UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Tell me which bin to put this into: Visual classification of recycling items

M.C. Deyzel
20688393

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Dr H. Kamper

November 2020

Acknowledgements

I would like to sincerely thank:

- My supervisor, Herman Kamper, for your interest, time and support.
- My parents, Craig and Tanja, for your love, support and pretence of understanding my degree.
- Annika, for the laughter and invaluable advice.
- The E&E class representatives Damon and Kimberley; and the entire crew, without whom I might just not have made it to my final year.

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

20688393 Studentenommer / *Student number*	MDeyzel Handtekening / *Signature*
M.C. DEYZEL Voorletters en van / *Initials and surname*	**09/11/2020** Datum / *Date*

Abstract

English

The general public finds it difficult to know which items belong to which recycling categories when prompted to self-segregate waste at recycling bins. This report describes the design of an automatic visual waste classification system. A supervised machine learning model is trained to classify waste items into six classes. A convolutional neural network (CNN) is used for its proven accuracy and reliability with image classification. Training is accelerated by use of transfer learning: where the model parameters are initialized by pre-training on a separate and sufficiently large image corpus. Hyperparameters are adjusted so that candidate CNN models achieve their best performance on validation data. The most suitable model is optimized for resource use and speed. The model is transferred to a Raspberry Pi, where a camera module captures a presented waste item and the software classifies it for the user by playing the appropriate audio file. The system performs with an 83% F score on the test data and classification latency is reduced to 2.8 seconds on average.

Afrikaans

Die algemene publiek vind dit moeilik om te weet watter afval tot watter herwinningskategorieë behoort wanneer gevra word om afval by herwinningshouers te skei. Hierdie verslag beskryf die ontwerp van 'n outomatiese visuele klassifikasiestelsel vir afval. 'n Masjienleermodel is onder toesig opgelei om afvalitems in ses klasse te klassifiseer. 'n Konvolusionele neurale netwerk (CNN) word gebruik vir die bewese akkuraatheid en betroubaarheid met beeldklassifikasie en opleiding word versnel deur die gebruik van oordragleer (*transfer learning*). Hiperparameters word so aangepas dat kandidaat CNN-modelle hul beste prestasie op valideringdata behaal. Die mees geskikte model is geskies volgense sy las op rekenaarhulpbronne en spoed. Die model word op 'n Raspberry Pi gelaai, waar 'n kameramodule 'n afvalitem afneem en die sagteware dit klassifiseer vir die gebruiker deur om die toepaslike klanklêer te speel. Die stelsel verrig 'n 83% F-telling op die toetsdata en vertraagdheid word tot gemiddeld 2.9 sekondes verminder.

Contents

Declaration ii

Abstract iii

List of Figures vi

List of Tables viii

Nomenclature x

1. Introduction 1

1.1. Objectives and Constraints . . . 2
1.2. Methodology . . . 2
1.3. Contributions . . . 3

2. Background 4

2.1. Deep Learning . . . 4
2.1.1. Convolutional Neural Networks . . . 4
2.1.2. Gradient Descent and Backpropagation . . . 6
2.1.3. Transfer Learning . . . 8
2.2. Related Work . . . 9

3. Model Design 10

3.1. Training Pipeline . . . 10
3.1.1. Data Preparation . . . 10
3.1.2. Mini-Batch Training . . . 11
3.1.3. Validation and Tuning . . . 13
3.2. Proposed Models . . . 13
3.2.1. VGG-16 . . . 13
3.2.2. ResNet-50 . . . 14
3.2.3. MobileNet v2 . . . 14
3.2.4. InceptionNet v3 . . . 14
3.2.5. DenseNet-121 . . . 15
3.2.6. EfficientNet-B4 . . . 15
3.3. Hyperparameter Tuning . . . 16

3.3.1. Architecture . . . 16
3.3.2. Learning . . . 17
3.4. Chapter Summary . . . 22

4. Model Testing **24**
4.1. Testing Pipeline . . . 24
4.1.1. Metrics . . . 24
4.1.2. Practical Constraints . . . 25
4.1.3. Comprehension Techniques . . . 25
4.2. Test Results . . . 26
4.2.1. Classification Performance . . . 26
4.2.2. Cost and Latency . . . 27
4.2.3. Final Model Selection . . . 28
4.3. Model Analyses . . . 28
4.3.1. Misclassifications and Confusion . . . 28
4.3.2. Feature Attribution . . . 29
4.4. Improvements . . . 31
4.4.1. Decision Threshold . . . 31
4.4.2. Quantization . . . 31
4.5. Chapter Summary . . . 34

5. System Implementation **35**
5.1. System Overview . . . 35
5.2. User Interface . . . 35
5.2.1. Audio Feedback . . . 36
5.3. Final System Tests . . . 36
5.4. Final Optimizations . . . 37
5.5. Chapter Summary . . . 37

6. Conclusion **39**
6.1. Summary . . . 39
6.2. Main Findings . . . 40
6.3. Future Work . . . 40

Bibliography **41**

A. Dataset Split **44**

B. Project Planning Schedule **45**

C. Outcomes Compliance **46**

List of Figures

1.1. Colour-coded bins on campus at Stellenbosch University, South Africa motivate the public to self-segregate litter. 2

2.1. A conceptual CNN with two feature extractor layers classifying an image of a waste item. 5
2.2. A convolution operation with a horizontal edge detecting filter. 5
2.3. An illustration of transfer learning common to image recognition problems. 8

3.1. An illustration of data preparation 12
3.2. ResNet blocks showing skip connections with identity mappings 14
3.3. Illustrations of inception and dense blocks 15
3.4. The common classifier block appended to the feature extractor layers of each model 16
3.5. Training curves for some models. 19
3.6. A diagram of the Bayesian optimization algorithm 20
3.7. The surrogate model as a GP and the associated EI for a univariate parameter space. 21
3.8. Models showing increasing validation accuracies as the BO search discovers better learning hyperparameter sets each trial 22
3.9. Mean accuracy and associated standard errors for Bayesian optimization results 23

4.1. A plot comparing the candidate models' performances against inference time. 27
4.2. Misclassified test samples by ResNet-50 28
4.3. Confusion matrices of three models. 29
4.4. Attribution visualizations for three test images. 30
4.5. Search results for best F score on the test set given different decision thresholds. 32
4.6. Confusion matrices comparison for decision threshold implementation. . . . 32
4.7. An illustration of quantization-aware training. 33

5.1. Illustrations of the design of the final classification system. 35
5.2. Images of the final system implementation. 36
5.3. Waste samples classified by the final implemented system. 37

A.1. The dataset split into training, validation and test sets per class. 44

B.1. An estimated timeline for the project tasks. 45

List of Tables

3.1. Grid search results for the architecture hyperparameters 18
3.2. Results from Bayesian optimization after 20 trials for each model. 21

4.1. Classification results on test set for trained candidate models. 26
4.2. Resource burden and inference time for each candidate model. 27
4.3. Comparison of unquantized final model with quantization-aware training. . 33

5.1. Results from latency improvement methods. 37

C.1. ECSA outcomes compliance. 46
C.2. ECSA outcomes compliance (cont.). 47

Nomenclature

Variables and functions

$\mathbf{W}_f^{[l]}$	Kernel weight matrix for filter f in layer l
$\mathbf{z}_{f,p}^{[l]}$	Filter operation result for filter f of patch p in layers l
$\mathbf{a}_p^{[l]}$	Activation result for patch p in layer l
θ	Parameter of a network
K	Number of classes
$\hat{y}^{(i)}$	Softmax probability of a class i for a given example
$y^{(i)}$	One-hot encoded ground truth label of a class i for a given example
$o^{(i)}$	Output logit of class i for a given example
g_t	Local gradient of a given weight
L_{CE}	Cross-entropy loss
α	The learning rate
β_1	The decay rate of the moving average (Nosterov momentum)
N_{B}	The size of a mini-batch
t	A single mini-batch in a total of T mini-batches
e	A single epoch in a total of E epochs
G_{sum}	Sum of gradients for a mini-batch, used to calculate the average G_{avr}
f_{M}	Classifier model function
E_{s}	Learning rate scheduler step size in epochs
A_{V}	Validation accuracy
H	Number of hidden layers in classifier
N	Number of neurons in hidden layer (size of hidden layer)
$\mathbf{X}$	A parameterization set for model hyperparameters
$\bar{P}$	Arithmetic mean of precision values P_k across K classes
$\bar{R}$	Arithmetic mean of recall values R_k across K classes
F	F score: harmonic mean of average precision and recall across classes
W_{L}	Length of square occlusion window
W_{s}	Stride of occlusion window
λ	Decision threshold for class prediction

Acronyms and abbreviations

IoT	Internet-of-things
CV	Computer vision
CPU	Central processing unit
CNN	Convolutional neural network
SGD	Stochastic gradient descent
ANN	Artificial neural network
MLP	Multi-layer perceptron
ReLU	Rectified linear unit
FC	Fully-connected
FLOPS	Floating point operations per second
FLOP(s)	Floating point operation(s)
BO	Bayesian optimization
GP	Gaussian process
EI	Expected improvement
RAM	Random-access memory
SHAP	Shapley additive explanations
FP32	32-bit floating-point representation
INT8	8-bit integer representation
MAC	Multiply-and-accumulate
QAT	Quantization-aware training
GPIO	General-purpose input/output
DC	Direct current
CSI	Camera serial interface
TTS	Text-to-speech

Chapter 1

Introduction

In our efforts to advance toward a closed-loop economy, recycling is and will become a requisite global practice for sustainable human growth and development. Waste segregation or waste sorting is the practice of separating solid waste items into different categories during disposal in order to facilitate recycling efforts at waste management facilities. Separating waste items before recycling is important to avoid contamination of waste, which diminishes the recycling rate at these facilities [1].

The concept of source segregation allows the public to self-sort their disposable items into categorized bins. The categories for waste recycling vary greatly from one institution or municipality to the next. Often, there is only a discrimination between 'dry' and 'wet' waste, or 'recyclable' and 'non-recyclable'. However, the public often finds it difficult to know which items are considered recyclable and which non-recyclable [2, 3]. This has led to the use of composition material categories. Common material categories include: plastic, glass, paper, metal, compost and non-recyclable. Bins are usually labelled with these categories – or combinations of them – and are colour-coded. Large institutions like universities often have their own waste management systems which service these recycling bins throughout campuses. At Stellenbosch University, South Africa, such campus-wide bins are available for source segregation as part of a waste management initiative. An example is shown in Figure 1.1.

Source segregation methods are effective. Chen et al. [3] have shown that Chinese municipalities that provide segregated bins to the public were able to effectively deal with and treat solid waste in towns and cities. However, the effectiveness of such a system is very dependent on the public's own knowledge of the materials that comprise their refuse and their own willingness and care to properly segregate.

With modern advancements in machine learning and a growing global appetite for Internet-of-Things (IoT)-enabled devices, the implementation of a smart bin that automatically identifies and distributes presented waste items by recycling category seems natural and inevitable. Computer vision (CV) technology has ushered in a new era of field service capabilities, and in the area of image classification specifically, computers and other devices are able to quickly and accurately classify an image or different objects in an image.

Figure 1.1: Colour-coded bins on campus at Stellenbosch University, South Africa motivate the public to self-segregate litter.

1.1. Objectives and Constraints

The goal of this project is to both design and implement an embedded image classifier system for automatic waste categorization by making use of machine learning techniques. The classifier model to be implemented is the design that has both a sufficiently high accuracy and an acceptable latency. More complex models often provide a higher accuracy, but do so at the expense of training speed, more memory use, higher computational load, and longer inference time (time to make a classification). In the context of this project, training time is not a consideration, since training is performed on a separate machine and only the model state containing the trained parameters is loaded onto the edge device. With this organization, memory size and CPU speed is a constraint, while accuracy, macro-averaged F score and inference time is an objective.

Considering these metrics, the goal is to implement the model with the best performance on a Raspberry Pi 4, a single-board computer the size of a bank card. This device's small size and low power consumption make it a good candidate for an edge device, while still providing enough real memory and CPU performance to be able to run inference software in-the-field. The Pi is programmed to take a photo of the presented waste item using a camera module, predict the category of the item and then play the corresponding audio file indicating the category predicted.

1.2. Methodology

This project seeks to determine the most suitable image classification model for a dataset constructed by Thung and Yang [4], which contains 650 images of waste items labelled as either one of six classes: cardboard, paper, plastic, glass, metal or trash. The dataset is split into three separate subsets for training, validation and testing.

The programming environment is PyTorch, an open-source library in Python for machine learning which provides automatic computational graph construction, automatic

differentiation, and optimization for computer vision and natural language processing applications.

Six different CNN models are candidates: VGG-16, MobileNet v2, ResNet-50, InceptionNet v3, DenseNet-121, and EfficientNet-B4. These models are pretrained on the ImageNet database [5] and are trained on the labelled, augmented training images by means of transfer learning and fine-tuning, which is highly effective even for image target sets very distant from ImageNet [6]. Images are augmented by random flips and rotations to further increase dataset size and diversity.

To each CNN is appended the same fully-connected classifier layers with adjustable hidden layer numbers and sizes. The final layer is a softmax layer which can assign a probability to each class. The model's error for a given training iteration as per the loss function is backpropagated into the network by the Adam optimizer [7], an extension of the stochastic gradient descent optimizer (SGD) algorithm. Validation images are used to tune hyperparameters and discover the learning framework and layer structure that result in the highest validation accuracy. Once each model is fine-tuned and trained, they are tested on the test image set to determine the class precision and recall, the macro-averaged F score, and the average peak memory use for inference.

Experiments to reduce memory use are conducted on select candidate models to attempt to achieve similar results with more efficient resource use. The best performing model is then loaded onto a Raspberry Pi 4 for inference testing to further determine peak memory use and inference time on the device. Based on these criteria, the most appropriate model is selected for implementation. The classifier model is run on the device which further employs a decision threshold to ensure that trash items are not placed in recycling categories, but provide leniency for vice versa. Finally, the device is programmed to play text-to-speech (TTS) audio to communicate to the user the class predicted and its own confidence in the prediction as per the softmax output, which provides a fun literal interpretation of the project title.

1.3. Contributions

This project is unique in its practical implementation of a machine learning litter classifier on a Raspberry Pi, which has not been observed before in the literature. It contributes the comparison of the novel EfficientNet architecture with other mobile-optimized CNN models. It is unique in its use of decision thresholds for classification, which improves the F score in such a problem setting. Lastly, it makes use of novel and unrefined optimization techniques such as quantization and neural network comprehension techniques such as feature attributions.

Chapter 2

Background

Accurate image classification is achieved overwhelmingly by deep learning methods. Since the performance of AlexNet [8] on the ImageNet Large Scale Visual Recognition Challenge in 2012, the accuracy of deep learning algorithms on benchmark computer vision datasets for computer vision has surpassed prior methods and maintained state-of-the-art status. This chapter provides some background on deep learning in computer vision and explores previous approaches to visual waste classification using these methods.

2.1. Deep Learning

Deep learning refers to the use of machine learning with artificial neural networks, which is the case in this project. Supervised learning is used to teach a convolutional neural network to classify images into classes by training on labelled image datasets.

2.1.1. Convolutional Neural Networks

Artificial neural networks (ANNs) are accurate function approximators that learn non-linear relationships between input data and target outputs [9]. One approach for building image classifiers is to feed a flattened array of raw pixel values into an ANN which consists of only neurons with fully-connected inputs and output activations. The disadvantage of this approach is that fully-connected network architectures are prone to overfitting given the number of parameters and are inefficient, because they do not take local spatial coherence of features into account.

Convolutional neural networks (CNNs) are regularized ANNs with connection structures inspired by the operation of mammalian visual cortex. They learn the same complex relationships with fewer parameters [10]. In lieu of passing raw flattened pixels into classifiers, they pre-process the image using layers that consist of convolutional and pooling operations, as illustrated in Figure 2.1.

A layer l applies N_F filters to an input image (or a preceding layer activation map) using convolutions, which implies that there are N_F activation maps for each input map from the previous layer. One filter kernel $\mathbf{W}_f^{[l]}$ in a certain layer l is a matrix of learnable

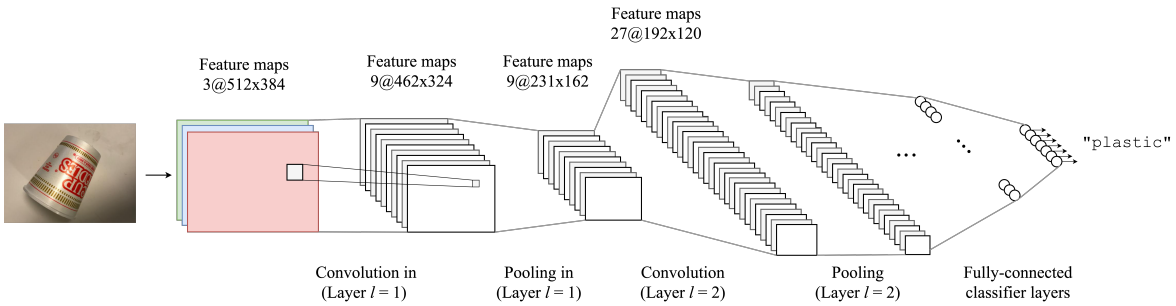


Figure 2.1: A conceptual CNN with two feature extractor layers classifying an image of a waste item. The first feature maps are the three image colour channels.

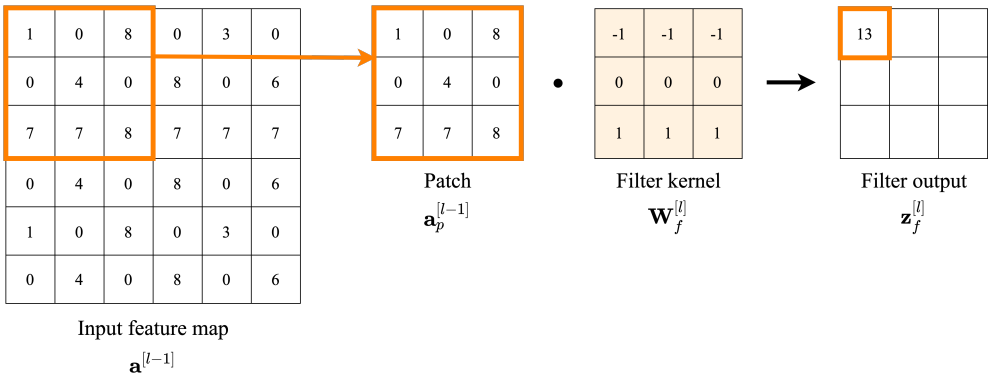


Figure 2.2: A convolution operation with a horizontal edge detecting filter. There is a high activation for the patch, indicating that a horizontal edge was detected.

weights that is convolved over the input feature map:

$$\mathbf{z}_f^{[l]} = \mathbf{a}^{[l-1]} * \mathbf{W}_f^{[l]} \tag{2.1}$$

The filter kernel slides across the two-dimensional input space in strides and performs element-wise matrix multiplication and summation with an image patch p:

$$z_{f,p}^{[l]} = \mathbf{a}_p^{[l-1]} \cdot \mathbf{W}_f^{[l]} \tag{2.2}$$

where $z_{f,p}^{[l]}$ is the scalar filter operation output for patch p for a filter f in a layer l and $\mathbf{a}_p^{[l-1]}$ is the input feature map in position p from the previous layer $l-1$. The operation performed is the dot product defined as being a sum of element-wise products and is illustrated in Figure 2.2. To be clear, a single element at position (x, y) in the output matrix in given by:

$$z_f^{[l]}(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{a}^{[l-1]}(x+i, y+j) \mathbf{W}_f^{[l]}(i, j) \tag{2.3}$$

where the weight kernel has size n by m.

The filters act as very specific feature extractors by yielding high activation outputs

for patches that correlate with it [11] and the operation is more accurately termed the cross-correlation. To be more accurate, the outputs are generally activated by an activation function g:

$$\mathbf{a}^{[l]} = \mathrm{g}(\mathbf{z}^{[l]}) \tag{2.4}$$

which in most cases and for this project is the rectified linear unit (ReLU) function given by:

$$\mathrm{g}(\mathbf{z}) = \max(0, \mathbf{z}) \tag{2.5}$$

The use of the ReLU function for activation provides non-linearity for learning more complex patterns in the data [10]. Importantly, the ReLU function is piece-wise differentiable, which makes it suitable for gradient backpropagation.

Shallower layers generally have smaller kernel sizes (3×3 pixels) and are used for low-level edge detection. As layer deepen, the filters become larger (7×7) to recognize high-level patterns. CNNs learn numerous layers of feature representations by applying different filters with different sizes so that it may preserve the spatial and temporal pixel dependencies present in images [10, 11].

A pooling operation in a layer simply down-samples the activations into fewer values to summarize features representations. This is done by grouping activation values in windows and summarizing the region by taking its maximum entry as in the case of max pooling, or finding the mean in the case of average pooling. In all the networks used in this project, the feature extractor layers end with an adaptive average pooling operation which flattens the activations into a linear array. This operation adapts the pooling window size and stride so that an image of any size can be fed to the network. Finally, fully-connected (FC) layers compose the classifier block where all the inputs from one layer are connected to every activation unit or 'neuron' of the next layer.

2.1.2. Gradient Descent and Backpropagation

Networks should be able to learn the parameters which define filter kernels able to recognize applicable image features. Gradient descent is an iterative optimization algorithm to minimize a loss function $\mathrm{L}(\theta)$ which is a function of the model's parameters $\theta \in \mathbb{R}^{\mathrm{d}}$. It updates the network parameters in the direction opposite to the gradient of the loss function with respect to the parameters $\nabla_\theta \mathrm{L}(\theta)$. The loss function used for image classification and in this project is the cross-entropy loss L_{CE}, which is given by:

$$L_{\mathrm{CE}}(y, \hat{y}) = -\sum_{i=1}^{K} y^{(i)} \log \hat{y}^{(i)} = -\sum_{i=1}^{K} y^{(i)} \log \mathrm{S}(o^{(i)}) \tag{2.6}$$

where y is the ground truth class and $\hat{y}$ is the probability given by S, the softmax function:

$$\mathrm{S}(o^{(i)}) = \frac{e^{o^{(i)}}}{\sum_{j=1}^{K} e^{o^{(j)}}} \tag{2.7}$$

and $o^{(i)}$ is output logit i of the network by the final FC layer. The derivative of Equation 2.6 with respect to the positive class is:

$$\frac{\partial}{\partial o_p}(-\log(\frac{e^{o_p}}{\sum_j^K e^{o_j}})) = \frac{e^{o_p}}{\sum_j^K e^{o_j}} - 1 \tag{2.8}$$

and the negative class is:

$$\frac{\partial}{\partial o_n}(-\log(\frac{e^{o_p}}{\sum_j^K e^{o_j}})) = \frac{e^{o_n}}{\sum_j^K e^{o_j}} \tag{2.9}$$

where o_n is the score of any negative class.

These gradients are backpropagated into the network to calculate the local gradients for the parameters at each function. Backpropagation uses computational graphs which link sequential function instances (such as addition, multiplication and squaring) during the forward pass phase [12]. This graph is used to calculate the gradient g_t at an instance t[1] of the loss function L_{CE} with respect to any learnable parameter θ in the network:

$$g_t = \frac{\partial L_{\mathrm{CE}}}{\partial \theta} \tag{2.10}$$

which the Adam optimizer uses to adjust the weights and biases with a learning rate of α. The Adam optimizer [7] calculates an exponential moving average of the gradient:

$$v_t = \beta_1 v_{t-1} - (1 - \beta_1) g_t \tag{2.11}$$

and the square of the gradient:

$$s_t = \beta_2 s_{t-1} - (1 - \beta_2) g_t^2 \tag{2.12}$$

The decay rates of the these moving averages (the momenta) are β_1 for the gradient and β_2 for the squared gradient. After correcting for the bias produced by the zero-initialization of the gradient average:

$$\hat{v_t} = \frac{v_t}{(1 - \beta_1^t)} \tag{2.13}$$

$$\hat{s_t} = \frac{s_t}{(1 - \beta_2^t)} \tag{2.14}$$

[1]Instance t specifically refers to a mini-batch in the case of this project. Mini-batch SGD is used and is further described in Section 3.3.2.

The update performed on a parameter θ is then:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{v}_t}{\sqrt{\hat{s}_t}} g_t \tag{2.15}$$

The learning rate α and β_1 values are taken as hyperparameters to be optimized, as described in Section 3.3.2.

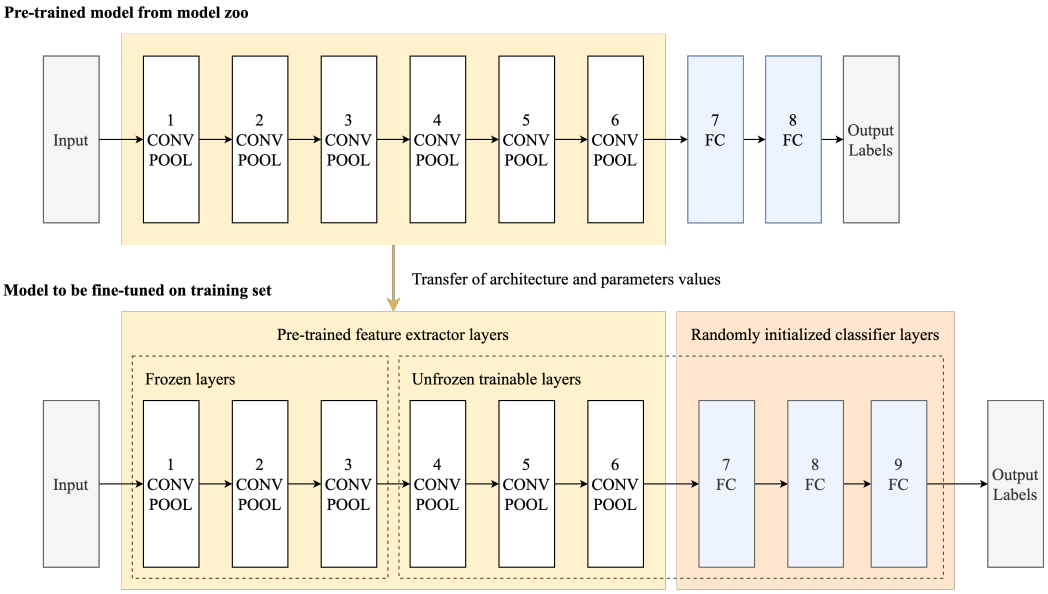


Figure 2.3: An illustration of transfer learning common to image recognition problems.

2.1.3. Transfer Learning

Parameters are not always trained from scratch. Given the computational load, time and resources required to develop and train CNN models for accurate image recognition, transfer learning and fine-tuning is used for this project. Transfer learning in the context of deep learning is a method whereby model architectures are used with pre-trained parameters which are taken from a model zoo.

In the context of image classification with CNNs, networks are generally pre-trained for the regular ImageNet 1000-class photograph classification competition. This approach is effective because the models are trained on a large corpus of images from a large number of classes which requires that the model learn to extract general features from images efficiently. CNN filter parameters extract more generic features in early shallower layers (such as edges and textures) and features more specific to the dataset in deeper layers (such as airplane wings and noses) [11]. Transfer learning is especially import for this problem given the small dataset, which will cause the model to overfit if random parameter initializations are used [13]. Additionally, research has shown that ImageNet classification is biased toward material features [14], which is beneficial in this problem setting.

To have models specialize on the dataset, the final FC layers that comprise the classifiers are replaced with randomly-initialized FC layers, as described in Section 3.3. Unfreezing refers to enabling some 'unfrozen' feature extractor layers to be iterated through by the optimizer to adjust their filter parameters to fit to the dataset trained on. The process of transfer learning is illustrated in Figure 2.3.

2.2. Related Work

Several researchers have attempted waste classification using machine learning methods, including the above. Thung and Yang use a Support Vector Machine (SVM) classifier to obtain an accuracy of 63% with their manually collected waste dataset named 'TrashNet'. Previous work on recycling chiefly rely on this dataset. A subsequent CNN-based approach by Bircanoglu et al. [15] on the same dataset uses transfer learning and fine-tuning to achieves a test accuracy of 95% on test data by the DenseNet-121 CNN model. Work by White et al. [16] uses CNN models with transfer learning and focus on inference at the edge. They obtain a prediction accuracy of 97% on the test data.

No previous work has empirically measured or considered the time and resource use profile of a waste classifier. This project uses the methods of transfer learning and fine-tuning that has proven to yield state-of-the-art waste image classification results and additionally inspects the efficacy of implementation of such a system by designing for practical constraints and physical implementation.

Chapter 3

Model Design

This chapter describes the design procedure of each of the candidate image classifier models. Specifically, it describes how the training data is prepared, the structure of the models that will undergo training on the dataset, how transfer learning and fine-tuning is applied, and how the optimal hyperparameters are determined. The model design step is only concerned with discovering the set of hyperparameters that result in the highest validation accuracy for each model. Many methods have been proposed in the literature to tune hyperparameters. The methods of grid search and Bayesian optimization is used here. To be clear, hyperparameters for the models are selected based on performance only on the validation dataset, which models are never trained on.

3.1. Training Pipeline

3.1.1. Data Preparation

The TrashNet dataset [4] used for training the classifiers in this project consists of a total of 2527 labelled images of waste items. The classes are, along with the total example images in each: cardboard (403), glass (501), metal (410), paper (594), plastic (482) and trash (137). All the images in the dataset are 512 pixels by 384 pixels in RGB colour space. The photos have good lighting which exposes the reflective properties of the material. Material visibility is important, since waste item can be distorted into any shape, so the classifier must learn patterns in the physics of the material properties. A disadvantage of the dataset is that all photos are taken on the same white platform, which will mean that the system implementation also requires a white background of similar material.

These images are further divided into three separate datasets: 80% of each class is reserved for training, 10% is reserved for validation and the remaining 10% for testing. It is important to note that this split is not done randomly: The final two tenths of labelled images from each class folder are withheld for the validation and test sets. This has the advantage that two different photographs of the same waste item will not appear in both the training and the test sets, because there are occasionally multiple different photographs of the same waste item present in the TrashNet dataset.

The purpose of the validation set is to have a collection of image data that the model

is never trained on in order to determine hyperparameters for the models as discussed in this chapter. A separate validation set also provides a validation loss which is used in conjunction with the training loss to ensure that the networks are not overfitting. The purpose of the test set is to compare the designed models and assess their suitability for implementation on the device, which is done in Section 5. An illustration of the dataset split per class is shown in Appendix A.

Some transformations are applied to the images that assist in training speed and model performance. Firstly, data augmentation is performed on the training set, which increases the dataset size and diversity [13]. Every image is randomly flipped horizontally and/or vertically with probabilities of 0.5 and then rotated by a random value between -20 and 20 degrees. Instead of expanding the dataset in storage, these random transformations are applied to each image in-memory as they are fed into the networks for training. By seeing different orientations of the same bottle or can, the models are encouraged to learn low-level material attributes like colours and reflective behaviours instead of shapes or even whole example images.

Normalization is also applied to all the image data, which scales the pixels values in each channel using the transformation:

$$p_N = \frac{p - \mu_C}{255\sigma_C} \tag{3.1}$$

where p_N is the new pixel value, p the old and μ_C and σ_C are the mean pixel value and the standard deviation in pixel value respectively of colour channel C. This benefits the neural networks' training by keeping initial input values in a similar range, causing their subsequent activations to be in a limited range so that during backpropagation, the gradients remain in control and are not swerved to extremes by large values and variances in the dimensions [17]. The data is centred around zero for the ReLU activation functions, so learned weights and biases do not have to be large in magnitude. This identical variance in each dimension also makes gradient descent easier for the optimizer by ensuring similar gradient landscapes in every dimension. Images from the dataset and their augmentations are shown in Figure 3.1.

3.1.2. Mini-Batch Training

Updating a network with the frequency of every image can result in a noisy gradient signal and cause parameters, and therefore the training loss, to show high variance over epochs, which may make it difficult for the model to descend to an error minimum [18]. An alternative to this approach is to only update the model after all example images in the training set have been fed and evaluated and all the errors have been accumulated, which is known in the literature as batch gradient descent. This runs the risk of the model prematurely settling on a sub-optimal set of parameters. The approach used for

(a) cardboard, paper, glass, plastic, metal, trash

(b) Augmentations of the same image of a bottle from the 'glass' class.

Figure 3.1: An illustration of data preparation. (a) shows sample images from the different classes and (b) demonstrates the effect of image augmentation.

training in this project is to feed a limited mini-batch of images to the model at a time to calculate and accumulate the errors before updating. Therefore, the mini-batch size determines the frequency of updating the model. The training dataset is divided into T equally-sized mini-batches of N_B samples each, where N_B is treated as a hyperparameter to be optimized. The mini-batch training algorithm is described in Algorithm 3.1.

Algorithm 3.1: Mini-Batch Training

for $e \leftarrow 1$ to E **do**
  for $t \leftarrow 1$ to T **do**
    Select N_B random samples from training set
    $G_\mathrm{sum} \leftarrow 0$
    for image i in N_B **do** ▷ For image in a batch
      $\hat{y} \leftarrow f_\mathrm{M}(i)$ ▷ Predict class from model
      $L \leftarrow L_\mathrm{CE}(y, \hat{y})$ ▷ Calculate loss
      $G_\mathrm{sum} \leftarrow G_\mathrm{sum} + \nabla L$
    end for
    $G_\mathrm{avr} \leftarrow G_\mathrm{sum}/N_\mathrm{B}$ ▷ Average gradients for batch
    Zero gradients in computational graph
    Backpropagate using G_avr
    Update weights using Adam
  end for
  if $e \mod E_\mathrm{s} = 0$ **then**
    $\alpha \leftarrow 0.1\alpha$ ▷ Step the learning rate
  end if
end for

For gradient descent using the Adam optimizer, the learning rate α as in Equation 2.15 and β_1 as in Equation 2.11 are taken as hyperparameters to be optimized.

3.1.3. Validation and Tuning

Hyperparameters are tuned using two methods and the objective performance metric of the model is validation accuracy (A_{V}). Validation accuracy is defined as the total number of correct class predictions as a fraction of the total number of class predictions performed:

$$A_{\mathrm{V}} = \frac{\#\text{correct validation predictions}}{\text{size of validation set}} \tag{3.2}$$

and validation is performed in each epoch after the training stage. In the validation stage, the function gradients are detached from the functions in the model so that they are not calculated or adjusted, which prevents training on the validation set. The A_{V} score is used for its ease of implementation in the programming environment and because it serves as a sufficient proxy for the ability of the models to fit to the data.

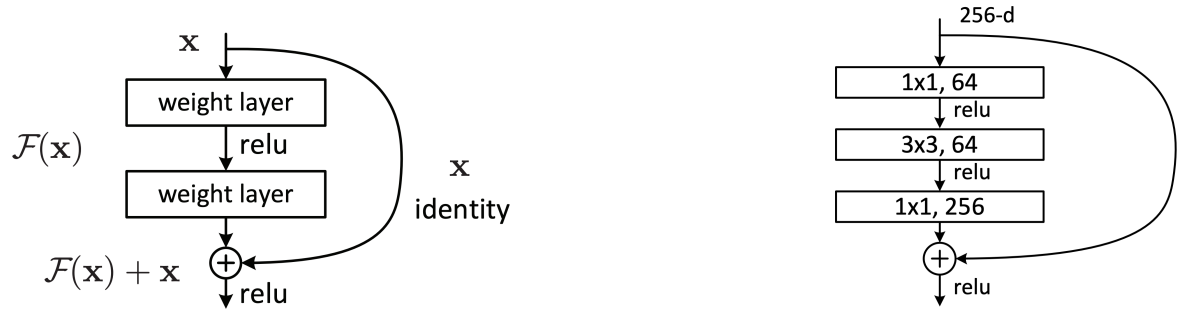
The hyperparameters that undergo tuning are divided into three categories: architecture, learning, and regularization. Architecture hyperparameters relate to the structures of the common classifier blocks attached to each model. Learning hyperparameters encompass those that specify the learning framework during training, such as speed and scope of learning. Not considered for design are hyperparameters involved in regularization, as convolutional neural network models observe self-regularization for sufficiently large datasets by virtue of their lower information capacity [11]. A dropout of 0.2 on the first FC layer of the classifier block is the only empirical regularization method implemented throughout the model design phase.

3.2. Proposed Models

Proposed models refer to candidate CNN models for the task. They have been designed by researchers in the literature and have proven to be effective at image classification tasks. This section explores the theory of the designs and their potential to perform on the problem.

3.2.1. VGG-16

VGG-16 is a deep CNN model proposed by University of Oxford academics in [19]. The major contribution from this model was the use of sequential 3×3 filters in lieu of large filter sizes, which promotes an effective receptive field. While the network is simple to understand and implement given the homogenous topology, the notable disadvantages of this network are the time taken to train given the computationally expensive FC layers and the large storage size of 533MB.



(a) A residual block. **(b)** A ResNet-50 bottleneck.

Figure 3.2: ResNet blocks showing skip connections with identity mappings as illustrated in the original paper [20].

3.2.2. ResNet-50

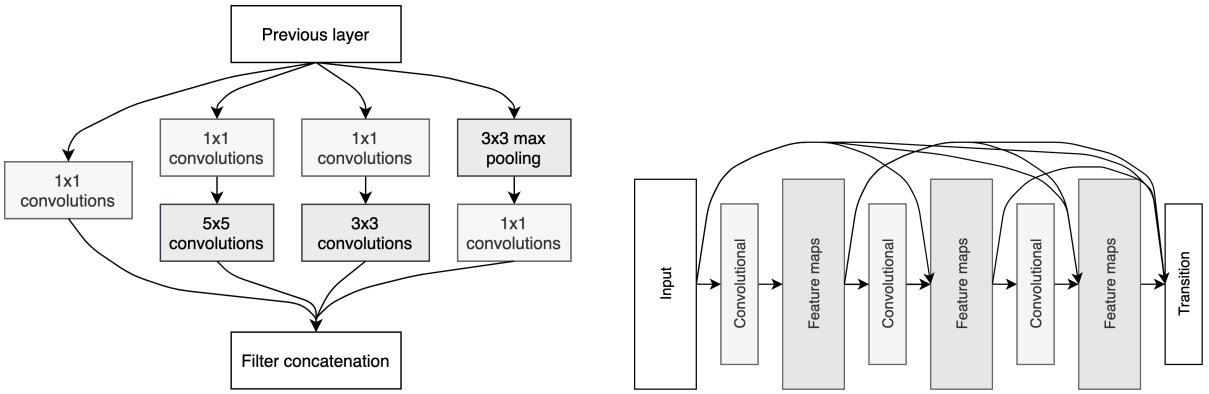
The ResNet architecture [20] was proposed in response to the observation that adding more layers to backpropagate through caused gradients to become vanishingly small. The concept of residual learning implemented shortcut or "skip" connections between layers, which allows an alternative route for gradients to flow through and ensures gradients never vanish for even arbitrarily small weights [21]. The model can then learn an identity mapping if needed, which will ensure that deeper layers perform at least as well as preceding shallower layers. A residual learning block is shown in Figure 3.2a and the bottleneck residual block used in ResNet-50 is shown in Figure 3.2b.

3.2.3. MobileNet v2

MobileNet was proposed in [22] as a lightweight CNN architecture for implementation in limited-resource embedded systems. It uses the concept of depth-wise separable convolutions which applies a single convolutional filter operation per single input channel, instead of to all channels simultaneously. This significantly reduces the computational work and the number of weights to learn. MobileNet approaches are therefore good candidates for the implementation environment.

3.2.4. InceptionNet v3

InceptionNet v3 was introduced in [23]. The main contribution of this architecture was the introduction of the inception module, shown in Figure 3.3a Instead of simply increasing the filter sizes as the layers go deeper, the inception module applies parallel convolutions with four filters of differing sizes to the input block of feature maps in the same layer before concatenating the results. This allows the model to learn which filter sizes to use when making an inference for a given dataset and for a given image. 1×1 convolutions are used to reduce the number of feature maps which minimizes computational requirements before



(a) An inception block showing parallel convolutions.

(b) A dense block showing all activations being fed forward.

Figure 3.3: Inception blocks are wider due to multiple convolution operations and dense blocks are deeper because adding more layer are feasible.

the subsequent expensive convolutions. An inception-inspired architecture model might be a good candidate for the waste classification problem given its ability to concentrate on learning complex patterns in low-level features.

3.2.5. DenseNet-121

The concept of DenseNet (Dense Convolutional Network) [24] is to take advantage of the increased performance of deepening networks. To solve the vanishing gradient problem that accompanies deeper networks, the design is partly inspired by residual learning; but instead of summing the feature activations, it combines them by concatenating the activations. In particular, all layer activation maps are passed on to all succeeding layers as shown in Figure 3.3b, which ensures that the gradients from even the final loss function is available to the very first layer. The original authors posit that this allows each layer access to 'global knowledge state' of the network, which eliminates the need to replicate states in every layer and reduces the number of required parameters. Since the concatenation operation is not possible as activation map sizes change, the densely connected layers are confined to blocks, with transition layers between them which scale activations appropriately. The DenseNet model might be a good candidate because it can obtain a competitive accuracy using much fewer parameters, which makes it suitable for implementation on an edge device with low resources and high performance requirements.

3.2.6. EfficientNet-B4

The EfficientNet family of network architectures [25] originated from efforts to devise principled ways to scale up the depth, width (filter sizes) and resolution (filters per layer) of networks, which has chiefly been done separately and arbitrarily in the literature. Scaling only in one dimension such as depth suffers diminishing returns on performance ability

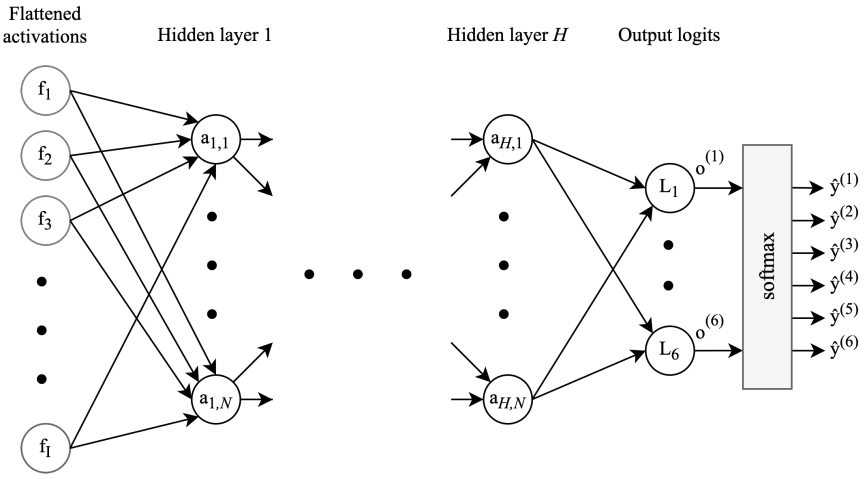


Figure 3.4: The common classifier block appended to the feature extractor layers of each model. The first layer depicted consists of the flattened feature activations from the feature extractor section of the network.

with respect to floating point operations per second (FLOPS) for inference. Using a grid search strategy, EfficientNet developers computed the fixed optimal ratio of 12:11:11.5 for depth-width-resolution compound scaling which results in the best accuracy while minimizing FLOPS. EfficientNet is a good candidate for the problem given its high accuracy and low computational load. EfficientNet-B4 is used due to memory constraints during training of the more sophisticated EfficientNet-B5 to EfficientNet-B7.

3.3. Hyperparameter Tuning

In this section, hyperparameters are designed for the objective performance metric of the model which is the validation accuracy (A_{V}). Architectures are determined using grid search and Bayesian optimization is used for the learning framework.

3.3.1. Architecture

The design of the classifier block is shown in Figure 3.4. All the proposed candidate models detailed in Section 3.2 have their fully-connected (FC) classifier layers removed (so that only the filters which extract and recognize features remain) before this classifier is attached. To be clear, the network then ends with flattened feature activations from the feature extractor section of the network, which essentially encodes the presence of different features. It is then the task of the classifier layers to learn (from random initializations) the logical combination that encodes the six classes.

The classifier consists of FC layers that are activated with ReLU functions. When designing each candidate model, the FC layer or set of FC layers are removed before training and this classifier is added to the remaining feature extractor layers (which are initialized with weights pre-trained on ImageNet). Therefore, the hyperparameters are the number

of hidden layers (H) and the number of neurons in each layer (N). To design H and N, a grid search strategy is used on each model with preliminary mid-range hyperparameters: an unfreezing depth of half the feature layers, a learning rate of $\alpha = 1 \times 10^{-4}$ and a batch size of $N_{\mathrm{B}} = 64$. All models are trained to a maximum of 35 epochs, and the performance of the highest-scoring epoch is recorded. The result of the grid search for each model is shown in Table 3.1.

All candidate models performed well on the dataset, obtaining accuracies of greater than 95% in most cases. A low training loss combined with a high A_{V} is desirable and architectures that show these results are chosen. More complex classifiers ($H = 3$ with $N = 2048$) show quick overfitting in most cases, especially for VGG-16. InceptionNet v3 and MobileNet v2 shows the smoothest training curves with the lowest training loss. Some training curves are shown in Figure 3.5

3.3.2. Learning

Learning hyperparameters determine the learning process during training. These are particularly important in the classifier design process since they determine whether the model overshoots or discovers the configuration of its internal parameters that results in the minimum possible error [10]. The learning hyperparameters are the learning rate (α), the momentum of the optimizer (β_1), the learning rate step size (E_{s}) and the mini-batch size (N_{B}). The learning rate α is the speed at which the model learns. Specifically, it is the magnitude by which the parameters in the model are updated in the gradient descent optimization step in Equation 2.15.

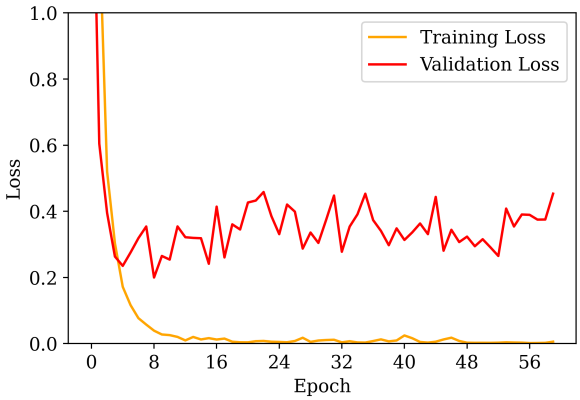
In the optimization sweep, the range $[1 \times 10^{-6}, 0.3]$ is searched on a log scale. It is unlikely that learning rates outside these bounds are necessary to consider. The learning rate is regarded in the literature as the most important hyperparameter to determine when designing neural networks [10]. In the same way, the rate of decay of the learning rate is important. If the learning rate remains too large, the model parameter set may simply end up bouncing around the minimum and not reach their optima. Thus, a step-based learning rate scheduler is applied which decays the learning rate by a factor 0.1 every E_{s} epochs. Step sizes frequencies in the range $[0, 40]$ are swept across in the optimization. The momentum β_1 is the decay rate of the exponential moving average of previous gradients in Equation 2.11.

The decay rate for the square of the previous gradients is kept as $\beta_2 = 0.9$ as suggested by the authors in the original Adam paper [7]. For β_1, the range $[0, 1)$ is available for sampling by the optimizer. Lastly, the mini-batch size N_{B} is also important in learning since too small of a batch size will result in a noisy gradient signal [18] over optimization steps.

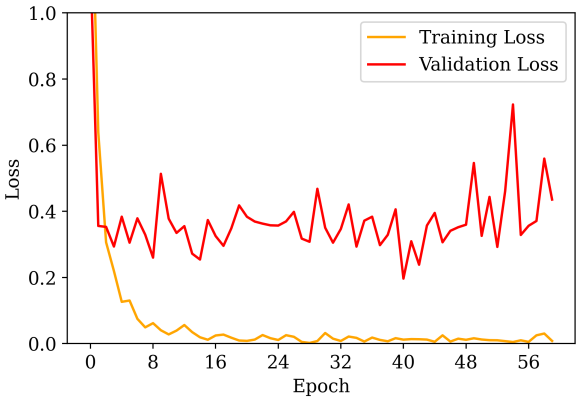
Given the number and possible scope of each, the optimal parameterization for these

	Hidden layers (H)	Neurons per layer (N)	Best epoch	Training loss	A_V [%]
VGG-16	2	1024*	25	0.0395	95.64
		2048	24	0.0389	95.32
	3	1024	9	0.4037	95.65
		2048	12	0.8355	96.65
MobileNet v2	2	1024	11	0.0645	94.69
		2048*	23	0.0519	95.79
	3	1024	16	0.0142	95.65
		2048	15	0.0178	95.73
ResNet-50	2	1024	19	0.0594	97.04
		2048	31	0.0911	96.38
	3	1024*	33	0.0068	97.37
		2048	11	0.0718	97.55
InceptionNet v3	2	1024	14	0.0016	96.31
		2048	34	0.0054	95.65
	3	1024	35	0.0150	95.65
		2048*	31	0.0312	97.70
DenseNet-121	2	1024	24	0.0127	96.05
		2048	34	0.0195	95.40
	3	1024	21	0.0592	96.05
		2048*	18	0.0097	96.05
EfficientNet-B4	2	1024	28	0.0109	95.48
		2048	14	0.0162	95.31
	3	1024*	30	0.0051	95.31
		2048	14	0.0095	95.31

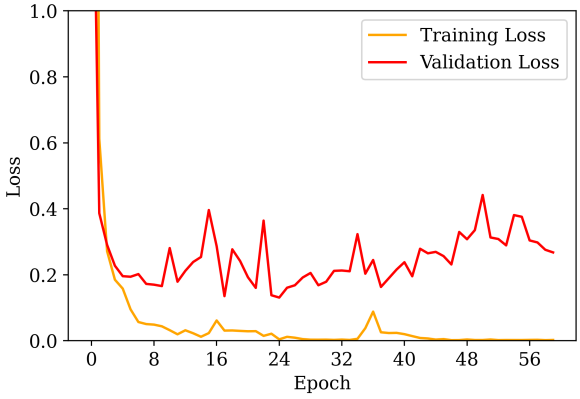
Table 3.1: Grid search results (best epoch, training loss and A_V) for the architecture hyperparameters (hidden layer number and size) per candidate model. Lower training loss is better and higher A_V is better. Asterisks indicate the architectures chosen for further design steps (e.g., MobileNet v2 with 2 hidden layers and 2048 neurons per layer is chosen).



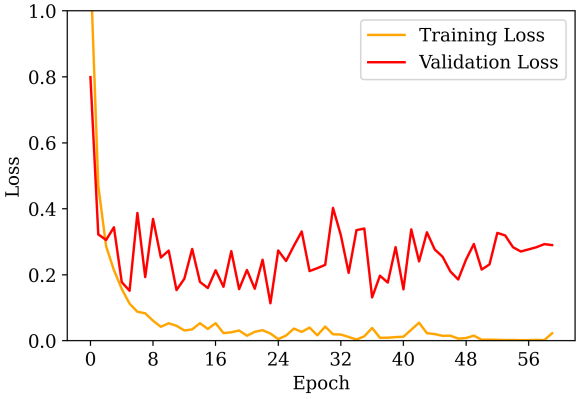
(a) ResNet-50, H=2, N=1024



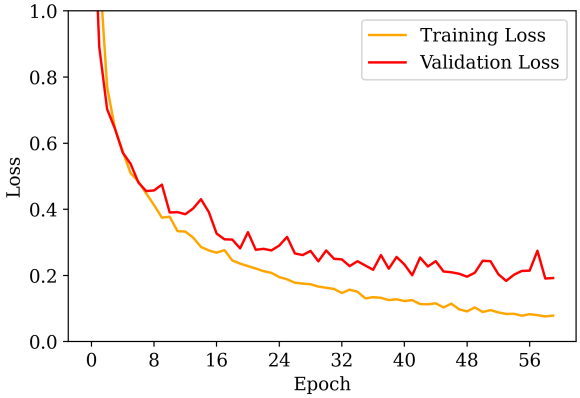
(b) DenseNet-121, H=3, N=2048



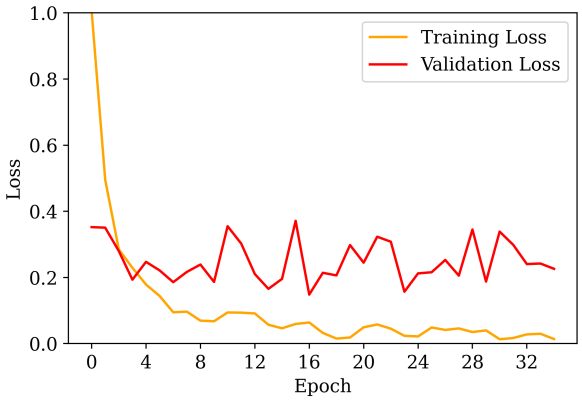
(c) InceptionNet v3, H=3, N=2048



(d) ResNet-50, H=3, N=1024



(e) MobileNet v2, H=3, N=2048



(f) ResNet-50, H=2, N=2048

Figure 3.5: Graphs showing the training metrics for some models as a function of epochs. Blue is training loss. Orange is the validation loss. Green is the validation accuracy. Training runs (b) and (c) show overfitting.

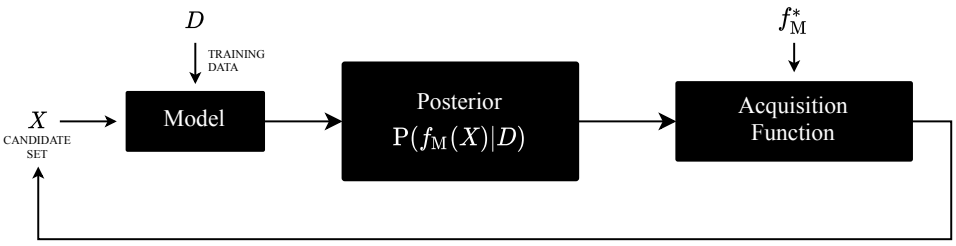


Figure 3.6: A diagram of the Bayesian optimization algorithm.

are determined by Bayesian optimization (BO) [26], which is illustrated in Figure 3.6. Bayesian optimization is used to determine the optimal parameterization for an unknown black-box model function f_{M} that is expensive to evaluate, like a neural network training. The output of f_{M} is a performance metric, in this case validation accuracy. By using a few observations from previous trials under some X (a set of candidate hyperparameters), BO builds a surrogate model for the function f_{M} to extrapolate for unknown parameterization points in the form of a posterior probability distribution $\mathrm{P}(f_{\mathrm{M}}(X)|\mathcal{D})$. The distribution is a Gaussian process (GP), which models the posterior on a finite set of points as a multivariate normal distribution.[1] The GP distribution has means $\mu(x)$, which models the expected accuracy in that region and covariance $\mathrm{E}(x)$, which models the uncertainty.

The next parameterization X to be evaluated for the model f_{M} must be the most useful set to find the maximum accuracy. An acquisition function selects the next X to be sampled by finding the maximum of the expected improvement (EI):

$$\mathrm{EI}(x) = \mathbb{E}\Big[\max(f_{\mathrm{M}}(x) - f^*_{\mathrm{M}}, 0)\Big] \tag{3.3}$$

where $f_{\mathrm{M}}{}^*$ is the current best observed outcome. The EI rewards evaluation of the objective f_{M} based on the likelihood that an X will provide the new highest accuracy. This promotes quick exploration of parameterizations with a high likelihood of providing a good accuracy and allows the optimization to hone in on the set that gives high accuracies [26]. Bayesian optimization generally works well for smooth response surfaces in practice, but does not prove useful for noisy responses from the model f_{M}.

Bayesian optimization is implemented in this project through the hyperparameter experimentation library `Ax`. 20 trials are used and each model is trained and evaluated with 35 epochs. The results from the algorithm run on all the candidate models is shown in Figure 3.2. The results show expected optimal learning rates, step sizes and batch sizes. The only surprising optima are all those for the models' momenta β_1, where the algorithm discovered small optimum values, of which two were almost zero (In contrast, the original Adam optimizer authors suggest a value of $\beta_1 = 0.9$ [7]). Nevertheless, all these hyperparameters values are used for further design.

[1] The formal mathematics of GPs and BO in general is not the focus of this project, but a simple explanation is worth providing.

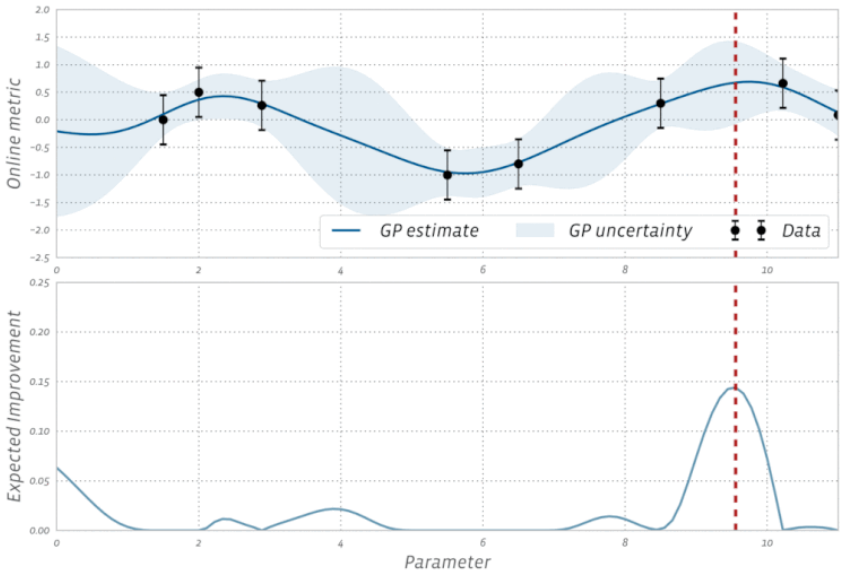
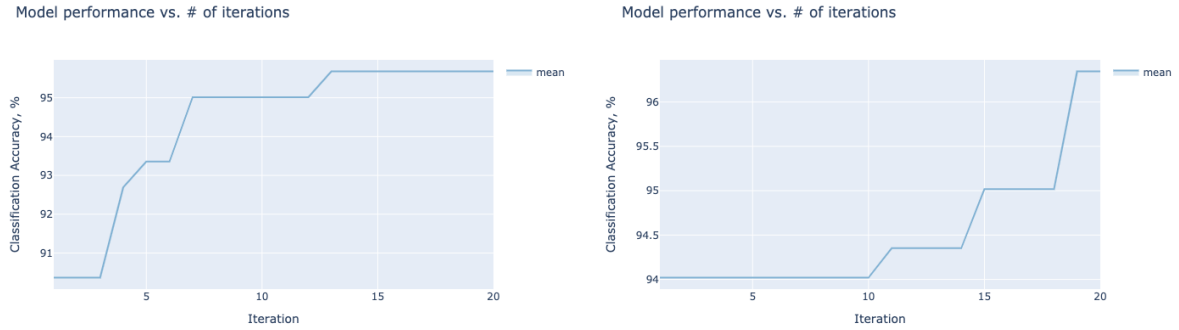


Figure 3.7: The surrogate model as a GP and the the associated EI for a univariate parameter space. The black dots indicate sampled parameterization coordinates. The red dashed line indicates the next parameterization coordinates to be sampled, as it maximizes the expected improvement. Illustration taken from [26].

	Learning rate (α)	Momentum (β_1)	Step size (E_s)	Batch size (N_B)	Mean accuracy
VGG-16	3.39152160e-05	0.3028278	8	29	0.946842966
MobileNet v2	5.53511380e-05	0.2817802	5	8	0.963455103
ResNet-50	2.68014042e-05	0.1788129	34	121	0.976744063
InceptionNet v3	1.42756669e-04	5.190841e-17	21	17	0.953488261
DenseNet-121	2.74235159e-04	4.149011e-17	7	14	0.956810554
EfficientNet-B4	1.16482985e-06	0.3249173	18	31	0.926910234

Table 3.2: Results from Bayesian optimization after 20 trials for each model.



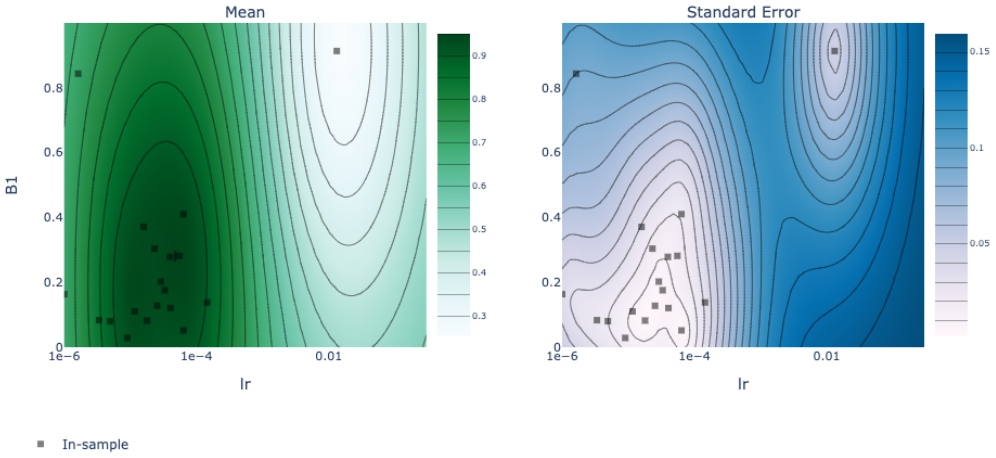
(a) DenseNet-121 **(b)** InceptionNet v3

Figure 3.8: Models showing increasing validation accuracies as the BO search discovers better learning hyperparameter sets each trial.

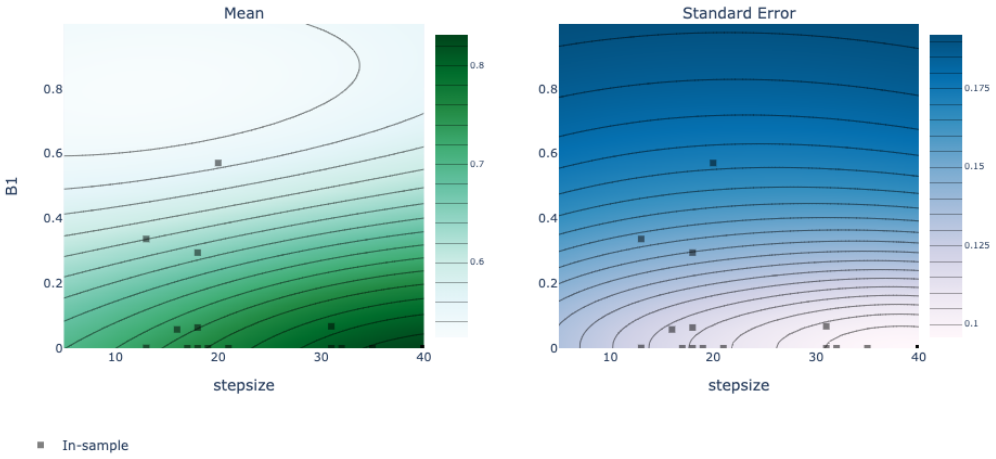
Figure 3.8 shows some models improving slightly for every iteration (trial) as better parameterizations are discovered. Figure 3.9 shows contour plots with respect two hyperparameters for some models. It is clear from the significant variation in the learning rate dimension that this is the most important hyperparameter. The BO algorithm performed most samples in the regions giving high accuracy estimations. Model training consistently shows affinity for a combination of small E_s with a small N_B, but a large E_s pairs well with a smaller β_1.

3.4. Chapter Summary

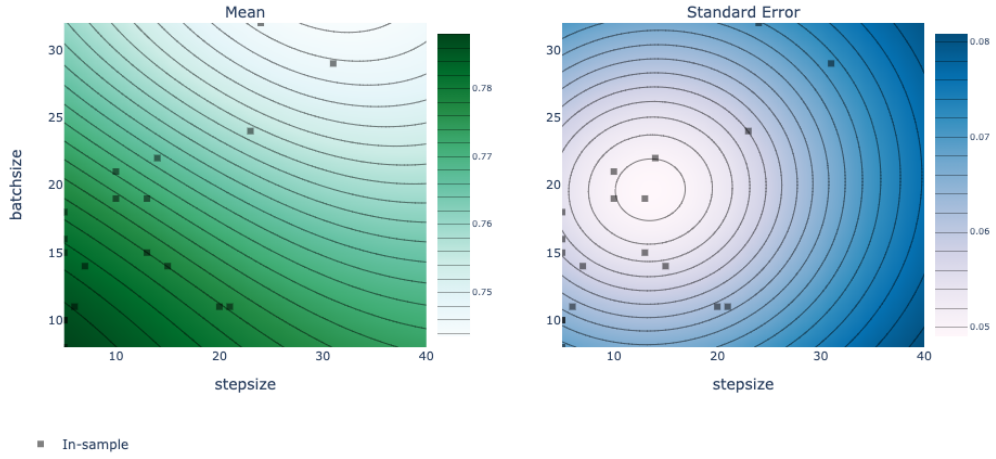
The six candidate models have been designed separately in terms of hyperparameters that determine network architecture and learning framework. Both laborious grid search and sequential design strategy were used to determine optimal parameterizations and all models obtain validation accuracies greater than 94%. The learning rate is the most significant consideration for design. Throughout design, half of the feature extractor layers were unfrozen for model fine-tuning. With these discovered parameterizations, the models are to be trained and compared by their performance on the test data.



(a) β_1 vs learning rate α plot for InceptionNet v3.



(b) β_1 vs step size E_s plot for MobileNet v2.



(c) Batch size N_B vs learning rate α plot for DenseNet-121.

Figure 3.9: Mean accuracy and associated standard errors (uncertainty) for BO results. As expected, model samples are near coordinates expected to give high validation accuracies.

Chapter 4

Model Testing

Models have been designed individually. In this chapter, they are trained on the training set twice: once with all layers unfrozen and once with only half the layers unfrozen. These trained networks are then evaluated on the test set for prediction and recall metrics. Their latency and computational load is also assessed and their classification performance is analyzed. Finally, the top-performing model is selected and further optimizations are made to improve latency and decrease recyclate contamination.

4.1. Testing Pipeline

4.1.1. Metrics

The models are compared using the F scores each obtains on the test data, which is a function of the macro-averaged per-class precision $\bar{P}$ and recall $\bar{R}$. The precision P_k for a class k is what fraction of waste images the model predicted as belonging to class k were correct:

$$P_k = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.1}$$

and recall R_k for a class k is what fraction of total waste items in class k the model could correctly classify:

$$R_k = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.2}$$

$\bar{P}$ and $\bar{R}$ are the arithmetic means of the class precision and recall scores respectively over all six classes:

$$\bar{P} = \frac{\sum_k^6 P_k}{6} \tag{4.3}$$

$$\bar{R} = \frac{\sum_k^6 R_k}{6} \tag{4.4}$$

and F score is the harmonic mean of the average precision and recall:

$$F = \frac{2\bar{P}\bar{R}}{\bar{P} + \bar{R}} \tag{4.5}$$

F score is preferred over accuracy as a performance metric because it captures a model's performance for problems with class imbalances [27]. This is applicable because the entire dataset contains more than twice as many images in the 'cardboard' class than the 'trash' class.

4.1.2. Practical Constraints

Along with the performance of each model, their demand on the hardware and their inference latency are important considerations for practical implementation. The RAM available on an edge device will pose a limitation on the peak memory use of the inference program. The memory requirements to run a CNN chiefly come from three sources: parameters of the CNN, intermediate network data storage, and the computation workspace [28]. For this reason, the peak memory use by the inference program for both loading a model and classifying a single image is measured empirically using the `memory-profiler` module. To further supplement the comparison stage, an estimation of the FLOPs performed for a single image inference by a model is obtained from calculations performed by either the original authors of the models or by empirical measurements from third party sources.

Users will be unlikely to use the system if the classification latency is too great. Therefore, inference time for a single image is also measured empirically, but only on the development machine (a Google Colaboratory Compute Engine) and so will only be useful for comparison with respect to other models. Only the first prediction of any single image is timed to factor out the effect of memory caching. A final inference time is measured during implementation in Section 5.3.

4.1.3. Comprehension Techniques

CNNs are often regarded as opaque black-box functions, but several comprehension techniques may be used to interpret the decision-making process of a classification model. It is import to try to understand and explain the types of misclassifications the final system will make to assess its limitations and guide improvements. While model interpretability is an active area of research, the PyTorch library `Captum` provides limited model interpretability algorithms and is used in this chapter for attribution visualization (which image features contribute to prediction decisions) via two techniques. Firstly, gradient-based attribution is performed using SHAP (Shapley Additive Explanations) introduced in [29] which quantifies the magnitude of the contribution that each input feature brings to the final model decision by approximating the marginal contributions of each using Shapley values from game theory. The algorithm is beyond the scope of this report.

Secondly, occlusion-based attribution slides a grey window across the input image with predetermined dimensions $W_{\mathrm{L}} \times W_{\mathrm{L}}$ and stride W_{s}. The model is evaluated at each instance to determine the effect of occluding the feature on the model error. Those feature

Trained models	Precision per class [%]						Recall per class [%]								
	cardboard	glass	metal	paper	plastic	trash	cardboard	glass	metal	paper	plastic	trash	$\bar{P}$	$\bar{R}$	F
VGG-16 (H)	**100**	90	96	92	68	38	94	98	77	93	71	**35**	80.7	78.0	79.3
VGG-16 (F)	96	**97**	94	93	51	**56**	**100**	93	**85**	88	**97**	31	81.2	**82.3**	**81.7**
MobileNet v2 (H)	94	94	88	90	49	6	98	91	69	90	72	6	70.2	71.0	70.6
MobileNet v2 (F)	96	**97**	94	93	60	19	96	96	**85**	85	83	16	76.8	76.5	76.6
ResNet-50 (H)	98	94	**100**	93	65	19	87	**100**	72	93	88	20	78.1	76.7	77.4
ResNet-50 (F)	96	95	98	93	58	50	96	98	84	93	80	28	**81.7**	79.8	80.7
InceptionNet v3 (H)	96	**97**	90	85	60	25	96	91	66	**94**	76	14	75.5	72.8	74.1
InceptionNet v3 (F)	96	94	94	90	60	38	98	97	81	84	81	25	77.7	78.7	78.2
DenseNet-121 (H)	98	91	96	89	58	44	98	99	76	90	75	24	77.0	79.3	78.1
DenseNet-121 (F)	98	90	98	89	54	46	88	95	78	90	75	20	79.2	74.3	76.7
EfficientNet-B4 (H)	96	95	98	**97**	49	6	96	95	70	80	88	17	74.3	73.5	73.9
EfficientNet-B4 (F)	92	94	92	92	**91**	0	98	94	79	78	70	0	69.8	71.8	70.8

Table 4.1: Classification results on the test set for trained candidate models with different unfreezing depths. (H) means half unfreezing and (F) indicates full unfreezing.

whose occlusions realize the largest change in error are assumed to contribute the most to the model's decision.

4.2. Test Results

4.2.1. Classification Performance

Each trained model is tested on the held-out test set and their average class precision $\bar{P}$, recall $\bar{R}$ and F score are measured. The results are shown in Table 4.1. The top-performer by these metrics is counter-intuitively the most unsophisticated model, VGG-16 with all layers unfrozen. Other models with good results include ResNet-50 with full unfreezing and InceptionNet v3 with full unfreezing. ResNet-50 might have performed well because its relatively low complexity prevents overfitting and its use of residual learning ensures deep weight adjustments for filters. The performance of InceptionNet v3 may be attributed to its ability to specialize on finding intricate low-level feature representations in the data. The worst performer is MobileNet v2, which is expected given its deliberately low complexity sacrificing its ability to capture intricate patterns in the images. EfficientNet-B4 failed to produce competitive results despite its meticulous design and preference in the deep learning community.

Trained models	FLOPs [billions]	Memory usage [MB]		Inference time [ms]
		Loading	Inference	
VGG-16	16.00	731.3	367.5	10260
MobileNet v2	**0.58**	59.2	**141.0**	**960**
ResNet-50	4.00	134.3	164.3	2404
InceptionNet v3	6.00	**53.9**	253.3	2207
DenseNet-121	8.00	68.7	253.4	2574
EfficientNet-B4	4.20	122.6	266.4	2564

Table 4.2: Resource burden and inference time for each candidate model for a single image.

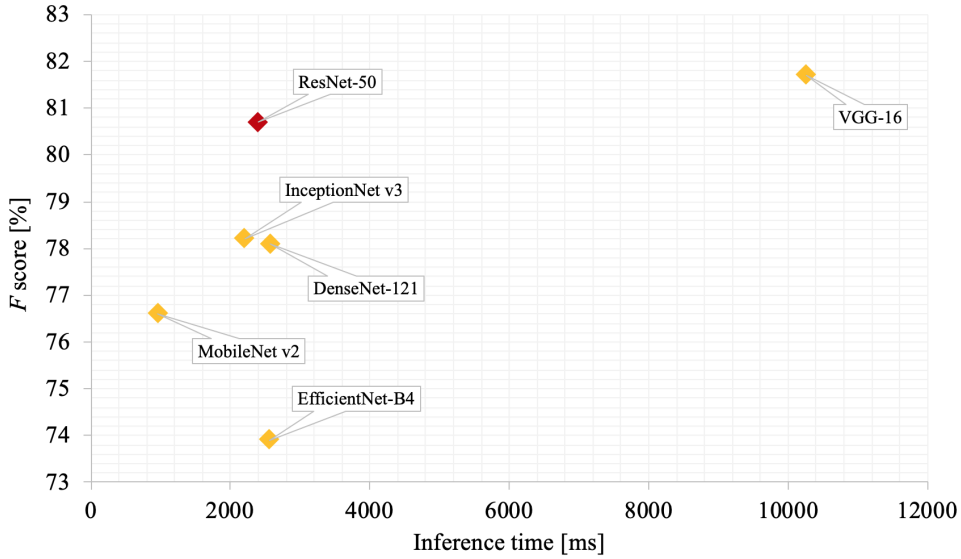


Figure 4.1: A plot comparing the candidate models' performances against inference time. The final selected model, ResNet-50, is shown in red.

4.2.2. Cost and Latency

Each model is profiled for time and memory usage while making a single cold-start classification (no memory caching) and the results are shown in Table 4.2 along with the estimated FLOPs for one classification. FLOP data are taken from [30], except data for EfficientNet-B4 which is taken from the original paper [25]. Inference time correlates heavily with FLOPs and inference memory usage. Despite its superior classification performance, VGG-16 takes more than ten seconds and 16 GFLOPs for a classification, which disqualifies it for implementation in a practical system. MobileNet v2 observes the lowest latency and resource burden by a large margin, but suffers from poorer classification performance.

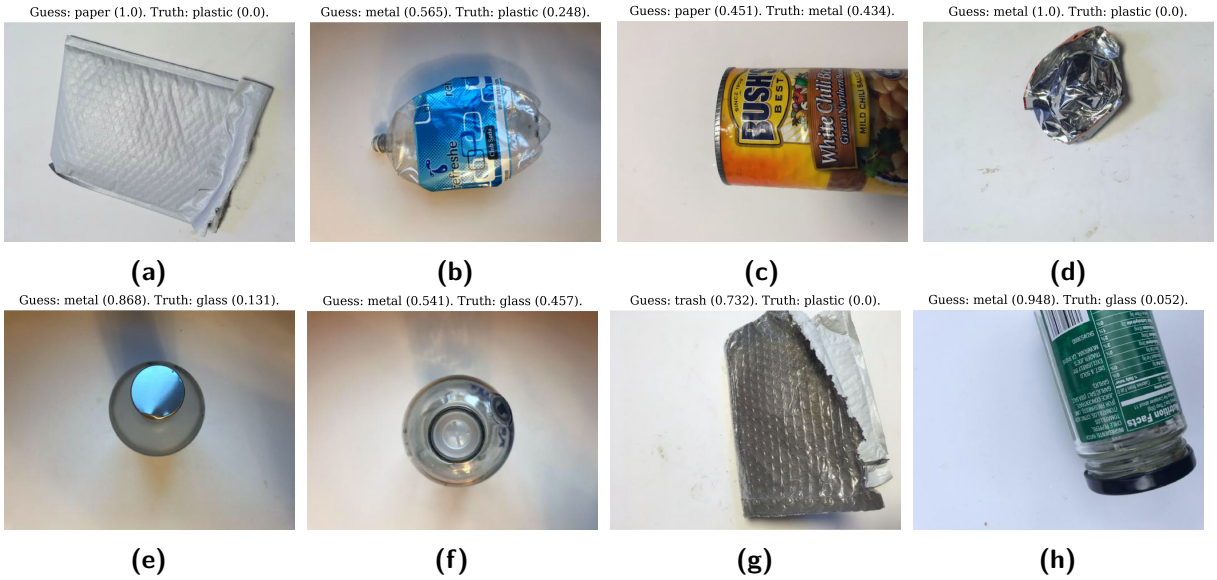


Figure 4.2: Samples from the test set that were misclassified, showing the predicted class and the ground truth class along with the confidence of each.

4.2.3. Final Model Selection

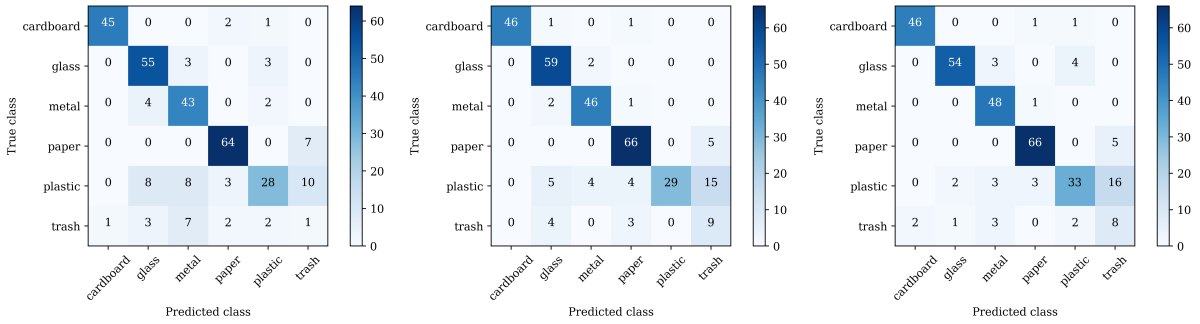
Selecting a final model is a decision based on the trade-off between classification performance and latency. Figure 4.1 plots the F score against the inference time. ResNet-50 (indicated in red) observes a comparable score to VGG-16, which is the best performer, but has a latency as low as other candidates. Notably, as shown in Table 4.1, it performs second-best on 'trash' item classification in both precision and recall. For this reason, ResNet-50 is the favoured candidate model for the problem and is used for implementation.

4.3. Model Analyses

4.3.1. Misclassifications and Confusion

Figure 4.2 shows some test samples that were misclassified by ResNet-50, along with the predicted class, ground truth and softmax probability of each. For most of these images, it is difficult for even a human to know with confidence to which recycling category the waste items belong. Confusion matrices can be used to understand the weaknesses of the candidate models and the types of misclassifications the final system will make. Confusion matrices table tallies for predictions against ground truth labels to see where the classifier confuses one class for another. Figure 4.3 shows the confusion matrices for the worst classifier model, the best, and the final chosen model.

It is evident from all three confusion matrices that the models could easily learn the features of a cardboard item and a glass item, but performed worse on plastic and the worst on trash. The chosen ResNet-50 classified more trash-labelled items incorrectly than



(a) MobileNet v2 **(b)** VGG-16 **(c)** ResNet-50

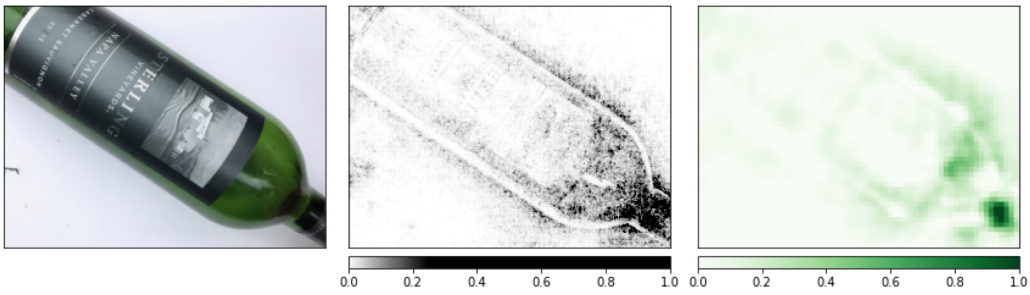
Figure 4.3: Test set confusion matrices for the (a) the worst performer, (b) the top performer and (c) the final selected model.

correctly. The model could classify recycling materials significantly greater than chance in every circumstance, but could not identify non-recyclable items with a precision greater than 56% or a recall greater than 35%.

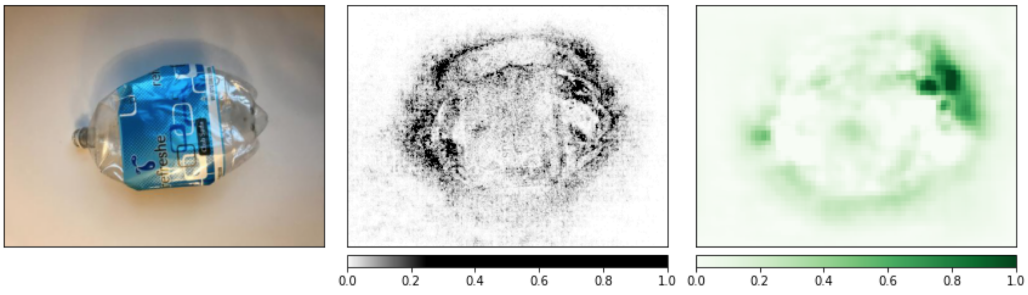
It is clear that the final model test accuracies of all the models are significantly lower than the validation accuracies found in Section 3.3 and test accuracies achieved by Bircanoglu et al. [15] and White et al. [16]. Swapping the test set for the validation set during training and testing phases shows that the models consistently perform poorly on the test set specifically. The most likely explanation for this discrepancy is that the test set contains waste images that are more difficult to classify than the validation set and that the test set features deviate more significantly from the training set than those of the validation set do. The TrashNet dataset contains a few instances where there are multiple different photographs of the same waste item. It is likely that both Bircanoglu et al. and White et al. use random, stratified dataset splitting (which is a common practice), as opposed to the method used in the project, where simply the final 10% of the images in each folder is assigned to the test set. This implies that previous approaches obtained good results because their models specialized on certain waste items found in both training and test sets. As such, the result obtained in this report more likely reflect the actual performance of a classifier on the dataset.

4.3.2. Feature Attribution

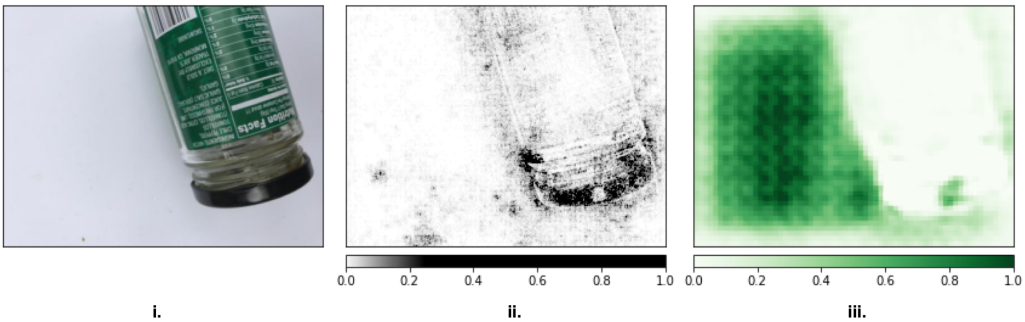
It is possible to understand which image elements lead to confusions by using SHAP and occlusions-based attribution visualization. Occlusion is performed with a window size of $W_{\mathrm{L}} = 15$ and stride $W_{\mathrm{s}} = 8$. This is shown in Figure 4.4. Figure 4.4a shows a wine bottle correctly classified as ‘glass’ and the visualization of its attribution shows that the glass neck of bottle above the product label contributed the most to the final classifier decision. This validates that the ResNet-50 model learned to classify using low-level features as intended.



(a) A glass wine bottle correctly classified as 'glass'. It is clear that the product label made little contribution to the classifier's decision.



(b) A plastic bottle incorrectly classified as 'metal'. The classifier seemingly used shadows and reflections for this decision.



(c) A glass jar bottle incorrectly classified as 'metal'. The SHAP method in (ii) shows that the metal lid contributed to this decision.

Figure 4.4: Attribution visualizations for three images in the test set. Figures under (ii) visualize the SHAP attributions with darker black representing a larger magnitude of attribution. Figures under (iii) visualize the occlusion-based attributions, with darker green representing a larger magnitude of attribution.

The misclassified test image shown in Figure 4.2b was labelled 'plastic', but predicted by ResNet-50 as 'metal'. The attribution investigation in Figure 4.4b shows that the highly reflective surface and the shadows somehow prompted a 'metal' classification.

The misclassified test image shown in Figure 4.2h was labelled 'glass', but predicted by ResNet-50 as 'metal'. Figure 4.4c visualizes the attribution magnitude of its local features, showing with both SHAP and occlusions that the model used the properties of the metallic lid to make the final decision, and disregarded the glass properties. The occlusion-based method shows large contributions from the empty portion of the photo, which is likely because a remote grey square in this area is not a feature the model has seen before and this subsequently causes a significant error. From these investigations, it is clear that the user should be prompted to separate product components that are of different materials from each other and have them classified individually.

4.4. Improvements

4.4.1. Decision Threshold

A decision threshold for inference is implemented to mitigate the risk of misclassifications causing contamination in recycling by choosing to rather classify images with low confidence scores as 'trash'. If a decision threshold λ is used, the greatest softmax output $\hat{y}$ must be greater than λ (i.e. the model must be sufficiently confident in its classification), otherwise the image is assigned to 'trash'. If no decision threshold is used, simply the class with the greatest softmax probability is selected. Inspection of misclassified examples in Figure 4.2 suggests a hypothesis that contamination can be reduced by implementing such a method. To find the best threshold λ that will improve model classification performance, the model is evaluated for F score on the test set using values for λ in the range $[0, 1]$ in increments of 0.01.

The search result is shown in Figure 4.5, which reveals that a decision threshold of $\lambda = 0.95$ serves to significantly improve the test set F score from 80.1% to 83.5%. The results show that images classified by the model with some uncertainty actually belongs to the 'trash' class. Confusion matrices for the model on the test set before and after the implementation is shown in Figure 4.6, which reveals more classifications in the trash column.

4.4.2. Quantization

Quantization [31] is a technique to reduce model size and latency Normally, every weight, bias and activation matrix value is stored in memory in a 32-bit floating-point representation (FP32). This maintains high precision throughout forward feeds during inference,

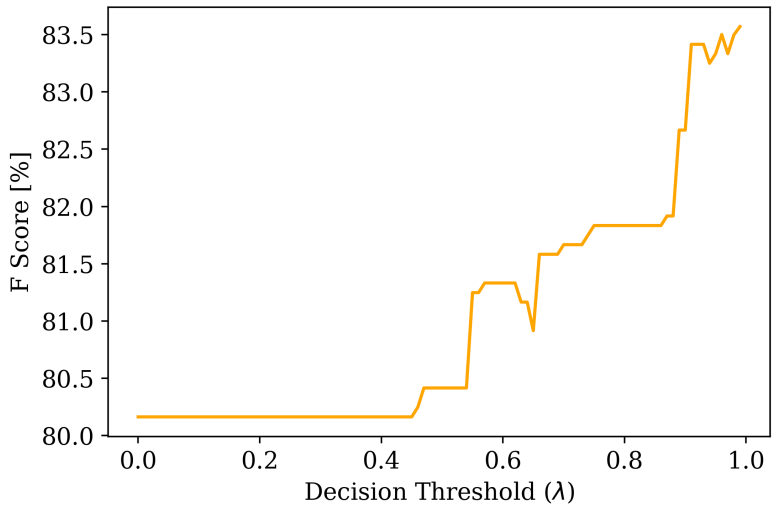
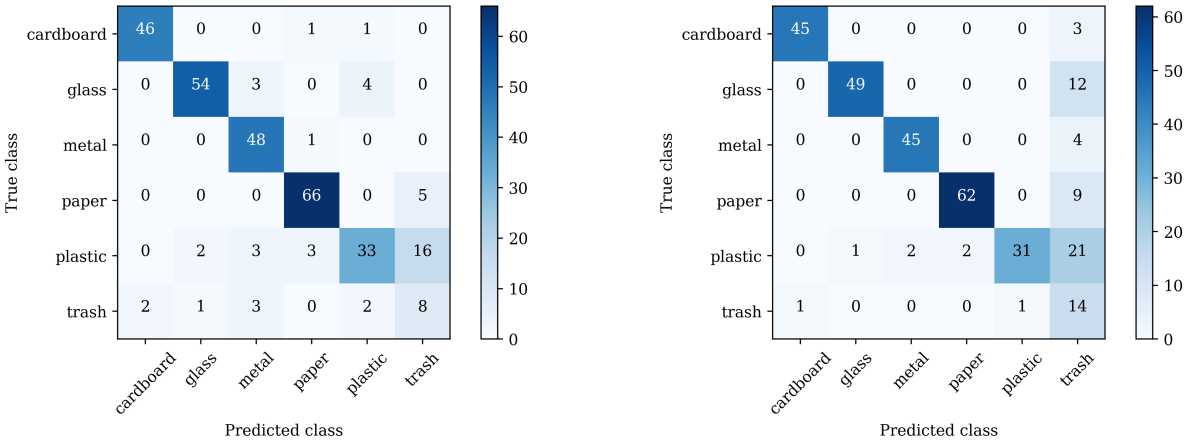


Figure 4.5: Search results for best F score on the test set given different decision thresholds.



(a) Test set confusion without the decision threshold.

(b) Test set confusion with the decision threshold.

Figure 4.6: Test set confusion matrices for (a) ResNet-50 with no threshold and (b) ResNet-50 with the threshold of $\lambda = 0.95$, showing fewer off-diagonal classifications for recycling categories, i.e. less contamination.

Quantization	$\bar{P}$	$\bar{R}$	F	Memory usage [MB]		Inference time [ms]
				Loading	Inference	
None	**81.7**	**79.8**	**80.7**	134.3	164.3	2404
INT8 QAT	78.5	77.5	78.0	**27.6**	**39.9**	**920**

Table 4.3: Comparison of unquantized ResNet-50 vs ResNet-50 with quantization-aware training.

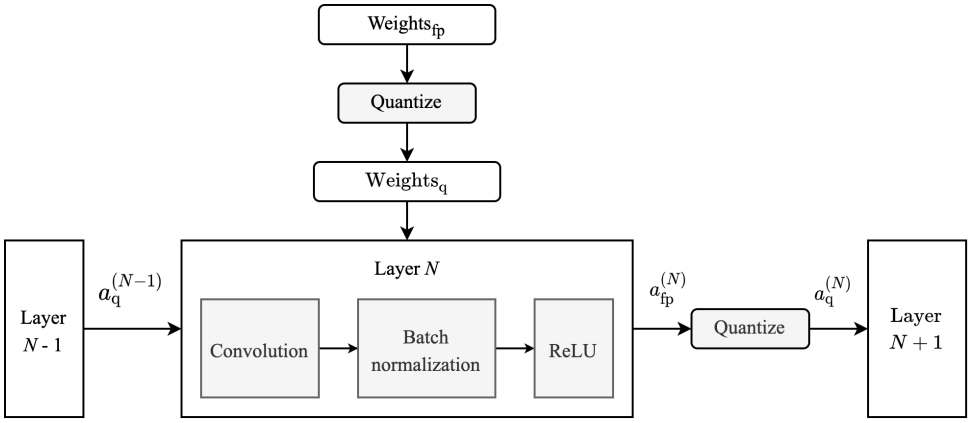


Figure 4.7: An illustration of quantization-aware training showing where parameters are quantized from FP32 to INT8-mimicking representations.

but multiply-and-accumulate (MAC) operations with FP32 precision are computationally expensive and demand more memory. Quantization refers to using lower precision representations to store values inside neural networks, which reduces model size and optimizes efficiency, but sacrifices accuracy. The process of quantization-aware training (QAT) is used in this work and is illustrated in Figure 4.7. QAT rounds all float values to mimic 8-bit integer representation (INT8) during training, but weight adjustment is still performed on float values. FP32 copies of the weights are kept during the training process ($\text{weights}_{\text{fp}}$) and a FP32 copy of the activations are used (a_{fp}). These copies accumulate changes from the gradients without precision loss and is also backpropagated through. This maintains high accuracy throughout training. During converting and exporting for inference, fully quantized INT8 storage is used.

The results of training on a quantization-enabled ResNet-50 model and converting to INT8 are shown in Table 4.3. There was a 3.3% decrease in F score, but a significant 75% decrease in memory consumption and 60% decrease in latency for inference. If latency proves too great during implementation, use of a quantized low-precision classifier is advisable.

4.5. Chapter Summary

In this chapter, the designed and trained candidate classifier models were tested for classification performance and inference speed. The best model was determined to be ResNet-50 with all layers unfrozen for parameter learning, given its competitive F score on the test data as well as acceptable size and speed of classification. Model intepretability techniques showed that the ResNet-50 model uses the appropriate low-level features as intended, but suffers confusion from products with diverse material composition. Decision thresholds for classification were tested and were shown to reduce cross-contamination of recycling categories and improve F score. Quantization lowered the storage precision of the model's numerical values and proved to decrease latency by 60%. With the final model chosen and optimized, it can be implemented on the edge device for final system testing.

Chapter 5

System Implementation

Following the selection of the final waste classification model, the classifier software is set up on a Raspberry Pi. This chapter discusses the system layout, the user interface design and the results from testing on images of waste presented in the field.

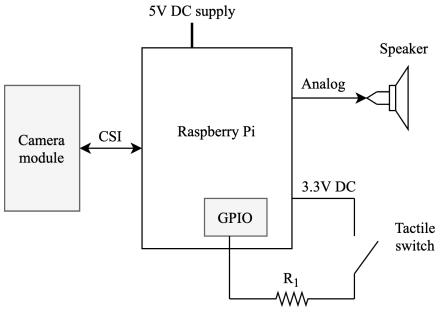
5.1. System Overview

The classifier model is loaded into memory on the Raspberry Pi using an ARM-compatible version of the PyTorch framework. The Pi instructs an attached digital camera device to take a still image when it senses that a button has been pushed. The image is cropped to 384×384 pixels, converted to a PyTorch matrix, and fed to the model for inference. When the prediction is available, the software makes a decision based on the top class softmax probability with a decision threshold λ. A block diagram and a flowchart of the classification process are shown in Figure 5.1.

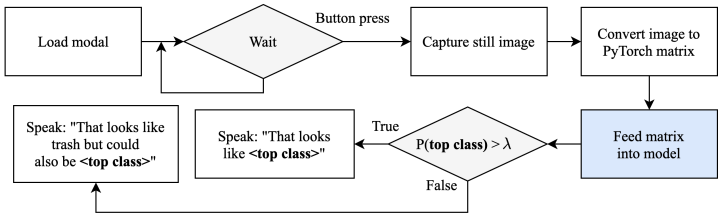
5.2. User Interface

The user interface consists of a button for the user to take a digital photograph of the presented waste and a speaker for verbal feedback of the recycling category prediction.

The tactile switch acts as the button the user depresses to capture a still image of the presented litter. It is connected between a 3.3 V pin and GPIO pin 10 on the Pi, with a



(a) A block diagram of the physical system.



(b) A flow diagram showing a use case for the system.

Figure 5.1: Illustrations of the design of the final classification system.

resistor R_1 in series to limit current draw. Using a resistance of 1 kΩ gives a current of:

$$I = \frac{3.3\,\text{V}}{1\,\text{k}\Omega} = 3.3\,\text{mA}$$

which is below the 160 mA current limit for a GPIO pin on the Raspberry Pi 4 [32]. The library `RPi.GPIO` is used which provides hardware abstraction for GPIO event listening. A high signal on pin 10 triggers a still image capture from the digital camera. The camera module is a Raspberry Pi Camera Module v2 and communicates with the Pi using the library `PiCamera` via the camera serial interface (CSI).

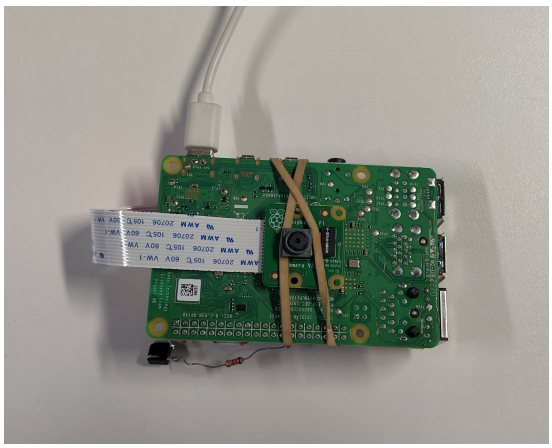
5.2.1. Audio Feedback

Text-to-speech (TTS) audio files for each class in the MP3 format are generated by online tools and loaded onto the Pi. If the confidence of the top class is greater than the threshold λ, the appropriate pre-generated speech audio file is played for that class with the format: "That looks like <**top class**>". When the confidence is lower than λ, the audio file for the 'trash' class is played and a caveat is added: "... but it could also be <**top class**>".

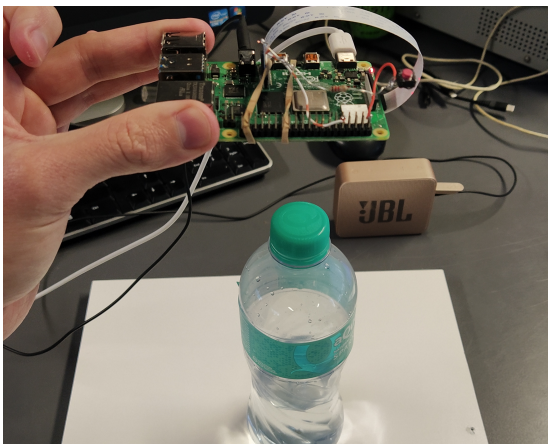
A value of $\lambda = 0.95$ is chosen for a preliminary threshold and is refined after testing. The audio is played using the module `playsound` through a common computer speaker accessory connected via the 3.5 mm headphone jack auxiliary port.

5.3. Final System Tests

A photograph of the Raspberry Pi implementation is shown in Figure 5.2a, and a classification test is shown in Figure 5.2b. System tests are conducted using waste items. The results are shown in Figure 5.3. The system is able to classify all presented waste correctly and play the correct audio file, except the waste item in Figure 5.3e.



(a) A photo of the edge classifier device.



(b) A photo of a classification test.

Figure 5.2: Images of the final system implementation.

(a) plastic **(b)** metal **(c)** paper **(d)** cardboard **(e)** plastic

Figure 5.3: Waste samples as they were classified when testing the final implemented system. (e) was predicted to be plastic, but is more accurately non-recyclable trash.

	Capture and save [s]	Open and transform [s]	Prediction [s]	Total [s]
Unquantized model	0.3355	0.1607	8.2120	8.7082
Quantized model	0.3148	0.1757	2.7130	3.2035
Quantized model + virtual storage	0.2925	0.0379	2.6077	2.9384

Table 5.1: Results from latency improvement methods. Each phase in the inference program in profiled for time.

5.4. Final Optimizations

Final optimizations of the system relate to the inference time. On the edge device, more than eight seconds pass before the classification is announced to the user. Therefore, a quantized version of the ResNet-50 model as described in Section 4.4.2 is used for the system. This serves to decrease the inference time by 63% from approximately 8.7 seconds to approximately 3.2 seconds.

Lastly, the `PiCamera` library methods save the captured still image to disk, which is then loaded back into memory by the main inference program and transformed into a PyTorch image data matrix. Transfer between storage and memory may take up a disproportionate slice of time during inference. To maintain image data in memory for fast access, a virtual folder is mounted using Linux's virtual file storage paradigm. `tmpfs` enables directories to appear as mounted file systems as if on a persistent storage device, but data is stored in volatile memory. A `tmpfs` implementation served to decrease saving and loading time. The results of both system optimization approaches and their effects on latency are shown in Table 5.1.

5.5. Chapter Summary

The final implementation of the system proved successful and performed as intended. The edge device was able to classify four out of the five presented waste items found in the environment. The classification latency could successfully be reduced to 2.9 seconds

using quantization and virtual storage. A significant shortcoming of the system is that inference remains slow and is considerably slower than the approximately 0.9 second latency discovered when testing in the development environment.

Chapter 6

Conclusion

This project sought to use state-of-the-art machine learning methods to design a solid waste classifier that is accurate enough to be a meaningful addition in a smart bin system and obeys resource constraints sufficiently to be practically realizable. Where machine learning practitioners often develop more advanced learning algorithms to improve on previous theoretical work, this project used proven and accessible techniques to build and implement a classifier successfully using iterative approaches, experimentation, and meticulous optimization.

6.1. Summary

Six candidate convolutional neural network (CNN) models that have shown appreciable accuracy performance in the past were selected from a model zoo. Their feature extractor architectures along with their parameters that are pretrained on images from the ImageNet were used and a common classifier structure was appended to all models. Hyperparameters that describe the classifier architecture and that describe the learning speed and scope were tuned for maximum validation accuracy using grid search techniques and the more sophisticated Bayesian optimization algorithm.

Once each model has been designed appropriately, they were trained with multiple epochs using early stopping and were compared with regards to their classification performance by F score, resource burden and latency. The model with the most suitable metrics was chosen and the further analysis was performed to gain transparency and interpretability. Experimentation to reduce misclassification using decision thresholds and improve latency while minimizing accuracy loss using quantization proved fruitful. Finally, the classifier was implemented on an edge device. The system was successfully and consistently able to recognize waste as belonging to certain material categories and classify them appropriately by a computer speech interface. Measures were introduced to reduce latency using quantization and virtual storage.

6.2. Main Findings

Automated classification of waste items is not an easy computer vision task. Litter come in a wide variety of shapes and sizes and are often deformed and unrecognizable. The issues are exacerbated by consumer products which are often made of multiple constituent materials which must be separated, and by contamination of recyclable waste.

Accuracy in the validation phase yielded state-of-the-art results. The best validation accuracy of 97.7% was achieved by InceptionNet v3 with 3 hidden layers and 2048 neurons per layer. However, VGG-16 outperformed all other CNNs at only 81.7% on the test set after all hyperparameters were optimally designed. The results indicated that the validation image set was likely similar to the training set in terms of learnable features. Bayesian optimization as a strategy to design for optimal hyperparameters significantly expedites design phases and proves to automatically illuminate parameterization sets that improve the model accuracy.

The test set appeared to be a more difficult problem set for the models to classify, which leads to the conclusion that the test accuracies obtained here are lower than one would experience in the field, and that previous work by Bircanoglu et al. [15] and White et al. [16], obtaining 95% and 97% respectively, most likely made use of randomized, stratified dataset splitting, which included in the test set features that are more representative of the training set, but also sacrifice the veracity of their final accuracy claims.

Furthermore, the techniques to reduce recyclate cross-contamination using decision thresholds proceeded to improve F score by 2.4%. Techniques to reduce model size and latency using quantization-aware training reduced model inference time by 60% and memory load by 75%, while sacrificing only a 3.3% decrease in F score.

Lastly, the system was successfully implemented on a Raspberry Pi 4, where it showed acceptable classification performance. Use of a quantized model improved the inference speed from greater than eight seconds to an acceptable 2.9 second latency on average.

6.3. Future Work

Future work on the problem of waste classification might benefit from a larger and more diverse dataset with a wider variety of backgrounds. A different approach to using deep learning architectures is also worth considering. Rather than fine-tuning models on a pre-populated dataset, one-shot or few-shot learning techniques, which learn image classification using one or very few training examples from users, will have the benefit of: (i) incorporating user contributions for crowd-sourcing training data (so-called human-in-the-loop techniques) if training at the edge or on cloud backends is feasible, and (ii) allowing classifier models to specialize on the consumer waste that is found in that public milieu.

Bibliography

[1] M. Koerth, "The Era Of Easy Recycling May Be Coming To An End," 2019. [Online]. Available: https://fivethirtyeight.com/features/the-era-of-easy-recycling-may-be-coming-to-an-end/

[2] M. Aljaradin, K. Persson, and H. Al-Itawi, "Public Awareness and Willingness for Recycle in Jordan," *International Journal of Academic Research*, vol. 3, no. 1, pp. 507–510, 2011.

[3] F. Chen, X. Li, J. Ma, Y. Yang, and G.-J. Liu, "An Exploration of the Impacts of Compulsory Source-Separated Policy in Improving Household Solid Waste-Sorting in Pilot Megacities, China: A Case Study of Nanjing," *Sustainability*, vol. 10, no. 5, 2015.

[4] G. Thung and M. Yang, "Classification of trash for recyclability status," 2016. [Online]. Available: http://cs229.stanford.edu/proj2016/report/ThungYang-ClassificationOfTrashForRecyclabilityStatus-report.pdf

[5] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

[6] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, vol. 27, 2014, pp. 3320–3328.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," vol. 25, 2012, pp. 1097–1105.

[9] A. K. Jain, Jianchang Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[10] H. Aghdam and E. Heravi, *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer

International Publishing, 2017. [Online]. Available: https://books.google.co.za/books?id=ZZXJAQAACAAJ

[11] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. Sebastopol, CA: O'Reilly Media.

[12] X. Zhou, "Understanding the convolutional neural networks with gradient descent and backpropagation," *Journal of Physics: Conference Series*, vol. 1004, p. 012028, 2018. [Online]. Available: https://doi.org/10.1088%2F1742-6596%2F1004%2F1%2F012028

[13] F. Chollet, "Transfer learning fine-tuning," 2020. [Online]. Available: https://keras.io/guides/transfer_learning/

[14] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness," 2019.

[15] C. Bircanoğlu, M. Atay, F. Beşer, Genç, and M. A. Kızrak, "Recyclenet: Intelligent waste sorting using deep neural networks," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*, 2018, pp. 1–7.

[16] G. White, C. Cabrera, A. Palade, F. Li, and S. Clarke, "Wastenet: Waste classification at the edge for smart bins," *ArXiv*, vol. abs/2006.05873, 2020.

[17] U. Jaitley, "Why data normalization is necessary for machine learning models," 2018. [Online]. Available: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029

[18] J. Brownlee, "A gentle introduction to mini-batch gradient descent and how to configure batch size," 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[21] ——, "Identity mappings in deep residual networks," 2016.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[24] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[25] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[26] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "Botorch: Programmable bayesian optimization in pytorch," 2020.

[27] Y. Sasaki, "The truth of the f-measure," 2007. [Online]. Available: https://www.cs.odu.edu/~mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07.pdf

[28] Z. Lu, S. Rallapalli, K. Chan, and T. Porta, "Modeling the resource requirements of convolutional neural networks on mobile devices," 2017.

[29] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[30] S. Albanie, "convnet-burden," 2019. [Online]. Available: https://github.com/albanie/convnet-burden

[31] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[32] www.raspberrypi.org, "General purpose input/output pins on the raspberry pi." [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md

Appendix A

Dataset Split

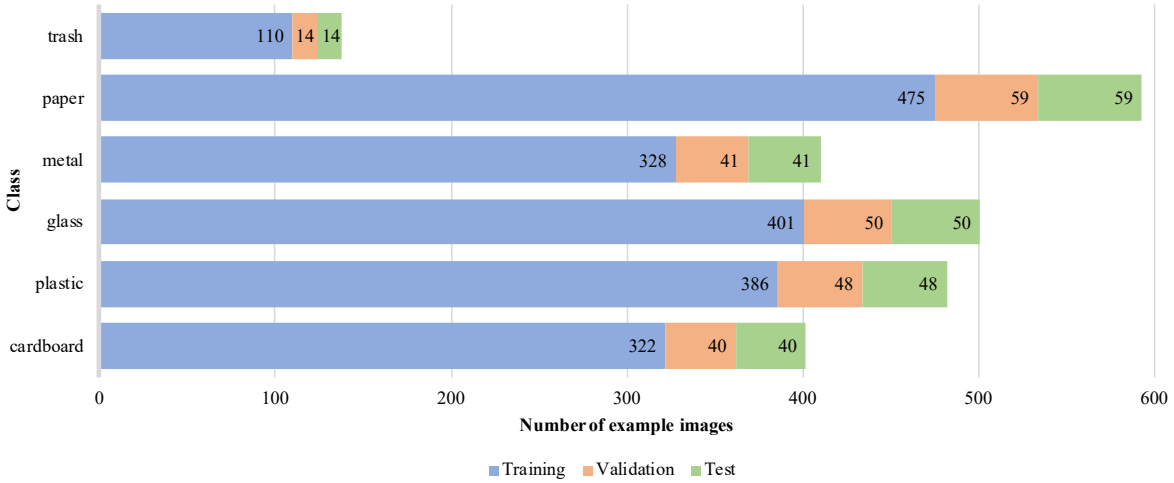


Figure A.1: The dataset split into training, validation and test sets per class.

Appendix B

Project Planning Schedule

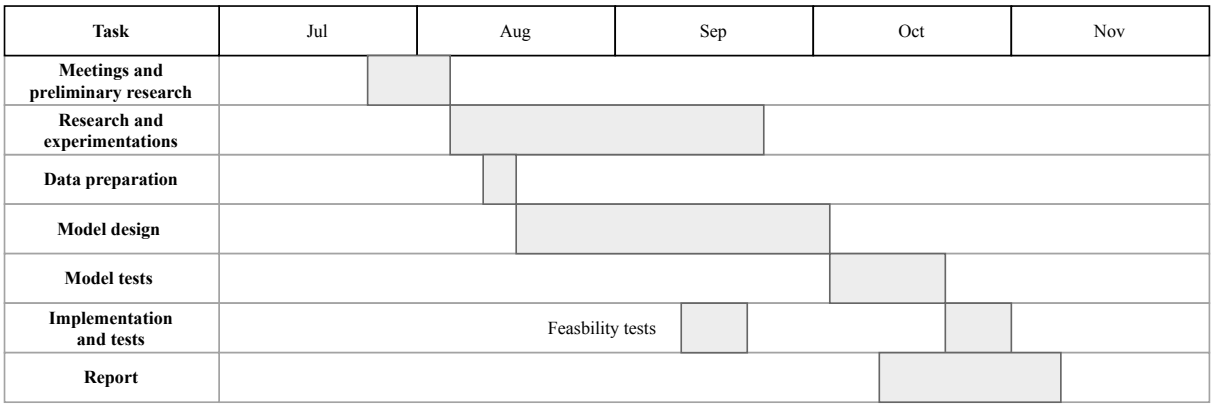


Figure B.1: An estimated timeline for the project tasks.

Appendix C

Outcomes Compliance

Table C.1: ECSA outcomes compliance.

Outcome	Description of outcome in report	Section(s)
ELO 1: Problem solving	The project identified and formulated the image classification task, analysed the implementation constraints and used knowledge from previous work along with creative solutions to solve the problem by developing and adjusting design techniques.	2.1, 2.2, 5.1
ELO 2: Application of scientific and engineering knowledge	The design and training of neural network classifiers requires engineering knowledge such as differential calculus, machine learning algorithms, object-oriented programming, computer science and system design.	3.1, 3.3
ELO 3: Engineering design	The entire project follows a strict system design process that incorporates both procedural design such as architecture testing using objective metrics and trade-off analysis and non-procedural design such as experimentation and assumptions.	3.2, 3.3, 4.2, 5.3
ELO 4: Investigations, experiments and data analysis	Extensive experimentation and empirical measurements were performed when designing, testing and implementing the final system for performance objectives.	3.3, 4.3, 5.3

Table C.2: ECSA outcomes compliance (cont.).

Outcome	Description of outcome in report	Section(s)
ELO 5: Engineering methods, skills and tools, including information technology	Software tools and packages such as Python and Google Colab for programming and supporting programs such as SSH and VNC for implementation were used extensively in development.	3.1, 5.1, 5.2
ELO 6: Professional and technical communication	The final technical report uses the appropriate academic style, format, structure, and language throughout, and was developed with LaTeX.	
ELO 8: Individual work	The project was performed entirely individually with no outside assistance and only supervision.	
ELO 9: Independent learning ability	Aside from the concept of supervised machine learning, all of the algorithms and technology used in the development of this system such as deep learning, gradient descent, optimization and implementations techniques were researched independently and newly discovered with online researches and books.	2.1, 1.2