)

[2.1. Classification of TrashNet Dataset Based on Deep Learning Models [3] 15](#_Toc123900891)

[2.1.1 Details 15](#_Toc123900892)

[2.1.2 Review 16](#_Toc123900893)

[2.2. WasteNet: Waste Classification at the Edge for Smart Bins [4] 17](#_Toc123900894)

[2.2.1. Details 17](#_Toc123900895)

[2.2.2. Review 19](#_Toc123900896)

[2.3. Exploring Features in a Bayesian Framework for Material Recognition [5] 19](#_Toc123900897)

[2.3.1. Details 19](#_Toc123900898)

[2.3.2. Review 21](#_Toc123900899)

[2.4. SpotGarbage: Smartphone App to Detect Garbage Using Deep Learning [6] 21](#_Toc123900900)

[2.4.1. Details 21](#_Toc123900901)

[2.4.2. Review 24](#_Toc123900902)

[CHAPTER 3: METHODOLOGY 26](#_Toc123900903)

[3.1 Overview 26](#_Toc123900904)

[3.2 User requirement analysis 27](#_Toc123900905)

[3.3 System Design 28](#_Toc123900906)

[3.3.1 Data Augmentation 29](#_Toc123900907)

[3.3.2 Transfer Learning 30](#_Toc123900908)

[3.4 User Interface and Packaging 31](#_Toc123900909)

[3.5 Training Setup 32](#_Toc123900910)

[CHAPTER 4: IMPLEMENT AND RESULTS 33](#_Toc123900911)

[4.1. Implementation 33](#_Toc123900912)

[4.1.1. Prerequisite 33](#_Toc123900913)

[4.1.2. Challenges 34](#_Toc123900914)

[4.2. Results 35](#_Toc123900915)

[CHAPTER 5: DISCUSSION AND EVALUATION 38](#_Toc123900916)

[5.1. Discussion 38](#_Toc123900917)

[5.2. Comparison 38](#_Toc123900918)

[5.3. Evaluation 39](#_Toc123900919)

[CHAPTER 6: CONCLUSION AND FUTURE WORK 40](#_Toc123900920)

[6.1. Conclusion 40](#_Toc123900921)

[6.2. Future work 40](#_Toc123900922)

[REFERENCES 41](#_Toc123900923)

[APPENDIX 43](#_Toc123900924)

# LIST OF FIGURES

Figure 1. Model workflow for training and testing 26

Figure 2. Visualization of waste production in US from 1960 to 2018 27

Figure 3. General structure of a block in MobileNet V2 model 29

Figure 4: Sample image augmentation 30

Figure 5: MobileNet V2 model fine-tuning process 31

Figure 6. User interface with PyQt5 32

Figure 7. Model training on Google Colab 33

Figure 8. Loss and Accuracy per epoch 35

Figure 9. Loss per epoch graph 35

Figure 10. Confusion matrix on test dataset of the model 37

# LIST OF TABLES

Table 1. Model’s performance on test dataset………………………………………………34

Table 2. Comparison between the proposed model, GarbNet and WasteNet……………….39

# ABSTRACT

Environment-related problems have become increasingly hazardous towards human and pose a risk to the ecosystem in the 21st century, one of which is the ever-increasing volumes of waste (excessive waste), mainly municipal solid waste. While recycling and other basic methods to treat waste have been introduced to daily life, the conversion of waste back to its constituent parts (known as recycling) is still in its infancy due to several difficulties relating to the technology and funding.

In this prethesis, I propose a machine learning model using convolutional neural networks and image processing techniques to help classify waste through a captured photo. The proposal includes data gathering, preprocessing and augmentation techniques, and model training with CNN. The model achieves 95% accuracy while requiring minimal time to train by implementing transfer learning and fine-tuning on a pretrained MobileNet V2. It can be implemented together with additional components to form a system to assess the recyclability and set priorities for waste packs at waste processing plants.

# CHAPTER 1: INTRODUCTION

## Background

Ever since the first industrial revolution in England from roughly 3 centuries ago, environmental problems have really become generally more and more severe, according to the warning of well-known scientists. By definition, environmental issues are effects of human activity on the biophysical environment specifically, most of which are for malicious intents and purposes that cause harmful effects called environmental degradation, such as the depletion of resources, decrease of quality of air, water and soil, the destruction of ecosystems and natural habitat, the extinction of wildlife and pollution in a subtle way. Nowadays, environmental degradation kind of is listed as one of the ten threats officially definitely cautioned by the “High-level Panel on Threats, Challenges and Change” of the United Nations (UN) in a subtle way. [1]

Besides some widely known difficulties such as global warming, deforestation and water pollution, there is a lesser-known environmental problem of the ever-increasing volumes of waste (excessive waste), which is mainly caused by poor waste management and over shopping as the global population and living standards rise. To minimize the consequences of this problem, I need to reduce the waste volume as well as increase our recycling capability.

In practice, our waste would be commonly divided into “dry” or “wet” category. Dry waste generally includes wood and related products, metal, plastic as well as glass. Wet waste typically refers to organic waste, food, together with vegetable products. Within this article scope, I only concern about the classification problem that belongs to municipal dry waste - solid waste that is produced by the local community.

## Problem Statement

The abundance of trash leads to several severe environmental consequences for that municipal solid waste is listed as one of the main sources for human-caused methane emissions. Massive landfills are driving people from their homes, causing unpleasant odor and irreversible pollution. Apart from the immediately noticeable problems, municipal solid waste is also the culprit for potential global issues such as the exportation of trash from rich countries to poorer nations.

For municipal solid waste, the recyclables usually consist of glass, plastic, paper, carton paper (cardboard), metal and cloth (fabric). Cardboard and paper belong to two different categories because they have different recycling process due to their own characteristics. Distinct types of plastic also require specific procedures. Currently, the recycling efficiency is limited by both technology and funding, thus cannot keep up with the rapid increase of waste over time. Usually, the recycling process consists of three main steps: collecting/processing, manufacturing, and reselling. The first step is the hardest of the three, with the challenge of sorting and treating each object so that I can convert it back to its smaller constituent parts. In developing countries such as Vietnam, the work is done mostly by hand. Household trash packs are loaded on a truck, which are then delivered to the city landfill. At the dump, the trash is first sorted by large filters and magnet, then the workers manually sort the pieces to collect the recyclable scrap to resell at cheap price. This poses an elevated risk to the health of the workers, as well as being a bottleneck to the process overall efficiency since the first stage of the recycling limits the two sequential steps. Compared to wealthier countries, local processing plants in poorer nations are often limited, because of the high initial investment as well as the technology difficulty, with most of the currently available plants only use processed trash as fuel to burn in order to produce energy. [2]

To improve the efficiency of the first step in the recycling process while overcoming the aforementioned difficulties, international organizations such as the United Nations have encouraged the act of waste sorting from its source (the most well-known one being “3 bins for different types of trash” and similar systems). While it may serve as a long-term solution, it seems to be not enough for the immediate drastic measure. Furthermore, the application of it seems vague, resource demanding and is possible only in big cities. Other than this proposal, there have been other small-scale models and campaigns to help leverage recycling knowledge, as well as recognizing and classifying objects including trash, but they all have some shortcomings such as costly training, low performance, smaller scales,... The situation calls for a systematic and more applicable method to solve the recycling challenge – which would be proposed later in this article – involves using technology advances to classify the “green” objects in the waste mass to be recycled.

## Scope and Objectives

Municipal solid waste recycling is the key to reducing excessive waste, which is a critical problem in several countries, especially developing nations and high population areas. In this paper, the proposed model is expected to be able to classify six categories of objects using available photos of trash with over 85% accuracy, hence serve as an early stage in waste classification, with the objects being different in type, size and color only. While the scope of this paper only includes the model and datasets, it has the potential to be scaled up/scaled down to suit other projects’ needs (add more categories, implement in a large-scale system, etc.)

The model utilizes multi-class classification Convolutional Neural Network (CNN) to make it possible to recognize and classify our objects among the waste mass. CNN, also known as Shift Invariant or Space Invariant Artificial Neural Networks - SIANN has been used in practice for a long time. CNN is a class of artificial neural network that is commonly applied to analyze visual imagery. Most CNN models are based on the shared-weight architecture of the convolution kernels or filters that slide along the input features to provide translation-equivariant responses known as feature maps.

In this proposal, I fine-tune a pretrained MobileNet V2 with Python programming language and utilizes PyTorch library for CNN. Due to the small size of the dataset, there is the necessity for data augmentation. This mostly includes scaling, rotating, shearing, controlling brightness, channel shuffling, adding Gaussian blur and noise to increase the diversity of the dataset. I also make a simple user interface with PyQt5 to make the model more user-friendly. The expected outcome is a high accuracy model with the ability to classify six types of household solid waste.

## Assumption and Solution

This article only proposed a model for classifying trash only, while outside of the paper’s scope, it can be improved and retrained to be used in waste processing and recycling plants, or help the workers sort the trash more efficiently. As stated, the proposed CNN model would utilize transfer learning with MobileNet V2 to be able to recognize and classify the objects from image inputs. To be more specific, this model classifies and recognizes six types of trash only (paper, plastic, cardboard, metal, glass, and fabric) using captured photos as the input. There are other possible scenarios that this model may help, such as assessing the recycle priority of different portions of trash through captured photos or recognizing objects for trash auto sorting system using a crane machine. The model is assumed to be applied in a processing plant, or a local trash collection area to help classify the objects and can be implemented together with other assisting technology (object detecting, mechanical robot arms…).

The paper assumes that the objects to be classified are not broken and smashed into small pieces (they need to retain their shapes to some extends to be recognized successfully). For example, glass bottles are not smashed into small pieces, and papers are not shredded into tiny bits. Deformation and different lighting conditions are acceptable.

For the train dataset, through data augmentation, the images vary to reflect real life conditions. They are of different colors and shapes and under different lighting conditions. The objects are not cleaned to make their conditions similar to their counterparts in the trash bin.

## Structure of thesis

CHAPTER 1: INTRODUCTION

1.1. Background

1.2. Problem Statement

1.3. Scope and Objectives

1.4. Assumption and Solution

1.5. Structure of thesis

CHAPTER 2: LITURATURE REVIEW/RELATED WORK

2.1. Classification of TrashNet Dataset Based on Deep Learning Models

2.1.1. Model details

2.1.2. Review

2.2. WasteNet: Waste Classification at the Edge for Smart Bins

2.2.1. Model details

2.2.2. Review

2.3. Exploring Features in a Bayesian Framework for Material Recognition

2.3.1. Model details

2.3.2. Review

2.4. SpotGarbage: Smartphone App to Detect Garbage Using Deep Learning

2.4.1. Model details

2.4.2. Review

CHAPTER 3: METHODOLOGY

3.1. Overview

3.2. User requirement analysis

3.3. System Design

3.3.1. Data Augmentation

3.3.2. Transfer Learning

3.4. User Interface and Packaging

3.5. Training Setup

CHAPTER 4: IMPLEMENT AND RESULTS

4.1. Implementation

4.1.1. Prerequisite

4.1.2. Challenges

4.2. Results

CHAPTER 5: DISCUSSION AND EVALUATION

5.1. Discussion

5.2. Comparison

5.3. Evaluation

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1. Conclusion

6.2. Future work

APPENDIX

AP.1. Model source code and dataset

AP.2. Overview of MobileNet V2 architecture

AP.3. Detailed architecture of MobileNet v2

# CHAPTER 2: LITURATURE REVIEW/RELATED WORK

## Classification of TrashNet Dataset Based on Deep Learning Models [3]

Classification of TrashNet Dataset Based on Deep Learning Models, which is one of the most relevant articles to this paper’s problem, is a publication by Rahmi Arda Aral et al at Gazi University. It provides valuable insights about popular models’ performance when it comes to tackling the challenge of recognizing recyclable garbage. The goal of the article is to compare and give commentary on the efficiency of each model’s approach. The authors draw the conclusion that Adam optimizer provides a better test accuracy in comparison to its Adadelta counterpart, with both the fine-tuned DenseNet121 and InceptionResNetV2 being two models with the best accuracy.

## Details

To be more specific, the article compares between Densenet121, DenseNet169, InceptionResnetV2, MobileNet and Xception architectures with Adam and Adadelta as the optimizer using the TrashNet dataset. The dataset in use contains images in which a single object is presented on a clean white background, containing six subclasses of paper, glass, plastic, metal, carton paper and garbage. The different exposure and lighting selection for each photo help increase the variations in the dataset. In short, deep learning is used to classify images into six categories of the aforementioned rubbish classes.

Regarding the experimented models and architectures, Xception architecture is based on depth wise separable convolution layers extensively. Mapping of cross-channels connections and spatial connections in the property maps of convolutional neural networks can be thoroughly set apart in this architecture. Xception is a stronger version of the underlying Inception architecture, which has thirty-six convolutional layers building the property extraction base of the network. All convolutional layers are structured into fourteen modules, all of which have linear remnants relation around them, outside of the first and the last modules. Briefly, this architecture is quite easy to be described and changed due to its characteristic being a linear heap of depth wise separable convolution layers with residual relations.

MobileNet V1 model consists of in-depth separable convolutions, which is used initially in the Inception model. Its structure was built on depth-wise separable convolutions, with the first layer being full convolution. MobileNet performs depth-wise separable convolution after the full convolution, thus a high accuracy rate can be achieved with a small number of hyperparameters.

DenseNets contains very short connections between input and output layers, which performs very efficiently as a CNN structures. It has an improved flow of information and gradients throughout the network in order to produce consistent improvement in accuracy with growing number of parameters, without much performance degradation or overfitting. DenseNets requires considerably fewer parameters and less computation to attain novel performances.

Inception-v4 is the advanced version of the Inception-v3 and a hybrid of inception modules and residual connections. In Inception-v4, batch normalization takes place on the top of traditional convolutional layers. Due to this property, inception block size has been increased. Generally, they are deep convolutional networks which have reliable performance on image recognition performance. Inception architecture was proven to perform well at relatively low computational costs.

For the training experiments of DenseNet models, the batch size was selected as eight. As for other training experiments, the batch size was selected as thirty-two. For DenseNet, Xception and MobileNet training experiments, the input size was selected as 224x224. For the Inception-ResNet-V2 model training experiments, the input size was selected as 299x299.

In the tier of one hundred epochs, the authors come to the conclusion that both InceptionResNetV2 and DenseNet121 perform at 89% accuracy, while Xception, DenseNet169 and MobileNet have the accuracy of around 80%. In the case of increasing the training epochs to 150, DenseNet121 and MobileNet have the highest accuracy, which is at 84%. On the other hand, Xception and DenseNet169 only achieve the accuracy of 82% and 78% respectively. Overall, the applied deep learning models achieved more than 76% test accuracy. The Inception-ResNet-V2 model with one hundred epochs achieved an 89% of test accuracy. Densenet-169 model with 150 epochs, I achieved an 84% of test accuracy. The highest test accuracy with MobileNet model is 84%, with 150 epochs. After fine-tuning, the most successful test accuracy rates Ire achieved with the fine-tuned Densenet-121 and Densenet-169 models. In the selection of the optimizer, Adam and Adadelta optimizers Ire tried with one hundred epochs in InceptionResNetV2 model. The authors also see a higher test accuracy was obtained in the Adam optimizer.

## Review

The paper is a comparison and evaluation between different architectures such as Densenet121, DenseNet169, InceptionResnetV2, MobileNet and Xception, as well as two popular optimizers (Adam optimizer versus Adadelta optimizer). The paper provides detailed insights about each architecture when trained with TrashNet dataset in a straightforward way. While the scope of the paper only includes trash classification task, the authors provide valuable perspicuity for each paper’s method. The testing method which is implemented by the authors is sufficient and accurate enough. About the dataset, it is not exceptionally large and diverse, but is compensated by the image augmentation preprocessing.

## WasteNet: Waste Classification at the Edge for Smart Bins [4]

WasteNet: Waste Classification at the Edge for Smart Bins is a proposal by Gary White et al at Trinity College Dublin - Ireland. WasteNet is a waste classification model based on convolutional neural networks that can be deployed on a low power device at the edge of the network. The authors suggest the idea of automated waste classification at the edge, which allows for fast intelligent decisions in smart bins without the need to access to the cloud. Similar to other waste classification models, waste is put into one of the six categories: paper, cardboard, glass, metal, plastic, and other trash. According to the paper, the model can achieve up to 97% prediction accuracy on the test dataset.

## Details

With the increase in urbanization and economic development around the world, which causes excessive waste generation, the need for waste incinerators becomes more and more noticeable. Incinerators are machines that are used to deal with the waste problem by burning trash to produce energy. There are several drawbacks of this approach, including the concerns over fine particle emission, which can lead to a number of associated health risks as well as contributing to global warming. In the scientific journal, the authors believe that waste classification should better happen at the initial stage where the waste is being disposed of in order to avoid potential recyclable materials becoming contaminated, thus propose the idea of WasteNet model implementation for smart bins.

The model utilizes machine learning and pattern recognition using a deep neural network called WasteNet. The data flow involves using an installed camera to take pictures of objects inside the bin sections and feeding those photos to the model, which would select the correct container for the waste to be disposed in to ensure that it would be recycled properly. To solve the problem of energy required to run, the proposed bins are fitted with solar panels, allowing low power edge devices such as Nvidia Jetson Nano (a low-power tiny computer) to run the model locally.

The design of the WasteNet neural network model uses transfer learning to leverage knowledge from a source task - models trained for a general image classification task on the ImageNet dataset with the ratio for train : validation : test being 50 : 25 : 25 splits, and focus on inductive transfer learning where the source and target have the same distribution or are in the same domain but the tasks that they are required to perform are different. The images from the dataset are augmented to improve the diversity and accuracy, as well as to prevent overfitting on the local training data. The list of transformation operations (augmentation) on the images includes translation, zooming, shearing, rotation and expansion. The authors also implement transfer learning to let the model gain several benefits such as improving baseline performance, speeding up overall model development and training time, with the aim to gain overall improved model performance compared to the traditional way of building the model from scratch. This is especially important and beneficial in deep learning, because training models can take a very long time and require a lot of resources.

The model uses VGG-16 model with some tweaks such as freezing all the blocks of convolutions layers and the flattening layer (which basically means fix weights and don’t train) and hybrid tuning (gradually unfreeze each layer starting from the top and update the weights so that the weights of the last few layers of the network are updated and trained as well as the fully connected layers at the end of the model). By freezing technique, the authors only update the fully connected classifier block at the end of the model. This allows the new model to transform the image from a new domain task into a large dimension vector based on hidden states, thus allowing the extraction of features from a new domain task using the source domain. This is a widely used methods of transfer learning for deep neural networks.

With hybrid tuning, there are basically two stages, in which the first pre-training stage the base network is used as a feature extractor by freezing the lower layers of the network and only updating the weights of the top layer. Once the loss function begins to stabilize and the network has reached a prominent level of accuracy with the new top layer of the network, the remaining layers of the network are gradually unfrozen. The remaining layers of the network are trained using discriminative layer training, where a different learning rate is applied to each layer. This allows for a lower learning rate to the low-level layer representations and adapt the weights of the higher-level layers faster as they contain more domain specific information. With the gradual unfreezing instead of fine-tuning all the layers at once, the authors can reduce the risk of catastrophic forgetting.

The results of the experimentation have provided a number of interesting insights for future waste classification. With the high accuracy of 97%, the authors are able to prove the potential of these methods. As the model can be deployed on devices at the edge of the network such as a Jetson Nano, it allows decision making and artificial intelligence at the edge of the network.

## Review

The paper provides an interesting way to detect and reduce the waste from the initial stages using smart bins, with a quick revision of every transfer learning category. The most noticeable improvement in comparison to previous works on this topic is the proposed “hybrid tuning”, which combines the benefits of freezing technique during feature extraction and fine-tuning. About data augmentation, apart from other common techniques (shearing, rotating, flipping, translation,…), the authors introduce the expansion of the photo by filling new pixels with their nearest surround pixels. For the dataset, the authors work with TrashNet – a popular waste dataset for machine learning models, which includes six classes: plastic, paper, glass, metal, cardboard and other. The tuned model can achieve an astonishing accuracy of 97% with low computation cost as reported in the paper, according to the authors.

## Exploring Features in a Bayesian Framework for Material Recognition [5]

Exploring Features in a Bayesian Framework for Material Recognition is an academic collaboration by Ce Liu, Lavanya Sharan, Edward H. Adelson and Ruth Rosenholtz. The aim of the paper is to identify different material categories such as glass, metal, fabric, plastic, or wood from a single image of a surface using computers. The paper proposes an augmented Latent Dirichlet Allocation (aLDA) model to combine the features of material appearance under a Bayesian generative framework and learn an optimal combination of features. The general strategy is to use a rich set of low and mid-level features that capture various aspects of the surface of materials for the model to be able to distinguish between them. Experimental results show that this model performs material recognition function reasonably well on a challenging material database.

## Details

Material recognition is an important aspect of visual recognition, even though we do not often think about it on a regular basis. Unlike other visual recognition tasks in computer vision, while trying to differentiate the categories, the computer cannot solve the challenge of picking good, reliable features that actually help. Therefore, it is valuable to build a visual recognition system that can infer material properties from images using visual appearances of a surface. Previous experiments often use several factors to distinguish the materials including the illumination conditions, the geometric structure of the surface sample at several spatial scales, and the surface reflectance properties, often characterized by the bidirectional reflectance distribution function (BRDF) and its variants. In the proposal, the authors focus on recognizing high-level material categories, such as glass, metal, fabric, plastic, or wood, instead of explicitly estimating reflectance properties using several low-level and middle-level features to characterize various aspects of material appearance. In addition to well-established features such as color, jet and SIFT, the authors also introduce several new features, such as curvature of edges, histogram of oriented gradient (HOG) feature along edges, and HOG perpendicular to edges.

The chosen features include color, texture and micro-texture, outline shape, glossiness, and opacity. The authors propose a combination of techniques to build an effective material recognition system. For feature quantization and concatenation, they use k-means algorithm to cluster the instances of each feature to form dictionaries and map image features into visual words. They also implement latent Dirichlet allocation (LDA) and augmented latent Dirichlet allocation (aLDA), with the probability density function aiming for maximizing the recognition rate and minimizing the error. By quantizing the features into dictionaries, the authors literally convert an image into a bag of words and use latent Dirichlet allocation (LDA) to model the distribution of the words. By allowing topics to be shared amongst material categories, the model concatenates dictionaries from various features and learn the optimal combination of the features by maximizing the recognition rate. That is the reason LDA is able to learn clusters of visual words that characterize dissimilar materials, thus the name augmented LDA.

For the dataset, the authors use the self-developed Flickr Materials database. The database contains ten common material categories - fabric, foliage, glass, leather, metal, paper, plastic, stone, water and wood, each has 100 color photographs from Flickr.com, including 50 close-ups and 50 object-level views. All images have 512 × 384 pixel resolution and contain a single material category in the foreground. These images capture a wide range of appearances within each material category. With the dataset of ten material categories, the authors aim to secure a boost in performance from the single best feature (SIFT, 35.4%) to the best feature set (color + SIFT + edge slice, 44.6%), which is due to the aLDA model that augments visual words. In total, the aLDA model contributes to the boost in performance from 37.4% to 44.6%. On the other hand, it is interesting to find out that augmenting more features decreases the overall performance.

While the model possesses some disadvantages in comparison to newer models (fabric is often misclassified as stone, leather misclassified as fabric, plastic misclassified as paper, …), the results are not surprising because there are certain commonalities between leather and fabric, plastic and paper, as well as metal and glass. The diversity and wide range of the Flickr Materials Database also contributes greatly to the problem the model has to solve, proving the dataset to be a challenging benchmark for material recognition. The authors claim that the proposed feature set and computational framework have constituted the first attempt at recognizing high-level material categories in the wild.

## Review

The paper focuses on methods for extracting specific material’s features, while proposing a variation of LDA algorithm (augmented LDA). The database this time includes ten categories with images collected from Flickr, with increased difficulty and diversity. The authors go into details about choosing the features for the material recognition task and introduce a wild variety of supporting algorithms such as LDA, aLDA, Varma Zisserman and SIFT. With the newly proposed aLDA, the model achieves state-of-the-art accuracy at publish time despite the challenging dataset. Even though there are better models with higher accuracy nowadays, the 2010 paper by Ce Liu et al is still cut for a good reference.

## SpotGarbage: Smartphone App to Detect Garbage Using Deep Learning [6]

SpotGarbage is a project by Gaurav Mittal et al at Indian Institute of Technology Ropar Rupnagar, India. With the aim of engaging citizens to track and report on their neighborhoods, the team presents a convenient smartphone app called SpotGarbage, which detects and coarsely segments garbage regions in a user-clicked geo-tagged image. The app utilizes the proposed deep architecture of fully convolutional networks for detecting garbage in images. The model has been trained on a newly introduced Garbage In Images (GINI) dataset and achieves a mean accuracy of 87.69%. The paper also proposes optimizations in the network architecture, which results in a reduction of 87.9% memory usage and 96.8% prediction time with no loss in accuracy, facilitating its usage in resource constrained smartphones.

## Details

In order to engage the citizens to check the garbage, the authors aim to equip them with an easy-to-access, prompt, and reliable medium through which they can report the presence of garbage in their vicinity. With the development of technology, smartphones become increasingly popular and are well equipped with reasonably powerful cameras. In the past, there have been mobile app-based initiatives that allow people to report such menace in their neighborhood by uploading images of garbage. However, these solutions rely on humans for identifying garbage in an image making them impractical for large scale use. Rather than uploading the full image, the proposed model aims to process the data and only send essential parts of the information. Processing the image on the phone also helps to elicit user-feedback for validating the machine learning model for garbage detection. Moreover, segmenting the garbage region in the image may allow determining the severity of garbage.

The paper takes the first step towards such a solution by introducing an Android app, SpotGarbage, which employs a Convolutional Neural Network (CNN) called GarbNet to automatically detect and localize garbage in unconstrained real-world images. The input to the CNN is an arbitrary sized image and the output is a coarse-grained segmentation of the image highlighting garbage patches. Further, GarbNet is optimized to perform in a resource constrained environment. This facilitates its deployment on a ubiquitous mobile platform. The paper also introduces a new annotated dataset, called Garbage In Images (GINI), using Bing Image Search API to crawl the data. The dataset is a collection of several in-the-wild images containing garbage. Each image is also annotated with perceived levels of severity and biodegradability. Queries (such as roadside garbage, market waste) Ire used to obtain a diverse set of images containing garbage, resulting in a compilation of 2561 images, out of which 956 images Ire obtained through garbage related queries. The images from the GINI dataset are processed to generate fixed-sized patches. Garbage images that had regions not entirely garbage is excluded. The rest of the images are first divided into 5 stratified folds to avoid correlation among the patches across the folds. The images in each fold are cropped to generate patches of different sizes, allowing the model to adapt to multiple scales and different levels of contextual information. The patch sizes Ire chosen to be 10%, 20%, 40% and 80% with stride as 9.1% of the smaller image dimension so as to perform Poisson-disk image subsampling. The patches are further oversampled by performing random rotations between [0,2π]. This increases the size of the training set, which helps to prevent overfitting. More importantly, it also makes the model rotation invariant. In total, this generates a set of 500,000 patches.

Afterall, the goal of the paper is to propose a method to detect the presence of garbage in an image and also approximately demarcate the regions in the image that correspond to garbage. This goal is achieved by training a model using patches extracted from images. The final prediction for a test image is obtained by again extracting patches and combining their predictions. Inspired by AlexNet, the objectification of garbage allows the weights of GarbNet to be initialized using the mentioned pre-trained model. By doing so, the model is able to exploit the rich hierarchy of already learned representations making it achieve a better generalization. The architecture has been modified to perform binary classification. The two fully connected layers of the network, following the five convolution layers and containing 4096 neurons each, are optimized to have 512 and 256 neurons, respectively. The supervised fine-tuning of GarbNet is conducted via 5-fold stratified cross validation with training and validation sets each consisting of approximately 380,000 and 20,000 patches, respectively. The initial learning rate of 1×10^−3 is reduced by a factor of 4 after every 25,000 iterations. The momentum is 0.9 and weight decay is 5×10^−5 . Further, the patch samples are randomly mirrored and cropped while training to prevent overfitting.

In order to be deployed on smartphones, the model needs to yield a quick response and have a low memory cost. Using a regular CNN with naïve sliding window approach results in a lot of redundant computation due to the overlapping receptive fields, which is not efficient. To figure out a way to overcome the difficulty, the authors turn to convolutional layers of CNN, which are translation invariant. They operate on local input regions and are agnostic to the spatial size. This characteristic can be exploited to expedite the feedforward computation, by allowing the entire image to be processed in a single pass instead of giving individual overlapping patches as input to the network. By converting the fully connected layers into convolutional layers, the authors make the architecture convolve with the entire image in one go. This transforms the regular CNN into a fully convolutional network without any change in the model size. The computation gets highly amortized reducing the time over the naïve sliding window approach by a factor of twelve.

GarbNet model produces a classification map for all the patches in a test image simultaneously instead of a single classification and returns positive if at least one image patch is classified as garbage. The size and overlap (stride) are varied to determine their optimal values. The basis of incorporating a Local Response Normalization (LRN) layer in the Alex network is empirical. Therefore, another optimization step would be to remove these layers reducing the prediction time. Experiments are conducted to rule out adverse effects of removing the LRN layer from the network on the model accuracy, with a back propagation network that is trained using a large number of features extracted from the patches being used as the baseline. The feature extractors used are the state-of-the-art image descriptors commonly found in the literature. The patches are first warped to a fixed dimension of 256×256 to extract feature vectors of constant size. The warped patches are convolved with a filter bank of Gabor wavelets having four wavelengths and 4 orientations.

The result is smoothened with a Gaussian filter with width equal to half the wavelength of the corresponding filter. This is followed by an application of PCA on each pixel to extract the direction with the maximum variance. The output is down sampled to produce 16,384 features. Further, HOG features are extracted using a cell size of 16×16 taking the maximum of the gradient magnitude across the 3 color channels, thus generating another set of 7,936 features. Finally, the histograms corresponding to RGB, HSV and Lab color spaces, with 25 bins for each channel, for every 50% overlapping 64 × 64 subsample is appended to generate a feature vector of size 35,345 in total. Thus, the feature vector provides an extensive characterization of an image patch. The learning rate and number of hidden layer nodes of the back propagation network are fine-tuned using cross validation experiments.

To implement the model, the authors introduce the SpotGarbage app that can automatically detect and localize regions containing garbage in user-clicked unconstrained geo-tagged real-world images. The app utilizes the proposed fully convolutional network, GarbNet for coarsely segmenting image regions containing garbage. The GarbNet model has been optimized for resource constrained smartphones, by reducing the number of network parameters and removing local response normalization layers from the network. The experiments conducted on the new GINI dataset suggest a significant reduction in the space and time requirements of the application with no considerable effect on the accuracy. The model is able to classify images with an accuracy of 87.69%, with the patch size and overlap parameters of the GarbNet model set based on 5-fold cross validation experiments. The probability of classifying a patch as garbage is set to 0.99 to ensure maximum confidence in prediction and minimum false positives. The authors also successfully reduced the memory usage of the model by bringing the size down to only 28 MB. This facilitates the deployment of the model on a wide spectrum of low-end smartphones and other IoT devices where memory space is a premium.

## Review

SpotGarbage is a proposal by Gaurav Mittal et al, which includes a smartphone app and a deep learning model called GarbNet. The authors state that SpotGarbage should process the images to feed to the model, and only send the essential information to the cloud. The article goes deep into optimization with a custom architecture. Generally, the model focus more on detecting trash in the photos instead of trying to classify the trash into categories. With the advantage of being light weight and easy to use, the proposal is a good innovation for combating the excessive production of waste.

# CHAPTER 3: METHODOLOGY

## Overview

Convolutional Neural Network (CNN) is a subset of deep learning, which is considered as machine learning but with multiple hidden layers. It is a type of neural network model which allows working with the images and videos, usually applied in computer vision. It takes the image’s raw pixel data, trains the model, then extracts the features automatically for better classification, compared to the more basic and traditional SVMs. In this paper, I utilize transfer learning and fine-tuning techniques on MobileNet V2 – a CNN by Google to classify objects.

Diagram

Description automatically generated

Figure 1: Model flow for train/val and testing

The workflow starts with collecting the data and labeling them. Then I process the data (augmentation, data cleaning…) and feed the preprocessed data to the model in batches for the CNN to try to extract the features of each class by converting the images into tensors and passing them through different fine-tuned layers to filter the features. After that, the model’s performance is inspected and modified if needed, which is essentially making changes to the hyperparameters. When the model performance is stable enough, I stop retraining and get the output model with its weight. For the testing, I take unseen samples of the data to pass it to the model, and I get the predicted output as the result.

## User requirement analysis

In the fast-moving world, the need for trash processing is never ending. According to the EPA – United States Environmental Protection Agency, the amount of municipal solid waste has almost tripled in the US alone during the time from 1960 to 2018 [7]. In less developed countries where environmental problems are usually not in the spotlight, the problem can get much worse. On worldwide scale, it is expected to get to 2.12 billion tons globally, with the accumulated amount in 11 months being more than 1.9 billion tons.

Chart, line chart

Description automatically generated

Figure 2: Visualization of waste production in US from 1960 to 2018.

There have been emphasizes on cutting down on the waste production, with trash classification and recycling playing a key role in helping to reduce the waste mass. Nowadays, with the advance of technology, computer vision and machine learning can replace manual labors to do the task. Computers and robots are generally cheaper, safer, and more long lasting. The only problem with using computers is that the efficiency and accuracy is low, especially with the tainted, deformed, and dirty objects like trash.

In landfills, the need for computer-aid recycling system is huge, but usually the cost is too high compared to the benefits, especially in developing countries where environmental problem is not considered a major threat to humankind. Even if it were, there would be not enough resources to take care of the problem. With the advance of technology, the application of machine learning to help categorize the waste has been made more possible. It brings about cheaper price and less complicated setup, which are some of the primary concerns for poorer countries where excessive waste is the serious problem.

Apart from the price and setup process, the model would need to be possible to scale up/scale down and retrain for each country’s cultural fit purpose. A local processing plant for a small area would require less computation and uptime than full scale centralized city waste plant. The ability to resize the model or retrain it for more suitable accuracy-complexity tradeoff is important for trash classification model.

## System Design

For the CNN model, I use PyTorch and apply transfer learning with MobileNet V2. MobileNet is a common CNN designed and developed by Google for edge devices. After researching, I decide to use MobileNet V2 over other models because it is optimized for fast training time, low computing cost and maintenance. The model also supports transfer learning and is pretrained with ImageNet dataset. I implement techniques to fine-tune the trained model with its weight and extend it to make it relevant to our problem and improve the performance as well as prediction accuracy. [9]

MobileNet V2 consists of 2 types of blocks: stride=1 and stride=2. For each block, there is a convolutional layer of 1x1 with ReLU6, then a depth wise convolution of 3x3 and a convolution of 1x1 without non-linearity. In total, the base model has fifty-three convolution layers and 1 average pooling layer. A widely known feature of MobileNet V2 is the proposal of Inverted Residual block, also called MBConv block for efficiency reasons. By implementing Inverted Residual block, it uses an inverted structure, which is described by the number of I/O channels following a “narrow > wide > narrow” approach.

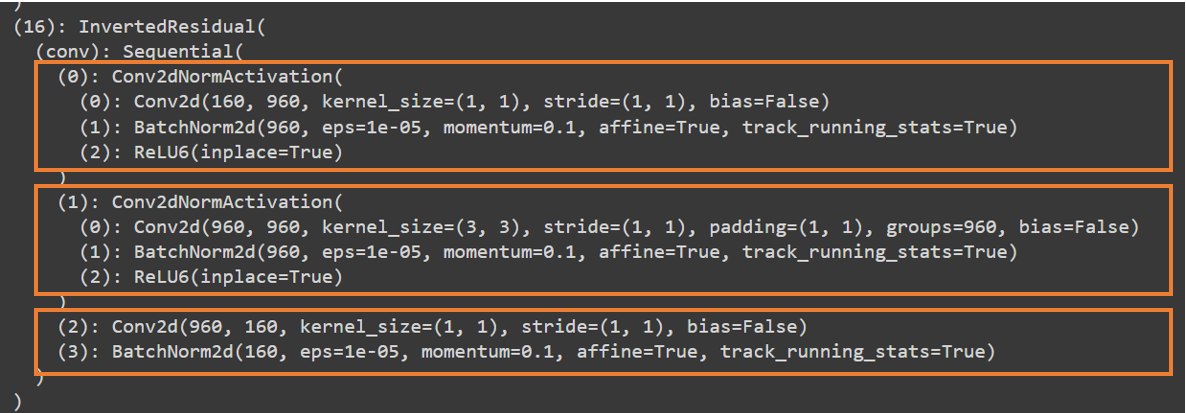


Figure 3. General structure of a block (The third layer is linear)

For finetuning, I freeze up to fifteen layers of the total 17 inverted residuals. The reason for freezing some layers is to avoid changing the information they contain (the weights) and to reduce training time. Usually, the first layers only detect general features such as edges, corners, and blobs of colors, which is not very helpful for our case. I opt for freezing most layers at once instead of the more gradually freezing/unfreezing since the accuracy and loss of our model are already acceptable.

### Data Augmentation

Since the dataset is quite small, data augmentation is applied for all classes. The images are transformed using “imgaug” Python library. [10] It is a library dedicated to image augmentation for machine learning experiments, especially in classification tasks. There are many options to choose from, and it is relatively simple to install and implement. The operations I chose include shearing, horizontal-vertical flipping, color grading, channel shuffle, blur addition, low noise addition and slight rotation. The p value for transform probability ranges from 0.3 to 0.6 randomly. Combined with OpenCV for reading/writing images, in the end, I have a total of around nine hundred samples for each class.

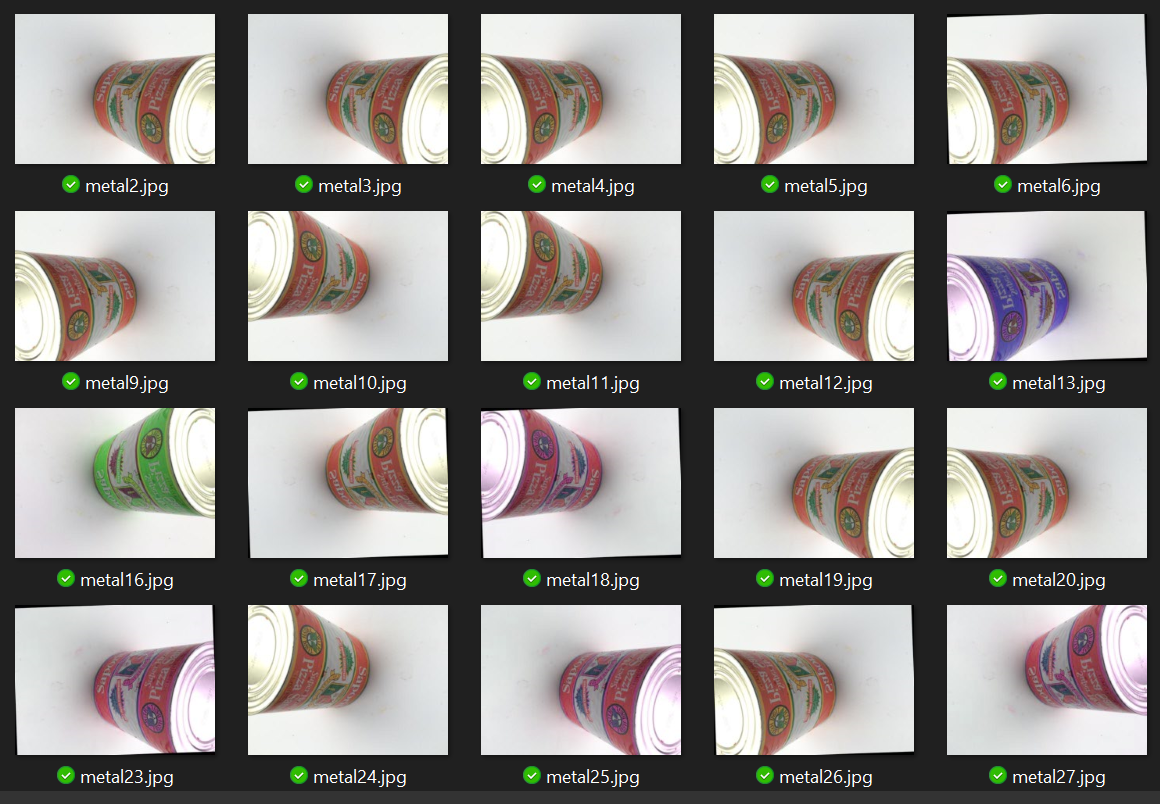


Figure 4: Sample image augmentation

### Transfer Learning

Transfer learning is the process of extracting and transferring the weights to another neural network. When a neural network is trained on a data, it gains knowledge of the classes, which is compiled as the weights of the network. In other words, it is a technique using a pre-trained neural network to solve a machine learning task that is similar to the problem the network was originally trained to solve. MobileNetV2 supports transfer learning, where it is previously trained with a large dataset (ImageNet), and then apply the knowledge to similar problems (trash classification).

Classifier function is a function which basically decide which class is suitable to put the data into. For our model, I change the classifier to the number of classes, which is six and add a dropout layer to reduce overfitting. I also freeze the first fifteen layers due to high similarities between the pretrained purpose and our model purpose. This significantly reduces the training time while maintaining high accuracy and low loss.

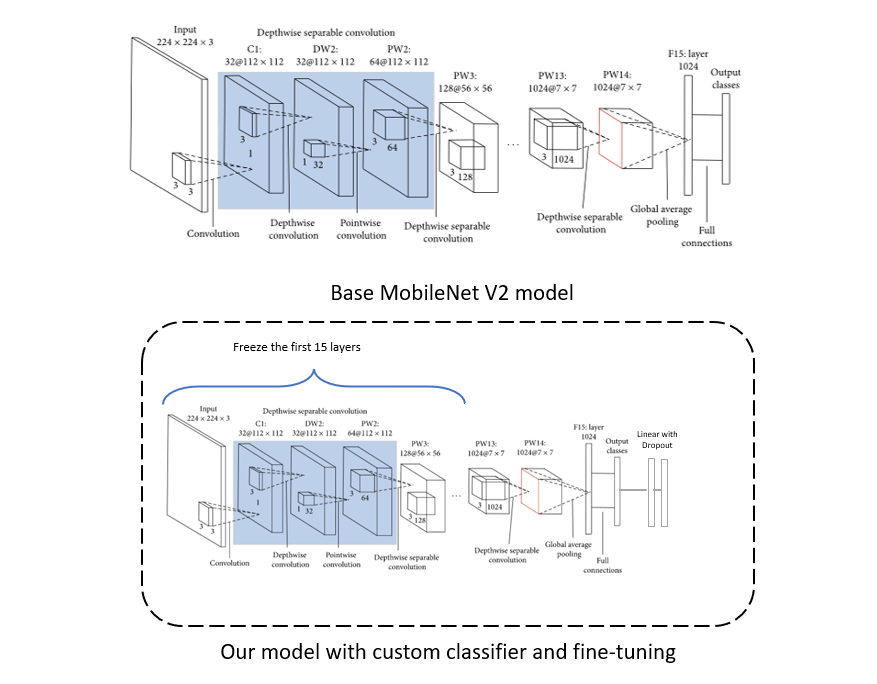


Figure 5: MobileNet V2 model fine-tuning process

### User Interface and Packaging

While the model is developed, I am required to have an interface for user to interact with the model. I use PyQt5 – a Python GUI library to build the user interface. Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems, while PyQt5 is a comprehensive set of Python bindings for Qt v5. PyQt5 supports rapid development and deployment of simple custom graphical user interface. [11]

The UI has basic functions such as displaying the output of the model, buttons to change photos and image preview panel. I use event slots and built-in components to create the UI and import the function to run the model from helper scripts. For easier usage, I make an executable file and package the scripts into a single zip file using PyInstaller. [12] This would be helpful for users without existing compatible Python/PyQt version or cannot run Python.

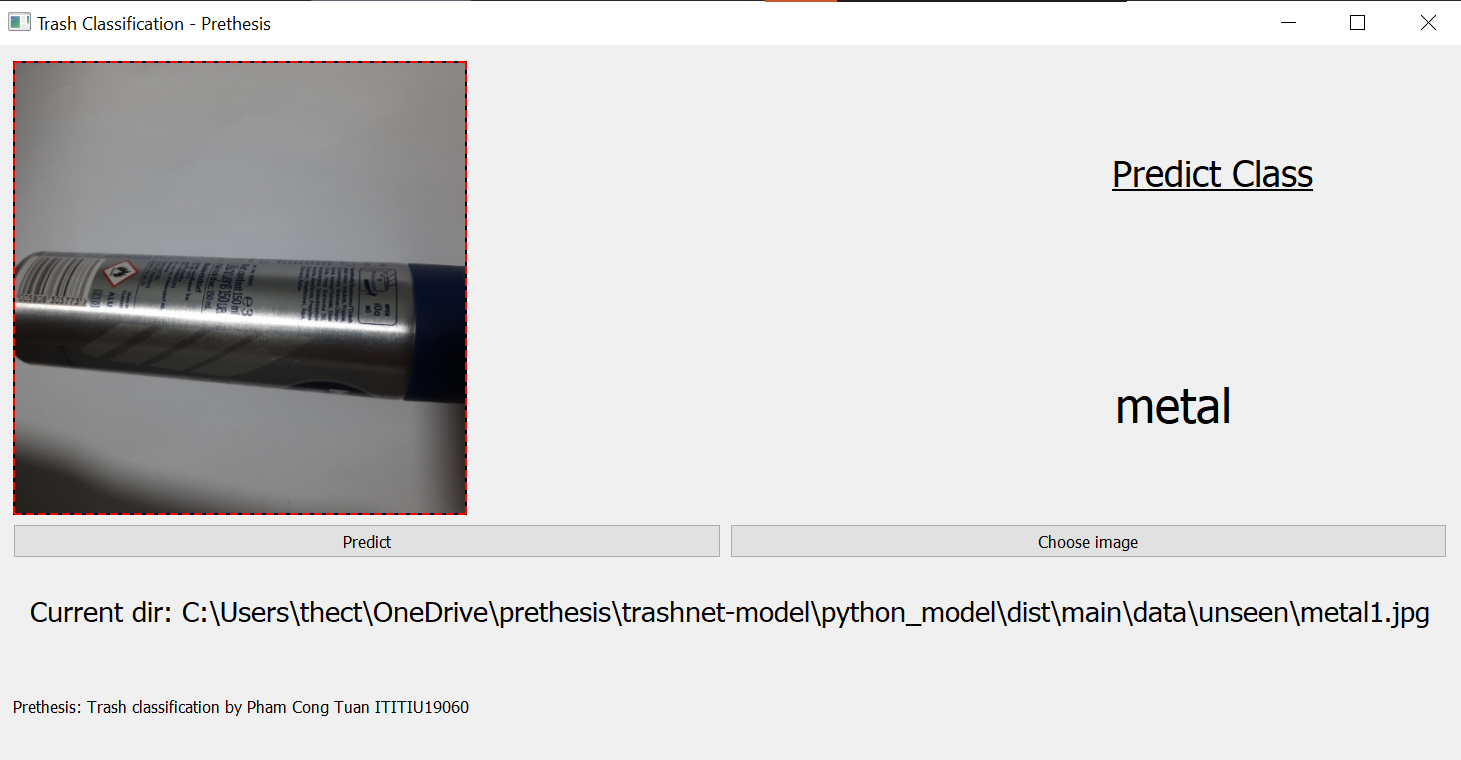


Figure 6. User interface made with PyQt5

### Training Setup

First, a separated script is used to apply image augmentation on the dataset using Imgaug and OpenCV, then combine with the original dataset and upload them to Google Drive. I then run the preparation process on Google Colab, which includes fetching the pretrained MobileNet V2 and applying data loading-resizing-converting to tensors and assigning label. After that, the model would be ready to train with around 5400 samples of the dataset, learning rate of 0.0005 (5e-4), SGD optimizer with momentum = 0.9.

# CHAPTER 4: IMPLEMENT AND RESULTS

## Implementation

To build the model, I use Pytorch with Python, and use imgaug Python library for data augmentation. The program is implemented and trained on Google Colab – a free scientific processing platform for machine learning tasks using its default free plan, due to the fact that training process can take an exceptionally long time, especially without a dedicated GPU. After a lot of testing, thirty-six epochs provide a good balance between performance and time cost. Our classifier is at the end, with a linear function and a dropout layer to minimize overfitting. I use PyTorch to load the data with the train : val : test ratio being 0.7 : 0.2 : 0.1. To make the process efficient, I utilize transfer learning and freeze the first fifteen layers of the pretrained model. In the end, the model is trained for roughly 1 hours and 30 minutes with the batch size being sixteen, momentum being 0.9 and learning rate being 5e-4. After training, I save the new model’s weight to Google Drive for later use.

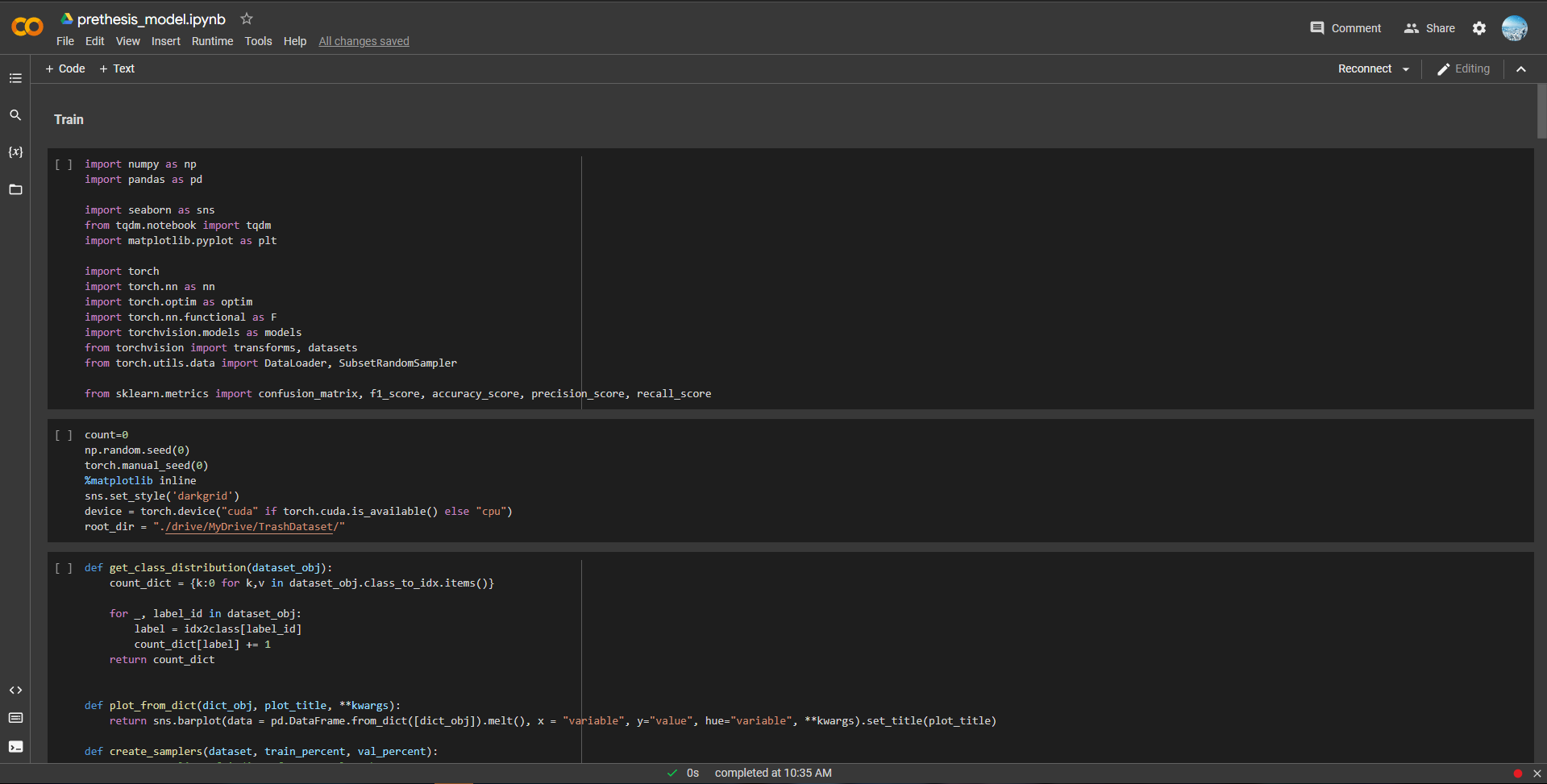


Figure 7. Model training on Google Colab

### Prerequisite

For package management, Anaconda is used to manage separated Python environments and packages. The augmented data is processed to remove outliers, then uploaded to Google Drive. Anaconda distribution of Python can be downloaded from the official website <https://www.anaconda.com/>. I then create an isolated environment and install the libraries inside the mentioned environment for easier management. With Anaconda, managing different environments, especially for scientific tasks is easy and efficient.

Other than Python, I need a few libraries in order for the code to run. For plotting, I use Seaborn and Matplotlib. Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Tqdm is used to illustrate the progress of the training. I also use the metrics from Sklearn, as well as functions from Numpy, Panda and Pytorch.

I use Google Colab for our training platform. In Google Colab, the drive is mounted, and the dataset directory is linked to the code to allow reading the images. No decent hardware is required locally since I train the model on the cloud and save the model & dataset to Google Drive. Colab provides different user plans with multiple perks and prices. For this model, I use the free plan with no upgrade, and 15GB free Google Drive storage by default.

For packing the program, I use PyInstaller to compress all the needed code into a single executable file.

### Challenges

Different materials have different recyclability scores. Even within the same class, they can also vary a lot. For example, there are seven types of plastic alone and only half of them are recyclable. For instance, both nylon and film are implementations of plastic and training the model to correctly recognize them can be challenging. Besides, some classes of trash can have a lot in common, with plastic-glass and glass-metal being the most noticeable examples. Another challenge is that due to the size of the dataset is small, even with data augmentation techniques, it is still limited and cannot fully reflect scenarios from the real life.

## Results

Graphical user interface

Description automatically generated with medium confidence

Figure 8. Loss and Accuracy per epoch

Chart, line chart

Description automatically generated

Figure 9. Loss per epoch graph

As can be seen, the model gets around 78% accuracy on the first try. This can be explained by the fact that I utilize transfer learning with a pretrained model, thus it is able to classify the category of the object, even though with low accuracy. While the original MobileNet V2 can recognize up to one thousand classes of object, the accuracy before optimizing is just acceptable. After extending the model and fine-tuning, I managed to achieve 95%+ test accuracy, with the loss being reduced from 0.9 to as low as 0.03. The loss fluctuates a bit during its decrease, but more epochs would only yield minimal improvement while demands large computational cost.

|  |  |
| --- | --- |
| Trash Classification Model | |
| Accuracy | 96.61% |
| Precision | 96.65% |
| Recall | 0.966 |
| F1 score | 0.966 |

Table 1. Model’s performance on test dataset

Accuracy and precision are popular means to rate a model’s performance. Precision finds out what fraction of predicted positives is positive, while recall criteria is calculated by the ratio of true positives over all positives. It can be understood that recall measures the model’s ability to predict the positives. For F1 score, it is the harmonic mean of precision and recall.

On test dataset, the actual accuracy goes down to 96% showing the model performance being good enough. The drop in accuracy is expected since 100% accuracy is too good to be true and would otherwise suggests that the model is overfitting. Both accuracy and precision are over 96%, with recall criteria and F1 score being ~0.96. In other words, out of one hundred samples, the model would be able to recognize 96 samples correctly thereotically. While the model performs very well given the criteria, the dataset is in fact relatively small, so further testing and optimizations are still possible.

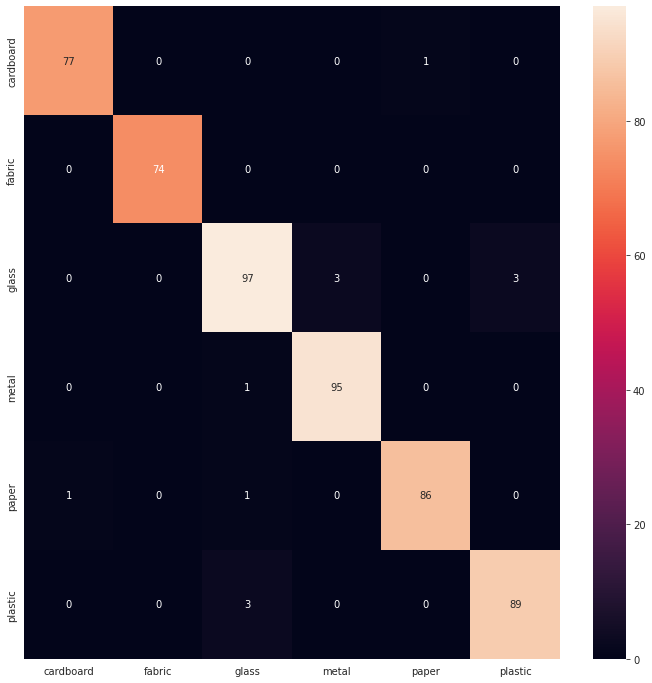


Figure 10. Confusion matrix on test dataset of the model

Figure 4.3 shows the confusion matrix of the model on test dataset. I have six classes of objects on the vertical left, and what they are falsely classified as on the horizontal bottom. Among the categories, glass is most often wrongly recognized as metal and plastic. This can be explained as they share several common characteristics, including the ability to reflect, opacity and smoothness. For other classes, there are occasional misclassifies, though not as frequent as glass. There is also a slight imbalance between the number of test sample from each class.

# CHAPTER 5: DISCUSSION AND EVALUATION

## Discussion

While the model performs well on limited resources, I believe the model can be optimized to gain a slight increase in accuracy. MobileNet V2 was chosen as a base for extension because it is a strong model, which can get to ~80% accuracy on the first try without optimizations. The implementation of freezing layers helps reduce the training time by a large margin while being able to maintain high accuracy. Despite the optimization, there are still some limitations to the current model, such as distinguishing “green” plastic from its non-recyclable counterparts, or periodic misclassification between glass and plastic. The model also requires the sample to be captured separately in order to correctly assign the category.

## Comparison

Among the models mentioned in chapter 2, MobileNet V2 stands out as a promising model. It is able to perform better than other models from section 2.1, with the accuracy competing for the top position. Other than that, the most similar models to ours are GarbNet, proposed by Gaurav Mittal et al and WasteNet by Gary White et al.

Both WasteNet and our proposed model see a few mutual features. The proposal model’s performance is on par with WasteNet model at 96% accuracy on test dataset, and both utilize transfer learning to fine-tune the original pretrained model. Other than that, I implement more data augmentation techniques and have a different dataset split, as well as using different pretrained models while WasteNet proposal introduced a new augmentation method called image expansion and a different set of categories. Regarding the finetuning, the authors of WasteNet decided to unfreeze the layers gradually, while I implement fine-tuning techniques to the first fifteen layers at once. Generally, both models perform competently despite different choices during development and implementation.

Compared to GarbNet, the proposed model is trained on different datasets with a different split ratio. The other main differences include the use of different base models and optimization techniques. GarbNet was able to achieve 87.69% of accuracy, while being noticeably lighter than our model (23MB application compared to ~600MB zip file). This can be explained as they focus more on limiting the computational resource requirements than gaining accuracy with the aim to make the model runnable on IoT devices, as well as the development of a dedicated native Android application, while I use Python libraries to create and package a Windows user interface instead of well-known optimized language like C++.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Implementation of MobileNet V2 | GarbNet | WasteNet |
| Dataset | TrashNet + custom Dataset | GINI Dataset | TrashNet |
| Pretrained model | MobileNet V2 | None | VGG-16 |
| Training time/Epochs | 1.5 hours/20 epochs | - | 1000 epochs |
| Accuracy | 96.61% | 87.7% | 97% |
| Precision | 96.65% | - | 97% |
| Recall | 96.6% | - | 97% |
| F1 score | 96.6% | - | 97% |

Table 2. Comparison between the proposed model, GarbNet and WasteNet

## Evaluation

The model sees practical use for single object classification task with high accuracy, while being flexible for retraining. Even though there are still shortcomings to overcome, the model can help with sorting waste into different categories. The model performs best when the input photo contains an object on a white background, though during my testing, any object with simple backgrounds would work.

# CHAPTER 6: CONCLUSION AND FUTURE WORK

## Conclusion

Excessive waste production is a major problem in many countries, especially with the advance of technology and the improvement of living standards. To tackle the problem, there have been efforts to increase recycling as well as educating young generations about the risk of trash. In this proposal, I have demonstrated an implementation of MobileNet V2 for trash classifying purpose with the hope to help make the recycling process more efficient. I use a modified TrashNet dataset with augmentation techniques from Python library Imgaug, which helps increase the diversity of the dataset. By applying transfer learning and fine-tuning the original pretrained model by freezing the first fifteen layers, the training process becomes more efficient. The model aims to be light-weight and retain the ability to be implemented on edge devices where computation power is often limited. With high accuracy and the flexibility in the model’s architecture, the model can be further adapted and retrained to suit the local needs. Despite the positive results, there are still problems to be considered.

## Future work

For future work, I intend to combine the proposed model with another model for object recognition to make the system more versatile. The model would be extended to support multiple classifications per image, as well as being lighter and more efficient. To be implemented in a waste processing plant, the model would need to be able to approximate how recyclable is each portion of trash, which requires the model to recognize and pickup parts of objects from the image and try to classify the materials to measure the recyclability percentage. This aims to make the recycling process easier and more efficient.

# REFERENCES

[1] United Nations, “High-level Panel on Threats, Challenges and Change”, 2003 Conference.

[2] Deltaway Energy, “Biomass explained: Waste-to-energy (Municipal Solid Waste)” from https://www.eia.gov/energyexplained/biomass/waste-to-energy-in-depth.php

[3] R. A. Aral, Ş. R. Keskin, M. Kaya, M. Haciömeroğlu, "Classification of TrashNet Dataset Based on Deep Learning Models", presented at 2018 IEEE International Conference on Big Data (Big Data), ISBN:978-1-5386-5035-6, DOI 10.1109, Dec 2018.

[4] Gary White, Christian Cabrera, Andrei Palade, Fan Li, Siobhan Clarke, “WasteNet: Waste Classification at the Edge for Smart Bins”, DOI 10.48550, Jun 2020.

[5] Ce Liu, Lavanya Sharan, Edward H. Adelson, Ruth Rosenholtz, “Exploring Features in a Bayesian Framework for Material Recognition”, presented at 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, ISBN:978-1-4244-6985-7, DOI 10.1109, Jun 2010.

[6] G. Mittal, K. B. Yagnik, M. Garg, N. C. Krishnan, "SpotGarbage," presented at the Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, DOI 10.1145 , Sep 2016.

[7] United States Environmental Protection Agency (EPA), “National Overview: Facts and Figures on Materials, Wastes and Recycling” from https://www.epa.gov/facts-and-figures-about-materials-waste-and-recycling/national-overview-facts-and-figures-materials

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks”, DOI 1801.04381, Mar 2019.

[9] Sebastian Mathot, “Customizing MobileNetV2 through transfer learning”, from <https://pythontutorials.eu/deep-learning/transfer-learning/>.

[10] Imgaug, official documentation from https://imgaug.readthedocs.io/en/latest/

[11] Martin Fitzpatrick, “Packaging PyQt5 applications for Windows with PyInstaller & InstallForge” from <https://www.pythonguis.com/tutorials/packaging-pyqt5-pyside2-applications-windows-pyinstaller/>. Assessed August 10, 2022.

[12] Luke Lee, “Using PyInstaller to Easily Distribute Python Applications”, from https://realpython.com/pyinstaller-python/.

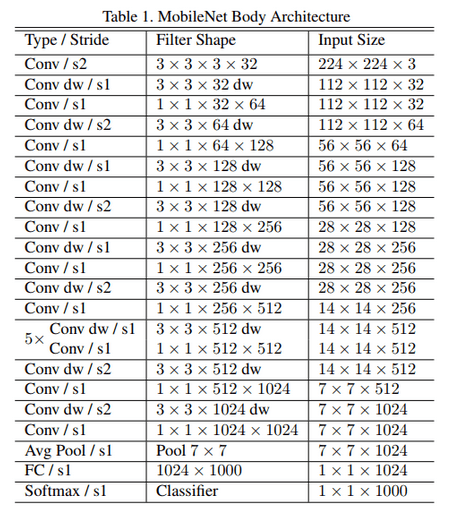
# APPENDIX

* **AP.1**

Source code: <https://github.com/TomPham204/prethesis>

Workspace folder: <https://1drv.ms/u/s!ArPVZgo5x8MIg4ZILypkoooYYs6OUQ?e=dIdl8W>

* **AP.2**



Appendix 2. Overview of MobileNet V2 architecture.

* **AP.3**

(1): MobileNetV2(

    (features): Sequential(

      (0): Conv2dNormActivation(

        (0): Conv2d(3, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        (2): ReLU6(inplace=True)

      )

      (1): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(32, 32, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)

            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2d(32, 16, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (2): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(16, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(96, 96, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=96, bias=False)

            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(96, 24, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (3): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(24, 144, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(144, 144, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=144, bias=False)

            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(144, 24, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (4): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(24, 144, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(144, 144, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=144, bias=False)

            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(144, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (5): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(32, 192, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(192, 192, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=192, bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(192, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (6): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(32, 192, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(192, 192, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=192, bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(192, 32, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (7): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(32, 192, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(192, 192, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=192, bias=False)

            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(192, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (8): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(64, 384, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(384, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=384, bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(384, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (9): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(64, 384, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(384, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=384, bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(384, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (10): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(64, 384, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(384, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=384, bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(384, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (11): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(64, 384, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(384, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=384, bias=False)

            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(384, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (12): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(96, 576, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(576, 576, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(576, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (13): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(96, 576, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(576, 576, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(576, 96, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (14): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(96, 576, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(576, 576, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=576, bias=False)

            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(576, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (15): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(160, 960, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(960, 960, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=960, bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(960, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (16): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(160, 960, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(960, 960, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=960, bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(960, 160, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (17): InvertedResidual(

        (conv): Sequential(

          (0): Conv2dNormActivation(

            (0): Conv2d(160, 960, kernel\_size=(1, 1), stride=(1, 1), bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (1): Conv2dNormActivation(

            (0): Conv2d(960, 960, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=960, bias=False)

            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

            (2): ReLU6(inplace=True)

          )

          (2): Conv2d(960, 320, kernel\_size=(1, 1), stride=(1, 1), bias=False)

          (3): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        )

      )

      (18): Conv2dNormActivation(

        (0): Conv2d(320, 1280, kernel\_size=(1, 1), stride=(1, 1), bias=False)

        (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

        (2): ReLU6(inplace=True)

      )

    )

)

Appendix 2. Detailed architecture of MobileNet V2.