Fall 12-13-2019

An Application of Deep Learning Models to Automate Food Waste Classification

Alejandro Zachary Espinoza
Portland State University

Recommended Citation
Espinoza, Alejandro Zachary, "An Application of Deep Learning Models to Automate Food Waste Classification" (2019). *Dissertations and Theses.* Paper 5365.
https://doi.org/10.15760/etd.7238

An Application of Deep Learning Models to Automate Food Waste Classification

by

Alejandro Zachary Espinoza

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
James McNames, Chair
John Lipor
Eric Wan

Portland State University
2019

Abstract

Food wastage is a problem that affects all demographics and regions of the world. Each year, approximately one-third of food produced for human consumption is thrown away. In an effort to track and reduce food waste in the commercial sector, some companies utilize third party devices which collect data to analyze individual contributions to the global problem. These devices track the type of food wasted (such as vegetables, fruit, boneless chicken, pasta) along with the weight. Some devices also allow the user to leave the food in a kitchen container while it is weighed, so the container weight must also be accounted for. Through the use of these devices, a company may better understand the amount and type of food being wasted, along with the cost and an estimated CO_2 impact. Unfortunately, data collection is often a manual process which requires a human to identify what is being thrown away at the time of the event. Manual data entry is prone to user error, which in turn leads to less accurate trending of waste and overall environmental impact.

In this thesis, convolutional neural networks (CNNs) are trained and evaluated on a novel food waste dataset to assist in automating food waste classification. To fully realize automation, both food waste and container classifiers are required. A major contribution of this work is to highlight the value of cleaning the test data, while also emphasizing the importance of leaving it untouched until the very end to avoid introducing bias into our estimate of the out of sample error. Another major contribution is to test the feasibility of learning on commercial food waste datasets. Following the recent successes of others in machine learning, best known practices are outlined and applied in dataset creation, network design, and performance evaluation. Some examples of performance metrics used include classification accuracy, micro-averaged precision and recall, F-Score, Grad-CAM visualizations, and the confusion matrix. The resulting models are high performing, but do have some limitations, such as the inability to effectively classify instances with mixed foods and fully distinguish container depth. Specifically, the food waste model achieved top 1 accuracy of 90.7% and top 5 accuracy of 98.8%, excluding mixed instances. The container model, which attempts to distinguish between different depths, achieves top 1 accuracy of 70.2% and top 5 accuracy of 94.6%, while the model which ignores depth achieves 84.6% and 98.6% accuracy, respectively.

In future work, more sophisticated networks can be trained to perform other computer vision tasks, such as semantic and instance segmentation. Through segmentation, images with mixed foods can be better classified. There may also be opportunities to reduce model size and inference time with deep compression. Other recommendations for improving the food waste and container classification models are also provided.

Acknowledgements

I would like to extend my deepest gratitude to the entire software engineering team I've had the pleasure of joining. This work would not have been possible without their contributions. In particular, I would like to thank my supervisor, Brian Boshes, for all of his coaching and insight on this project. I would also like to thank Bill Sarra for his advice, ideas, and feedback throughout all stages of this work.

I am grateful for the advice of my entire thesis committee. My advisor and committee chair, Dr. James McNames, has been very encouraging and supportive through this entire process. He has taught me a lot about what it takes to write a thesis which can convey technical work yet also communicate a good story, and also provided extensive feedback across all stages. The two remaining members, Dr. John Lipor and Dr. Eric Wan, have also been very supportive throughout this process. I would like to thank Dr. John Lipor for his coaching in the past, willingness to listen, and advice on this project. It was his direction that led me to study machine learning. I would also like to thank Dr. Eric Wan for all of his invaluable feedback, which has helped shaped this thesis into its present state.

Finally, I would like to thank all of my friends and family. Their support has been invaluable in completing my work. They were also instrumental in my practice for the defense of this thesis. My dog, Thor, was also very helpful in sitting by my side while I wrote this thesis and practiced for its defense.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 The Global Food Waste Problem

When was the last time you purchased food, only to throw it away? It might not be something you often consider, but the food waste problem is one of global proportions. Not only is it expensive to buy food to only have to throw it away later, but it is also damaging to the environment. The average household is not just to blame —it is also common to throw away food in the commercial sector, such as restaurants and cafeterias. Each year, approximately one-third of food produced in the world for human consumption is either lost or thrown away [3]. This amounts to roughly *2.9 trillion pounds* of food waste annually, with losses worth approximately $990 billion USD. $680 billion is estimated to come from industrialized nations and $310 billion from developing nations. With respect to the environment, food waste has a significant, unsustainable impact on land, climate change, water, and biodiversity [4]. Let us take a closer look at this.

1. **Land**

   The impact of wasted food on land is two fold, and begins with intensive farming, which drains the soil of its fertility. When soil quality degrades, farmers must introduce synthetic additives, which both cause pollution and eventually lead to loss of arable land. In 2007, almost 1.7 billion hectares of land was used to produce wasted food.

2. **Climate Change**

   Fossil fuels are heavily used in production of food, from planting to harvesting and shipping. Due to this, each wasted food item has a cascading impact on consumed resources and the resulting pollution. When food is thrown away, CO_2 is emitted from decomposition. While an indirect comparison, if food decomposition were a country, Global CO_2 emission from it alone would rank third, just behind the United States and China. The carbon footprint is estimated to be 3.3 gigatons of CO_2.

3. **Water**

   Agriculture is responsible for 70% of global freshwater consumed annually. When food is produced but not consumed, this also means that water used to produce it is wasted. The types of food which require the most water to produce are cereals, fruits, and meat. The water footprint is estimated to be 250 cubic kilometers of water.

4. **Biodiversity**

   The main effects of food waste on biodiversity can be observed in both the land and marine habitats. Land is often deforested in order to clear room to grow crops. When this occurs, the local population of flora and fauna is harmed in the process. 9.7 million hectares are deforested annually in order to grow food, which represents 75% of total deforestation. In marine habitats, excessive fishing disrupts the local food chain, which means less resources for the surrounding fish, mammals, birds, and amphibians. Approximately 70% of fish caught through trawling are wasted.

Clearly, wasting food is expensive and has serious environmental consequences. So, who is responsible for food waste? According to ReFED [5], consumer-facing businesses are responsible for 40% of wasted food, out of the 63 billion tons wasted in the US every year. Consumer-facing business include: supermarkets/grocery stores/distribution centers, restaurants, institutional food services, limited service restaurants, and government entities. Consumers are responsible for 43%, farms for 16%, and manufacturers for 2%.

1.1.1 What Others Are Doing About It

In an effort to reduce food waste from consumer-facing businesses, some companies have developed products to help businesses understand how much they are contributing to the problem on an individual level. In this thesis, these products are referred to as *food waste tracking systems.* These systems aim to aid the end user in trending top food waste categories (such as vegetables, fruits, beef, etc...), the value of the food waste, the reason why it is thrown out (this will also be referred to as a *loss reason*), and other related metrics over time. From the business' perspective, the practice of tracking food waste provides an opportunity to establish a baseline to better understand what kind of food is thrown out, how much of it is thrown out, and the loss reasons over time. Some loss reasons could be that the food is overcooked, sent back to the kitchen, expired, contaminated, and so on. This information is presented in the form of reports. By periodically reviewing these reports, management can experiment with different actions to drive down both overall food costs and the estimated environmental footprint with respect to the established baseline. Taking action to reduce food waste can also be good for employee morale, as staff in the kitchen are ultimately responsible for accurate recording of events. Only when events are properly documented will management have an accurate idea of true waste costs and loss reasons. Another potential upside is on the public relations front. Tracking food waste allows the business to show customers that they are striving to be environmentally conscious and aware of their individual contribution to the global food waste problem. This is especially important, now that many consumers are making more eco-conscious decisions when deciding where to shop or patronize [6]. As previously mentioned, the effects of food waste are cascading, and reduction of unnecessary consumption leads to numerous environmental benefits.

There are a few different companies which offer these devices on the market. Leanpath, a large supplier of food waste tracking systems, has deployments in over 32 countries around the world. In the past five years, with the help of these trackers, consumer-facing businesses in these countries have prevented over 40 million pounds of food waste from going to landfill [7]. One such device is called the Leanpath 360. This product consists of a desktop scale, equipped with an overhead mounted camera and an android tablet for capturing and displaying relevant information associated with the waste events. Each time food is thrown out, an operator must record the type of food being thrown away, the loss reason, and the container it is in. An image is then acquired of the waste event and the food is weighed. Based on this data, periodic reports are generated to provide the business an idea of how much food is being wasted, how much it costs, and the resulting CO_2 footprint. These reports include simple visualizations such as pie charts, displaying the top food waste categories, bar charts with top loss reasons, and more.

In addition to Leanpath, other companies such as Grace Organic and Winnow offer similar food waste tracking systems. Grace Organic offers a varying range of products, from a tabletop tracking system to a large composting system. The most comparable offering is called the Food Waste Tracker, which is similar to the Leanpath 360, but is not equipped with a camera to take images of the food. Winnow's offering is called the Winnow System. The Winnow System tracks food in a large waste bin resting on a floor scale. This product differs from the Leanpath 360 in that it does not track individual kitchen containers. As food is added to the bin, a new image is acquired, and the system infers what is new versus what has already been away. More details on what is known about this process is given in Section 2.2, but for now can be referred to as a form of *image classification*. The device incrementally weighs newly added food, so the waste can be thrown out successively. This system also offers trending of what food has been thrown away, the loss reason, and resulting CO_2 impact. The end-user can view this information in the form of a report, and interpret it in a way which allows reduction of food waste over time.

1.1.2 How Image Classification Can Help

Image classification typically follows a process as shown in Figure 1. First, an image is selected from a dataset or process as the system input. Some pre-processing may be performed, such as scaling, cropping, normalization, or augmentation. The pre-processed image is then fed into a feature extractor, which teases out discriminating features. Some methods of feature extraction include: Fisher vector, bag of visual words, principal components analysis, autoencoders [8], random forests [9], and convolutional neural networks [10]. These extracted features are then fed into a trained classifier —typically a supervised learning algorithm— such as (but not limited to) a support vector machine (SVM), artificial neural network (ANN), logistic regression, discriminant analysis, or decision tree. The classifier determines which class the image belongs to. In the case of food classification, the class may be an integer which maps to one of the food waste categories. Note that some more simple images, which contain less unrelated information or background noise (such as those from the MNIST [11] database), may skip the feature extraction step and still perform well.

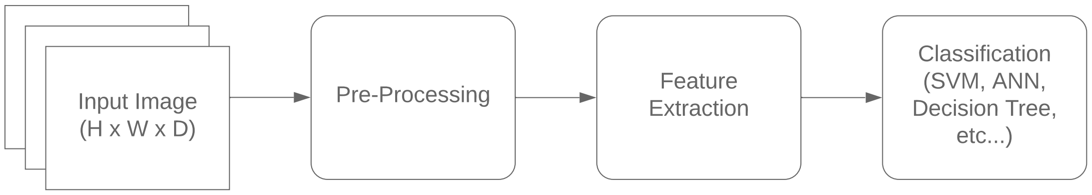


Figure 1: Example Image Classification Process. Given an input image of dimensions H $\times$ W $\times$ D. Let H represent the height (in pixels), W represent the width (in pixels) and D represent the number of channels (eg: red, green, blue).

When computers are able to understand the world through visual percepts, it is referred to as *computer vision*. Specifically, when computer vision is applied in industrial, machine, or device applications, it is referred to as *machine vision*. One example would be a device performing image classification on food as it is thrown away. Convolutional neural networks (CNNs) have been revolutionary for computer vision applications. They yield state-of-the-art results on challenging datasets, which often consist of many diverse and nuanced classes. One reason for this is because they combine the benefits of kernel convolution, which is able to perform varying types of feature detection, over many deep layers [8]. There can be varying numbers of kernels (filters) at each layer of the network. As the network is trained on a specific image dataset, the filter weights are learned. The kernels learned in this process are often much smaller than the image, which means that they learn sparse representations of features, such as edges, corners, lines, and even textures. The same kernel is applied across the entire input feature map, which is in contrast to a fully connected neural network, which has a single weight parameter for each input data point. This feature of CNNs is referred to as *parameter sharing*. Due to parameter sharing, convolutional layers require fewer parameters than fully connected neural networks, and thus consume less memory. The extracted features can then be fed into a classifier at the end. The classifier will typically be a feed-forward neural network, but could also be another type of algorithm as mentioned previously. Often, the classifier is simply the final layer of the CNN, so they are one network. For food image classification, there are many publicly available datasets online, such as: Food-101 [9], Food-256 [12], Food-11, and Food-5k [13]. There are none for the specific case of food waste classification. Previous work has applied other algorithms to some of these datasets for classification [9] [14], but have ultimately been outperformed by architectures utilizing CNNs. It is for these reasons that a good first choice for a custom food waste image classifier would be a type of CNN.

As mentioned in Section 1.1.1, tracking and taking accountability for food waste is already a great first step towards minimizing our problem. However, some current technology (such as the Leanpath 360) requires the end user to manually enter the type of food being thrown away and the type of container it is in. On average, searching through a user interface to find the proper item category can take about 3 to 5 seconds, longer for users with a more diverse selection of dishes. As kitchens tend to be fast-paced and demanding environments, every second counts, and this manual classification task detracts from other duties. In a situation where multiple waste events need to be recorded at once, these small time slices can also accumulate and become burdensome. Furthermore, manual data entry in itself is prone to user error, which can lead to inaccurate accounting of actual waste costs. This is where image classification can help. Consider a scenario in which the product to be thrown out is placed on the device, a picture is taken, and shortly after the device recognizes what type of waste item it is along with the type of container it is in. The weight is measured and the additional container weight subtracted. Now, all that is left is for the user to enter the remaining data such as the reason for why it is being thrown out. Using state-of-the-art neural network architecture, not only can this be made possible, but it can also surpass human-level accuracy, leading to improved cost and environmental footprint tracking.

1.1.3 Barriers to Success

Lack of available data. Two of the main challenges in machine learning are finding enough suitable data to train a classifier, and being able to discover a pattern in the data which can be approximated by a model. In regards to food waste classification, the datasets mentioned in Section 1.1.2 can be good for evaluating or benchmarking candidate model architecture, but may not directly transfer well for the specific task. One reason for this may be that food waste is often composed of more general categories and different representations of food. For example, some waste categories may be less granular, such as fruits or vegetables. Some of the images of fruits or vegetables may be scraps, like the tops of carrots or the stems of grapes. The feature detector must be trained to activate when presented with these patterns. In addition, the listed datasets are all subject to copyright limitations, which prohibit commercial and business use, outside of academic research. As the work referenced in this thesis is being incorporated in a device developed by a for-profit supplier of food waste tracking devices, datasets with such restrictions cannot be used.

One solution to the above issue is to use data without these restrictions. In this work, a dataset collected the same company will be used, which consists of thousands of images of food waste events from various customer sites, dating from 2015 to 2018. The goal of this company is to utilize the classifier described in this thesis for integration into its food waste tracking system. Previous attempts have been made to use the data to train a computer vision model, but have not been successful due to poor accuracy in metadata and lack of domain expertise. The poor accuracy in metadata can be traced back to human error at the time of recording a waste event, and a lack of clear direction for which category certain items should be assigned to. These inconsistencies, also called *class noise*, can lead to data which is extremely difficult to accurately train and score a high performing classifier on. By integrating a robust image classifier into the food waste tracking system, the customer may benefit from more accurate waste and cost accounting, in addition to reduced user interface time as mentioned in Section 1.1.2. The company providing the food waste tracking system also benefits by the addition of such features to its product, as well as staying on the cutting edge of technology, since companies such as Winnow already have a type of image classifier deployed.

Class noise. The provided dataset is also unbalanced and contains pictures which have sometimes been taken by mistake, occluded by hands or food container lids, or are very blurry. Unbalance in a dataset refers to having different proportions of class sizes, rather than uniform class sizes. For example, in the food waste dataset, there are many more examples of vegetables than fruit. The frequency of these issues, including class noise, vary depending on the customer site selected. A detailed estimate will be provided later on in this document. Imperfect examples are both common and somewhat expected in real-world datasets. Typically, neural networks are fairly robust to this type noise in data, and in fact, noise is often introduced into networks as a way to mitigate over-fitting [15]. However, class noise does reduce overall test accuracy. Techniques to address such issues will be discussed in the experimental design.

1.2 Contributions

A primary contribution of this thesis is to show that, with proper application of best known practices, a well-trained deep learning model can show promise in classifying food waste items. Specifically, to satisfy the performance requirements of the company which is benefiting from the work described in this thesis. By introducing more automation into the process of tracking waste events, our hope is that accuracy will be increased at each customer site, leading to both better cost accounting and a reduction in food wasted. Another goal is to join the current successes in food image classification, demonstrating that these models can be applied to more niche markets, provided the proper training data.

A secondary contribution is to provide a reference for others who wish to perform similar tasks, by providing a comprehensive guide outlining model architecture, along with the tools and techniques required to develop such a classifier from training to production. The individual techniques are all open source and available to the public. A goal of this thesis is to simply knit these techniques together in a cohesive way, which is easy to follow and apply with some programming knowledge and a custom dataset.

2 Related Work

Much of the current work described in this chapter is related in terms of food image classification, rather than the task of food waste classification. This is because food image classification from the point of view of reducing food waste is a relatively unexplored application, with one exception which will be discussed. Often, the motivation behind researching food image classification is to integrate into a system which aids the end user in reducing caloric intake, promote a healthy diet, mitigate obesity, and assist in other health-related ways. Many of the same principles and methods which have been researched in food image classification, in general, are still very relevant when applied to the subset focused on in this thesis.

2.1 Other Feature Extraction Methods

As previously stated, CNNs have provided state-of-the-art classification accuracy on today's large image datasets. Prior to the availability of these datasets, and before network architecture was as sophisticated, image classification was often performed using techniques such as Fisher vector (FV) [16] or bag-of-visual words (BoW) representations [17]. Both of these are considered image patch encoding techniques, which aim to capture local image information using *descriptors* as inputs. These descriptors can be extracted from each image patch using a feature detection algorithm like scale invariant feature transform (SIFT)[18] or histogram of oriented gradients (HoG)[19]. In the case of BoW representations, an offline "codebook" is learned by performing k-means clustering with representative descriptors from each class. A supervised learning algorithm is then trained on a set of training images, by extracting descriptors from each example and compiling histograms of the corresponding codebook

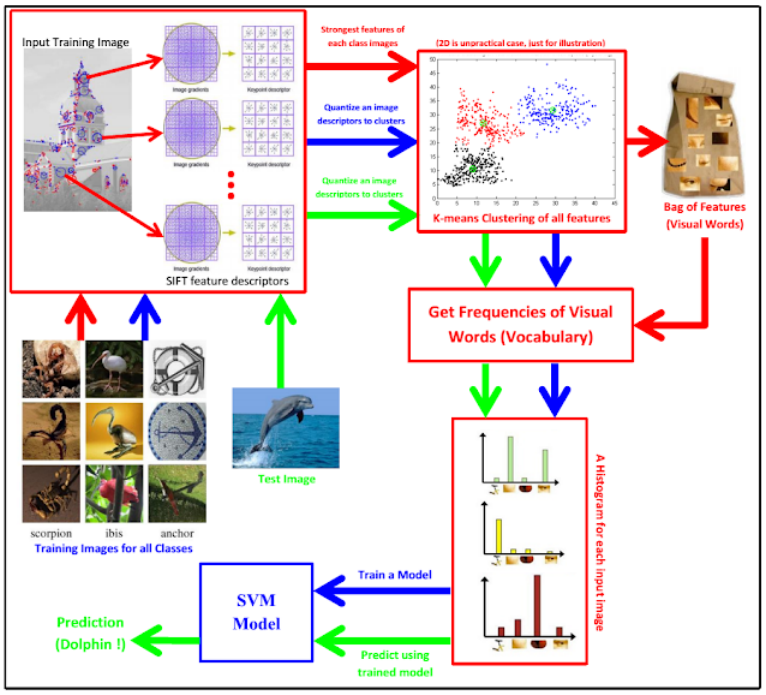

Input Training Image
Image gradients
Keypoint descriptor
SIFT feature descriptors
Strongest features of each class images
Quantize an image descriptors to clusters
Quantize an image descriptors to clusters
(2D is unpractical case, just for illustration)
K-means Clustering of all features
Bag of Features (Visual Words)
Get Frequencies of Visual Words (Vocabulary)
A Histogram for each input image
scorpion
ibis
anchor
Training Images for all Classes
Test Image
Train a Model
SVM Model
Predict using trained model
Prediction (Dolphin !)

Figure 2: Bag of Words Classification, used with permission from [1]

entries. These histograms are the inputs to the learning algorithm. Once trained, the same process is repeated on a test set, and performance can be evaluated. For reference, an example BoW classification system is depicted in Figure 2.

The Fisher vector encoding differs from BoW in that it builds a codebook by training a Gaussian mixture model (GMM) on image descriptors. The parameters of the GMM are learned using an expectation maximization (EM) algorithm to optimize maximum likelihood criterion. While a complete introduction to the underlying theory and implementation of the FV encoding can be found in the work by Sanchez, et al [16], a brief overview is provided here. Essentially, a sample of T D-dimensional image descriptors $X = \{x_t, t = 1 \ldots T\}$ is extracted from an image. The Fisher vector is then defined as:

$$\mathscr{G}_\lambda^X = \sum_{t=1}^{T} L_\lambda \nabla_\lambda \log u_\lambda(x_t) \tag{1}$$

under which, the operation

$$x_t \longrightarrow \phi_{FK}(x_t) = L_\lambda \nabla_\lambda \log u_\lambda(x_t) \tag{2}$$

represents an embedding for each descriptor in a higher-dimensional space, in which linear classification can be more easily performed. Equation 1 represents the sum of normalized gradient statistics $L_\lambda \nabla_\lambda \log u_\lambda(x_t)$, for each element t. u_λ represents the learned GMM model with K components $\lambda = \{w_k, \mu_k, \sigma_k, k = 1, \dots, K\}$, trained on a sample of descriptors. It is defined as:

$$u_\lambda(x) = \sum_{k=1}^{K} w_k u_k(x) \tag{3}$$

where u_k represents the Gaussian k, Σ_k the corresponding covariance matrix, and mean vector μ_k:

$$\mu_k(x) = \frac{1}{(2\pi)^{(D/2)} \left|\Sigma_k\right|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu_k)' \Sigma_k^{-1} (x - \mu_k)\right\} \tag{4}$$

with weight parameters, w_k:

$$w_k = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp(\alpha_j)} \tag{5}$$

∇_λ can be calculated in terms of each of the 3 gradient components which correspond to the GMM parameter λ. These are shown in Equations 6, 7, and 8. Equation 9 is the soft assignment of descriptor x_t to Gaussian k.

$$\nabla_{\alpha_k} \log \mu_\lambda(x_t) = \gamma_t(k) - w_k \tag{6}$$

$$\nabla_{\mu_k} \log \mu_\lambda(x_t) = \gamma_t(k) \left(\frac{x_t - \mu_k}{\sigma_k^2} \right) \tag{7}$$

$$\nabla_{\sigma_k} \log \mu_\lambda(x_t) = \gamma_t(k) \left(\frac{(x_t - \mu_k)^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right) \tag{8}$$

$$\gamma_t(k) = \frac{w_k u_k(x_t)}{\sum_{j=1}^{K} w_j u_j x(t)} \tag{9}$$

L_λ represents the square root of the inverse of the Fisher information matrix (FIM). Instead of directly computing this, the diagonal FIM is approximated by a coordinate-wise normalization of the gradient vectors. The gradient vectors are then:

$$\mathscr{G}_{\alpha_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} (\gamma_t(k) - w_k) \tag{10}$$

$$\mathscr{G}_{\mu_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \left(\frac{x_t - \mu_k}{\sigma_k} \right) \tag{11}$$

$$\mathscr{G}_{\sigma_k}^X = \frac{1}{\sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) \frac{1}{\sqrt{2}} \left[\frac{(x_t - \mu_k)^2}{\sigma_k} - 1 \right] \tag{12}$$

The corresponding Fisher vector is finally obtained by concatenating each gradient vector and normalized to remove dependence on sample size:

$$\mathscr{G}_\lambda^X = \frac{1}{T} \left(\mathscr{G}_{\alpha_1}^X, \ldots, \mathscr{G}_{\alpha_k}^X, \mathscr{G}_{\mu_1}^{X'}, \ldots, \mathscr{G}_{\mu_k}^{X'}, \mathscr{G}_{\sigma_1}^{X'}, \ldots, \mathscr{G}_{\sigma_k}^{X'} \right)' \in \mathbb{R}^{(2D+1)K} \tag{13}$$

Which is then *power normalized* and l_2 *normalized*:

$$[\mathscr{G}_\lambda^X]_i \leftarrow \frac{\text{sign}\left([\mathscr{G}_\lambda^X]_i\right)}{\sqrt{[\mathscr{G}_\lambda^X]_i}} \text{ for } i = 1, \ldots, K(2D+1) \tag{14}$$

$$\mathscr{G}_\lambda^X = \frac{\mathscr{G}_\lambda^X}{\sqrt{\mathscr{G}_\lambda^{X\prime} \mathscr{G}_\lambda^X}} \tag{15}$$

The l_2 normalization is meant to account for the fact that different images contain different amounts of background information and works to discard this information. One justification for the power normalization is to account for the fact that the FV becomes sparser as the number of GMM components increase, which has a negative effect on the dot product. Introducing the power normalization helps to make the FV less sparse. The resulting Fisher vector can then be fed into a supervised learning algorithm, like an SVM, for classification. Further details on the algorithm and implementation can be found in the aforementioned work by Sanchez, et al.

Advancements in food image classification have mostly been motivated by the desire to improve human health through more automated food logging and calorie counting. One such model, FoodCam-256 [14], is a food image classification app designed to run on mobile android architecture. It was developed with the goal of allowing a user to take a picture of a dish they are about to consume and understand how many calories it contains. FoodCam-256 utilizes Fisher vector encoded features to classify food images. The classifier is trained on the UEC-Food256 dataset, which contains 256 different classes and was also built by the authors for this work. Inputs to the system are modified HoG descriptors, extracted from each training image. The encoded FVs are then fed into one-versus-all linear classifiers, which are trained on the AROW algorithm. This approach yielded 50.1% top-1 accuracy and 74.4% top-5 accuracy.

In later work by Bossard, et al. [9], a new method for mining discriminative components was introduced, along with a new benchmark multiclass food dataset. The new method, called *random forests discriminant components* (RFDC), utilizes the random forests algorithm to identify distinguishing regions (referred to as *component mining*) in a given image. These mined components can then be fed into a supervised learning algorithm for classification. The component mining technique is presented as an alternative to other descriptor-based methods, such as Fisher vector or BoW. Bossard, et al. benchmark this method against some of the previously mentioned algorithms on the Food-101 dataset, along with a CNN based on the Alexnet architecture. The Food-101 dataset, released in this work, was a novel contribution as it was the first food image dataset considered large enough to train deep learning models. It is still widely used today to benchmark new algorithms. Food-101 contains 101 different classes of food images, with 750 training and 250 test images for each class. When building this dataset, the authors purposefully left the training images as collected, but manually some cleansing on the test images. While it was shown that RFDC mining provide a good alternative to descriptor based methods, it was still outperformed by the CNN approach by almost 6% in terms of classification accuracy.

2.2 CNN-Based Classifiers

With the release of the Food-101 dataset, others have begun to apply CNNs to the food recognition problem. Similar to Bossard, et al., Yanai and Kawano [20] applied two AlexNet-based models to both Food-101 and the UEC food datasets. They utilized transfer learning —both were pretrained on Imagenet, but fine tuned on each respective food dataset. These models outperformed previous results using both Fisher vector encoding and CNNs. This work was followed by Fairnella, et al. [21], who not only introduced another dataset, *UNICT-FD1200*, but another approach to classification called the *bag of textons*. The bag of textons approach is more focused towards performing classification on smaller datasets, an area where applying CNNs has traditionally not performed as well. Accordingly, the newly introduced UNICT-FD1200 is presented as a smaller dataset, consisting of 8 classes with a total of 4,754 images. Fairnella, et al.'s results showed that the bag of textons approach outperformed a fine-tuned GoogLeNet (CNN)-based model on this dataset by nearly 30%. Since then, however, it has been shown that techniques such as data augmentation can significantly boost generalization of a CNN in image classification [22]. It is unclear if this was attempted in Fairnella, et al.'s work. The algorithm was also not benchmarked against a larger dataset such as Food-101.

Building upon the successes of [23], [12], and [9], Singla, et al. [13] released two new datasets for food image classification: Food-11 and Food-5k. Both were built by combining images from Food-101, UEC-Food100, and UEC-Food256, but have different purposes. Food-11 is intended for use in multinomial classification over 11 different classes, whereas Food-5k was designed for only identifying the binary case of food/not food. Two classifiers, each utilizing the GoogLeNet architecture, were trained on the respective datasets. The authors were able to exceed 99% classification accuracy on food/not food, and 83.5% on the multiclass problem.

As CNN architecture has continued to improve, so has performance over the available datasets. One network design, called WISeR, [24], proposed utilizing *slice convolutions* in order to take advantage of the vertical layers in certain food dishes. A slice convolution is similar to a square convolution, but instead shares the width of the input image. It is noted that typical squared convolutions may capture features unique to vertical structure over many deep layers, but a slice convolutional layer may allow these to be detected in a single layer. WISeR incorporates these slice convolutions in parallel with a wide variant of ResNet and concatenates the detected features from both branches at the end. The concatenated features are then fed into fully connected layers for classification. This architecture was benchmarked against the UEC-Food100, UECFood256, and Food-101 datasets, and outperformed the prior state-of-the-art.

One commonly held belief has been that better accuracy can be achieved by simply scaling a convolutional neural network, but in general this practice has not been well understood. This is investigated in very recent work by Tan and Le [25]. Improved performance was achieved on existing network architectures by optimizing depth (number of layers), width (number of channels), and input image resolution. This is achieved by uniformly scaling each of these dimensions by a *compound coefficient*. In addition to optimizing existing architectures, state-of-the-art performance has been achieved on many benchmark datasets by utilizing neural architecture search to create a new family of neural networks called EfficientNet. The EfficientNet family of models utilizes several stages called *mobile inverted bottlenecks* as the building blocks. The depth, width, and input resolution vary depending on the variant of EfficientNet selected, which begin with the base model (EfficientNet-B0), and end with the largest model (EfficientNet-B7). When pretrained on ImageNet and fine-tuned on Food-101, EfficientNet-B7 yields state-of-the-art top-1 accuracy of 93%, which is equal to the performance of Google's Gpipe model, with 8.7 times fewer model parameters. This is an important development—as the memory requirements for model inference become higher as model parameters increase, which means that better performance can be achieved with more modest hardware.

Another recent approach by McAllister, et al. utilizes CNNs pretrained on ImageNet as deep feature extractors [26]. These networks are only used to extract features from benchmark food image datasets, instead of the more common approach which either fine-tunes the network on a new dataset or trains from scratch. The extracted features from each input image are compiled into a separate "deep feature dataset" with the same ground truth labels. These newly extracted features are then used to train supervised learning algorithms. The pretrained networks used are GoogLeNet and ResNet-152, and the benchmark datasets used are Food-5K, UNICT-FD889-Caltech, Food-11, RawFooT-DB, and Food-101. Food-5K and the UNICT-FD889-Caltech datasets are used for only food/not food classification, and the others used to perform multinomial classification. Performance is compared between the popular naive Bayes, SVM (RBF and polynomial kernels), ANN, and random forest algorithms. The authors found that classification with features extracted from ResNet-152 architecture consistently outperformed those extracted from a GoogLeNet model, in terms of classification accuracy. The classification accuracy was the highest when utilizing an ANN compared to the other listed algorithms. It is also noted that CNN features consistently yield better classification accuracy than prior, traditional feature extraction methods.

It is also relevant to mention that the Winnow system (see: Section 1.1.1) currently employs a form of machine vision, but only the most basic details are available to the public. It is known that the local model is a type of neural network, running on an NVIDIA Jetson TX2 embedded system [27]. Newly acquired images are classified by this model and asynchronously uploaded to Amazon Web Services to train a new version. The training utilizes NVIDIA V100 GPUs. This implementation claims to identify the top-5 matches for a waste event upon placing food in the bin, but the accuracy and precision are not advertised. The source suggests model may also require between 200-1000 examples before it is able to recognize a new class. The guidelines for what is considered recognition are not given in terms of accuracy, precision, or any other metric.

2.3 Tying It Together

When treating the food waste image recognition problem as an image classification problem, the best practice for training a model on a new dataset would be to follow the recent successes of others. One such example of the introduction of a new and novel dataset is found in the Food-101 dataset [9]. A normal practice when creating datasets for learning algorithms is to split the collected observations into separate training and testing sets. In Food-101, the training data is actually left somewhat noisy, but the test data is cleaned. As previously mentioned, noisy training data can be beneficial to the learning process as it provides some regularization. The test data, however, should be free of mislabeled examples in order to accurately gauge the performance of the model. In this work, the same practices for creating a dataset are followed.

It is clear that the research behind Food-101 and CNN optimization helped lay the groundwork for a potential solution to automated food waste classification. While the Food-101 dataset itself would be a good stepping stone for food waste classification, for previously mentioned reasons it cannot be used to train a food waste classifier or augment the dataset in this thesis. However, the performance of current state-of-the-art models on this dataset can provide a reasonable indication of what performance may look like on the datasets with similar features and difficulty, such as our custom dataset. By this metric, the argument could be made that the EfficientNet family of models provide a good starting point for this application. In particular, The EfficientNet-B0 architecture provides reasonable performance with a very small footprint, which is good for mobile platforms. The performance of other models will also be explored. This will be investigated further in the following sections.

3 Model Design

In this chapter, some central concepts will be presented to aid the reader in better understanding the design choices and model architectures presented in this thesis. Much of this content has been adapted from Learning From Data [28], An Introduction to Statistical Learning [29], Deep Learning [8], and other cited sources. After reviewing these concepts, the model architecture and datasets will be introduced.

3.1 Learning from Examples

Before applying a learning algorithm to a problem, it is important to understand the conditions under which they are most successful. Machine learning is best applied on a problem for which [28]:

- Data has been collected,
- a pattern exists in the data,
- and the pattern cannot be explicitly defined by a deterministic relationship.

When these requirements are satisfied, a learning algorithm has a better chance of success. Our goal in this case is to approximate some unknown mapping $f : \mathcal{X} \to \mathcal{Y}$, where f represents the true (or target) function, $\mathcal{X}$ represents the collected observations, and $\mathcal{Y}$ represents the target(s). Observations can come in many different formats, ranging from real-valued time series, digitized images, to bio-metric data and more. Similarly, examples of targets include: sequences, binary classes (cat/not cat), categorical (what kind of cat is this?), and real values (like stock prices). When the algorithm predicts some kind of label from a finite set of categories, the task is called *classification*. When the algorithm predicts some real valued target, the task is called *regression*.

In general, there are four types of learning algorithms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. In a *supervised learning* problem, a set of labeled examples is used to train an algorithm. The dataset is partitioned into training and test sets, the usual rule of thumb is to set aside $30 - 10\%$ of which for testing. The goal is to minimize an *objective function*, which in this case is referred to as a *loss function* or *error function*, which is a method used to measure error in prediction. A commonly used loss function is the *Mean Squared Error*, or $L2$ Loss, can be defined for inputs $\mathbf{X} \in \mathbb{R}^{N\times D}$, labels $\mathbf{y} \in \mathbb{R}^{N}$, and model weights $\mathbf{w} \in \mathbb{R}^{D}$ as:

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{f}(\mathbf{x}_i) - y_i\right)^2 \tag{16}$$

The loss function evaluated on the training data yields the *in-sample error*, E_{in}. Typically, we want to minimize this. In order to obtain an estimate of how a model will perform on unseen data, the loss function will need to be evaluated on the test set to obtain E_{out}. It is important to get a good estimate of E_{out}, as this is the best indication of how the model will *generalize*, a term which refers to its accuracy on unseen data drawn from the same distribution. The in-sample error is inherently unreliable, as the model may learn patterns in the data which lead to memorizing the labels, but are unrelated to the true pattern. A larger test partition leads to a better estimate for for E_{out}, but less data to learn on. Similarly, a larger training partition will lead to better learning, but a less accurate estimation of E_{out}. It is important to strike a balance between the two. Note that any changes to the model after assessing E_{out} will introduce bias into the estimate.

An *unsupervised learning* algorithm does not utilize labels to approximate f. Instead, it seeks to exploit patterns solely in the input data. Clustering algorithms and dimensionality reduction techniques are some of the more popular unsupervised learning methods. A clustering algorithm has a similar goal to a supervised learning problem, and simply assigns an input to a group learned by analyzing input features. Dimensionality reduction includes techniques such as principal components analysis (PCA). The goal of dimensionality reduction is to represent data from a higher dimension in a lower dimension, with minimal loss of information content.

Semi-supervised learning utilizes both labeled and unlabeled data in learning tasks [30]. One example of this could be applying a model already trained using supervised learning methods to predict labels for unseen data. The previously unseen data and predicted labels, along with the past training data, can then be fed back into the model as a new training set. The idea is that a model trained using semi-supervised methods may be able to attain high accuracy with only a small amount of human-labeled data.

A goal of some learning algorithms is to optimize the behavior of an artificially intelligent *agent* in a given task environment. An agent can refer to any entity which perceives its environment through sensors and is able to take action upon it through some type of actuator. *Reinforcement learning* algorithms aim to help the agent discover the best set of actions, given a state, by rewarding actions deemed as positive and reprimanding actions deemed as negative. Through simulated episodes with many iterations, the goal is to eventually learn a *policy* which defines how the agent may map a sequence of states to actions which lead to the best rewards.

3.2 The Bias-Variance Decomposition

The scope of this thesis is mainly concerned with supervised learning, in which we wish to minimize an error measure like Equation 16. The out of sample error, E_{out}, is one of the ways we can quantify a model's performance. E_{out} can be examined in terms of *bias* and *variance*. This helps us see the relationship between model flexibility and performance in generalization [28]. Recall that the variance of a random variable X is defined as:

$$\begin{aligned} \mathrm{Var}[X] &= \mathrm{E}[(X - \mathrm{E}[X])^2] \\ &= \mathrm{E}[X^2] - \mathrm{E}[X]^2 \end{aligned} \tag{17}$$

and the bias of a model $\hat{f}(x)$ is defined as:

$$\mathrm{Bias}[\hat{f}(x)] = \mathrm{E}[\hat{f}(x)] - f(x) \tag{18}$$

A model has high bias if it does not fit the data well. On the other extreme, a model can have high variance when it fits the training data too much and does not generalize well.

Now, let $y = \sin(x)$ be a target function we are trying to approximate, with $y, x \in \mathbb{R}$

[28]. We can sample from y as many times as desired, but only have a small amount of data points d at any given time. Let a single model fit to a set of randomly sampled data points, $\mathbf{x} \in \mathbb{R}^d$, be represented as $g^D(\mathbf{x})$. The D simply denotes that a model is fit to a specific dataset. Assuming we have access to all possible datasets, the average hypothesis will be denoted as:

$$\begin{aligned} \bar{g}(\mathbf{x}) &= \frac{1}{K}\sum_{i=1}^{K}(g^{D_i}(\mathbf{x})) \\ &= \mathrm{E}_D[g^D(\mathbf{x})] \end{aligned} \tag{19}$$

for K datasets, D_i, of d samples. $\bar{g}(\mathbf{x})$ can also be viewed as the best possible hypothesis which can be arrived at. With this defined, the variance of a prediction can be represented by:

$$\begin{aligned} \mathrm{Var}[\mathbf{x}] &= \mathrm{E}_D[(g^D(\mathbf{x}) - \bar{g}(\mathbf{x}))^2] \\ &= \mathrm{E}_D[g^D(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2 \end{aligned} \tag{20}$$

Which is simply the expected value of how far a prediction varies from the average hypothesis. E_{out} can be defined based on E_{out}^D:

$$E_{out}^D(\mathbf{x}) = (g^D(\mathbf{x}) - f(\mathbf{x}))^2 \tag{21}$$

$$E_{out}(\mathbf{x}) = \mathrm{E}_D[E_{out}^D(\mathbf{x})] \tag{22}$$

In this context, we can decompose E_{out} as follows:

$$\begin{aligned}
E_{out}(\mathbf{x}) &= \mathrm{E}_D[(g^D(\mathbf{x}) - f(\mathbf{x}))^2] \\
&= \mathrm{E}_D[g^D(\mathbf{x})^2 - 2g^D(\mathbf{x})f(\mathbf{x}) + f(\mathbf{x})^2] \\
&= \mathrm{E}_D[g^D(\mathbf{x})^2] - 2\bar{g}(\mathbf{x})f(\mathbf{x}) + f(\mathbf{x})^2 \\
&= \mathrm{E}_D[g^D(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2 + \bar{g}(\mathbf{x})^2 - 2\bar{g}(\mathbf{x})f(\mathbf{x}) + f(\mathbf{x})^2 \\
&= \mathrm{E}_D[g^D(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2 + (\bar{g}(\mathbf{x}) - f(\mathbf{x}))^2 \\
&= \mathrm{Var}[\mathbf{x}] + \mathrm{Bias}[\mathbf{x}]
\end{aligned} \tag{23}$$

Which means that the expected out of sample error can be broken down into the bias and variance of the hypothesis with respect to the data points $\mathbf{x}$. The Var[$\mathbf{x}$] can tell us how far away a specific hypothesis is from the best possible hypothesis $g(\mathbf{x})$ is capable of producing, $\bar{g}(\mathbf{x})$. The Bias[$\mathbf{x}$] tells us how far off $\bar{g}(\mathbf{x})$ is from the true function. This decomposition is important because it facilitates troubleshooting model performance. If using this information to change any aspects of the model, it is important that a separate validation set is used (and not a test set). The moment that test data is used to assess performance and make model changes, the estimate becomes biased. The decomposition is informative in the following ways. If a model has high bias, it may not be complex enough to fit the data. If the model has high variance, it may be too complex. The basic idea is that the more complex a model is, the lower the bias, but the higher the variance. More complex models have more degrees of freedom and are increasingly likely to learn patterns in the data unrelated to the true function, in which case E_{in} may be low but E_{out} will be high. This is called *overfitting*. Similarly, the simpler a model is, the higher the bias, but the lower the variance. High bias models are less likely to overfit, but also less likely to generalize well. A simple example helps explain this.

Let's try approximating $y = \sin(x)$ with two different models, sampling $d = 5$ data points at any given time. Our first model will be chosen from $H_0(\mathbf{x}) = b$, the set of all possible hypothesis for $b \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$. Our second model will be chosen from the set of hypotheses $H_1(\mathbf{x}) = a\mathbf{x} + b$, with $a, b \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^d$. The optimal solution for H_0 would simply be the average of d points. H_1 can be solved as a least squares problem. In both cases, the best hypothesis can be computed by sampling over many iterations. The graphs in Figure 3 show the solutions to this problem. The best hypotheses, H_0^* and H_1^*, are drawn in red, with the lighter shading around each line representing the variance of each hypothesis set. E_{out} for each solution is also shown.

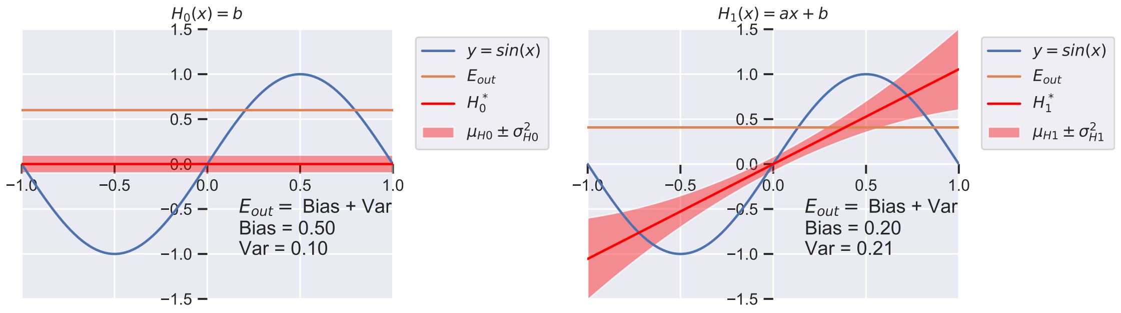


Figure 3: Bias and Variance Tradeoff Example

It is clear that the simpler hypothesis, H_0^*, has a lower variance, but much higher bias. H_1^* has a much lower bias but higher variance. For this problem, H_1^* is the clear winner with $E_{out} = 0.41$.

When evaluating model performance, it is important to consider E_{out} relative to E_{in}. An obvious sign of overfitting is when $E_{in} \ll E_{out}$. As previously stated, this points to a model memorizing patterns which allow it to predict the target without actually learning the target function. There are various techniques which can be employed to fight overfitting, the implementation of which is called *regularization*. Regularization reduces the overall variance, but increases the bias of the model. A common regularization method is to introduce an additional penalty for complexity into the loss function, such as the $L1$ or $L2$ norm of the weight matrix. Adding $L1$ regularization to the MSE Loss from Equation 16:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{f}(x_i) - y_i\right)^2 + \frac{\lambda}{N}\sum_{i=1}^{D}\|w_i\| \tag{24}$$

for each weight w_i in $\hat{f}(x)$. Similarly, adding $L2$ regularization to Equation 16:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{f}(x_i) - y_i\right)^2 + \frac{\lambda}{2N}\sum_{i=1}^{D}\|w_i\|_2^2 \tag{25}$$

Specific techniques with respect to neural networks will be discussed in more detail in the following subsections.

3.3 Convolution

One of the most frequently used operations in computer vision and image processing is the *convolution* operator. A convolution is simply the weighted sum between two functions f and g [31]. For the discrete case, a one-dimensional convolution can be defined as:

$$h(x) = (f * g)(x) = \sum_{u=-\infty}^{+\infty} f(u)g(x-u) \tag{26}$$

and a two-dimensional convolution:

$$h(x,y) = (f * g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)g(x-u, y-v) \tag{27}$$

When linear filtering is performed on an image, it is accomplished by convolving the filter with the image. For example, in certain image processing applications, it may be useful to reduce the information content of an image by only extracting the edges. This is also called *edge detection*, and can be accomplished through a variety of means, but for this example we will use the *Sobel filter* [32]. The Sobel filter consists of separate vertical and horizontal kernels, K_x and K_y:

$$K_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad K_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{28}$$

Notice that $K_x = K_y'$. Each kernel is used to approximate the gradient in the respective x or y direction. The magnitude of the gradient determines an edge. Edges are detected with the Sobel filter by first convolving the image, I, with each kernel and then evaluating the Euclidean norm:

$$G = \sqrt{(K_x * I)^2 + (K_y * I)^2} \tag{29}$$

Although, a common approximation is to simply sum the absolute values of each gradient approximation:

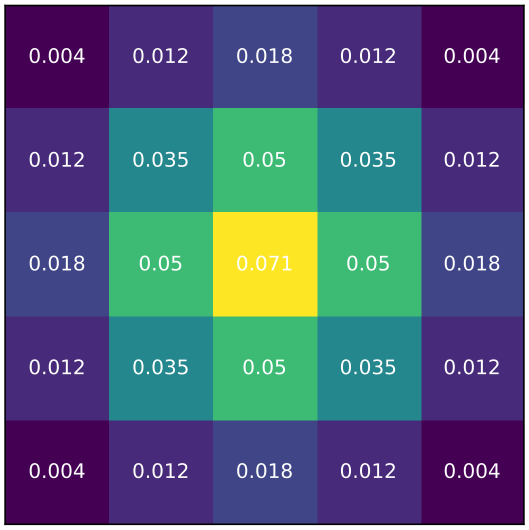
$$G = |K_x * I| + |K_y * I| \tag{30}$$

Typically, this is done in gray-scale, but can be extended to RGB by performing the same process on each channel independently.

Consider an image, I, corrupted with random noise. We wish to detect the edges in this image. A usual first step is to apply a basic smoothing operation, such as the *Gaussian blur*, to help with de-noising and better distinguish the edges. The Gaussian blur is a low-pass filter which averages image pixels by its neighbors, with the closest pixels weighted the highest, gradually decreasing with distance. The filter kernel is defined by the standard 2-dimensional Gaussian:

$$N_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{31}$$

Choosing $\sigma = 1.5$ yields a 5×5 kernel as shown in Figure 4.



0.004	0.012	0.018	0.012	0.004
0.012	0.035	0.05	0.035	0.012
0.018	0.05	0.071	0.05	0.018
0.012	0.035	0.05	0.035	0.012
0.004	0.012	0.018	0.012	0.004

Figure 4: 5×5 Gaussian Kernel, $\sigma = 1.5$

Applying the sobel filter to this image without smoothing results in the noisy outline shown in Figure 5. Certain boundaries are difficult to distinguish, such as the outline of the hand. The noise is very visible throughout the background and other objects in the image.

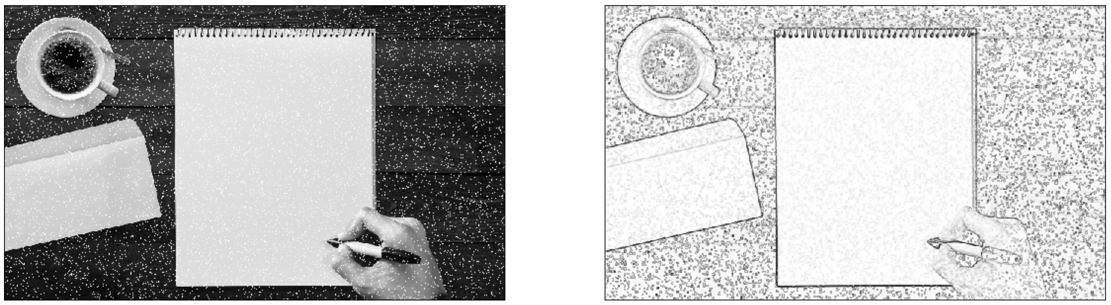


Figure 5: Noisy Image (Left) and Detected Edges from Sobel Filter (Right)

Now, by convolving the image in Figure 5 with the Gaussian kernel $N_{\sigma=1.5}$, we obtain the image in 6.it is worth noting that the combination of the Gaussian blur with the Sobel filter (high pass filter) is essentially bandpass filtering. The noise is mostly smoothed out, and the edge boundaries are bolder and more pronounced. The hand, in particular, is much more easily distinguished. Applying convolution to an image inherently decreases the output size. Notice that, in both of these examples, the output image is the same size as the input image. This is because zero padding is introduced as a pre-processing step, which essentially enlarges the border of the original image with zeros to preserve the same output dimension.

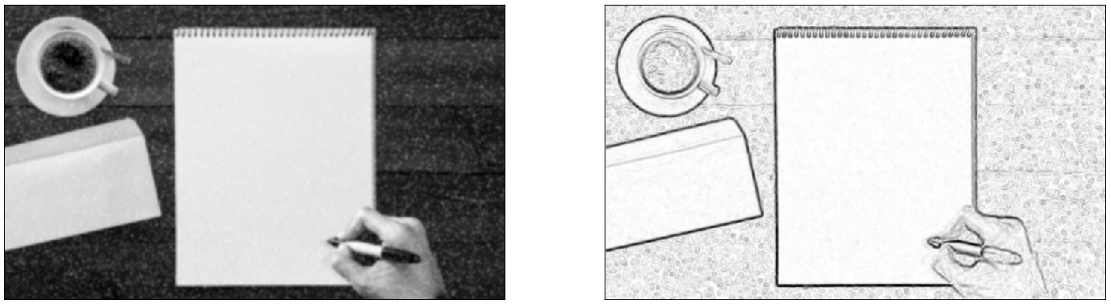


Figure 6: Gaussian Filtered Image (Left) and Detected Edges from Sobel Filter (Right)

3.4 Neural Networks

In neuroscience, the human brain is recognized as the seat of consciousness and center for processing information. The brain is what allows us to think and feel while navigating complex situations. We understand it as a collection of nerve cells which are called *neurons* [31]. Signals are transmitted from one neuron to another through long fibres called *axons*, and received via smaller fibers surrounding the cell body called *dendrites*. At each junction, a *synapse* relays the signal from an axon to a dendrite. When a neuron "fires", it transmits information to another neuron. Given certain stimuli, this can cause a chain reaction between different sets of neurons, some of which "fire together" given a certain input.

A subset of machine learning algorithms, *neural networks*, attempts to loosely mimic this model of the brain. A neural network is a type of biologically inspired supervised learning algorithm composed of many different chained functions [8]. It can be best described by a computational graph, consisting of vertices (called *neurons*) which are interconnected with edges. Each edge has a corresponding weight, which is a parameter learned through an iterative training process. The simplest type of neural network is called a *feed-forward neural network*, which consists of an input layer, any amount of hidden layers, and an output layer. Each layer is composed of some amount of neurons and a bias, except the output layer which does not have a bias. A layer can be thought of as a function, $g(x)$, where x represents the input signal. A visual example of a neural net with one hidden layer is shown in Figure 7.

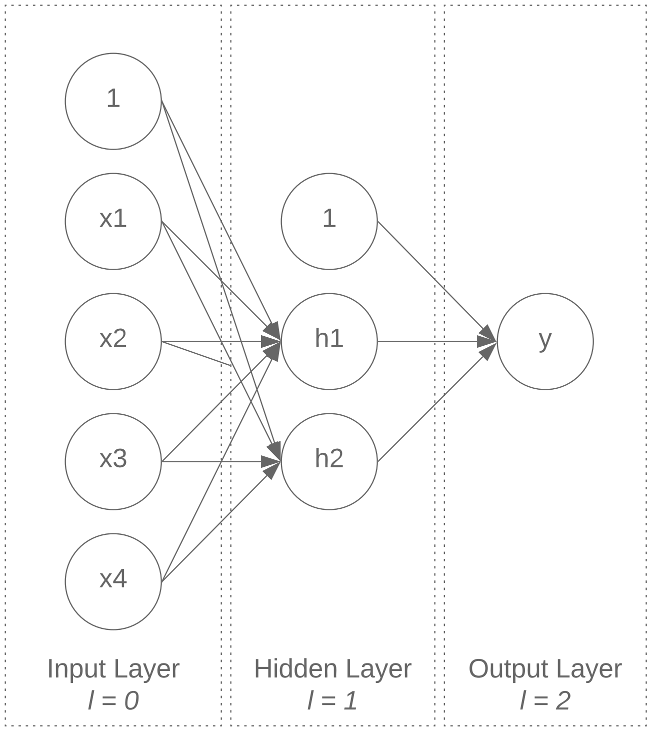


Figure 7: Example Feed-Forward Neural Network

Activation functions determine a neuron's response to an input. Activation functions can be linear (such as the identity function $\theta(x) = x$), but in practice tend to be more complex so that non-linearities can be introduced into a system. By utilizing non-linear activation functions, a network can learn more flexible approximations to the true function $f(x)$. Some popular activation functions include: Unit step, Sigmoid, Hyperbolic Tangent (tanh), Rectified Linear Units (ReLU), ReLU6, and Swish. See Figure 8 for examples of each along with some mathematical definitions. The model builder may find it advantageous to choose less computationally expensive activation functions, such as ReLU or ReLU6, when processing large sets of data. In this thesis, activation functions will be denoted by $\theta(\cdot)$.

A prediction is made when input data is propagated forward through a neural network. Consider a sequence $\mathbf{x} = \{x_0, x_1, x_2, x_3\} \in \mathbb{R}^4$ as an input to the graph as depicted in Figure 7. For the case of regression, each data point could be a stock price ranging from day $n = 0$ to day $n - 3$. The output of the network, y, would represent the price at day $n + 1$. The activation function can be $\theta = \tanh()$ for this example. In order to compute y, the forward propagation would be:

$$y = g(\mathbf{x}) = \mathbf{w}^{(2)T}(\theta(\mathbf{w}^{(1)T}\mathbf{x})) \tag{32}$$

with weights and bias at each layer l represented as $\mathbf{w}^{(l)}$.

A *hyperparameter* is any kind of parameter explicitly set by the model builder. For example, the number of neurons in a layer is a hyperparameter. Others include: the learning rate (how much the model is updated based on training from a single example), weight decay (regularization penalty), and mini batch size (how many examples the model is processing in one forward computation). Hyperparameters often need to be tuned, and changing one may affect another. It is not uncommon to evaluate model performance by varying different hyperparameters using *grid search* or *random search*. A *parameter* is a value which is learned through model training. Network weights are an example of a parameter.

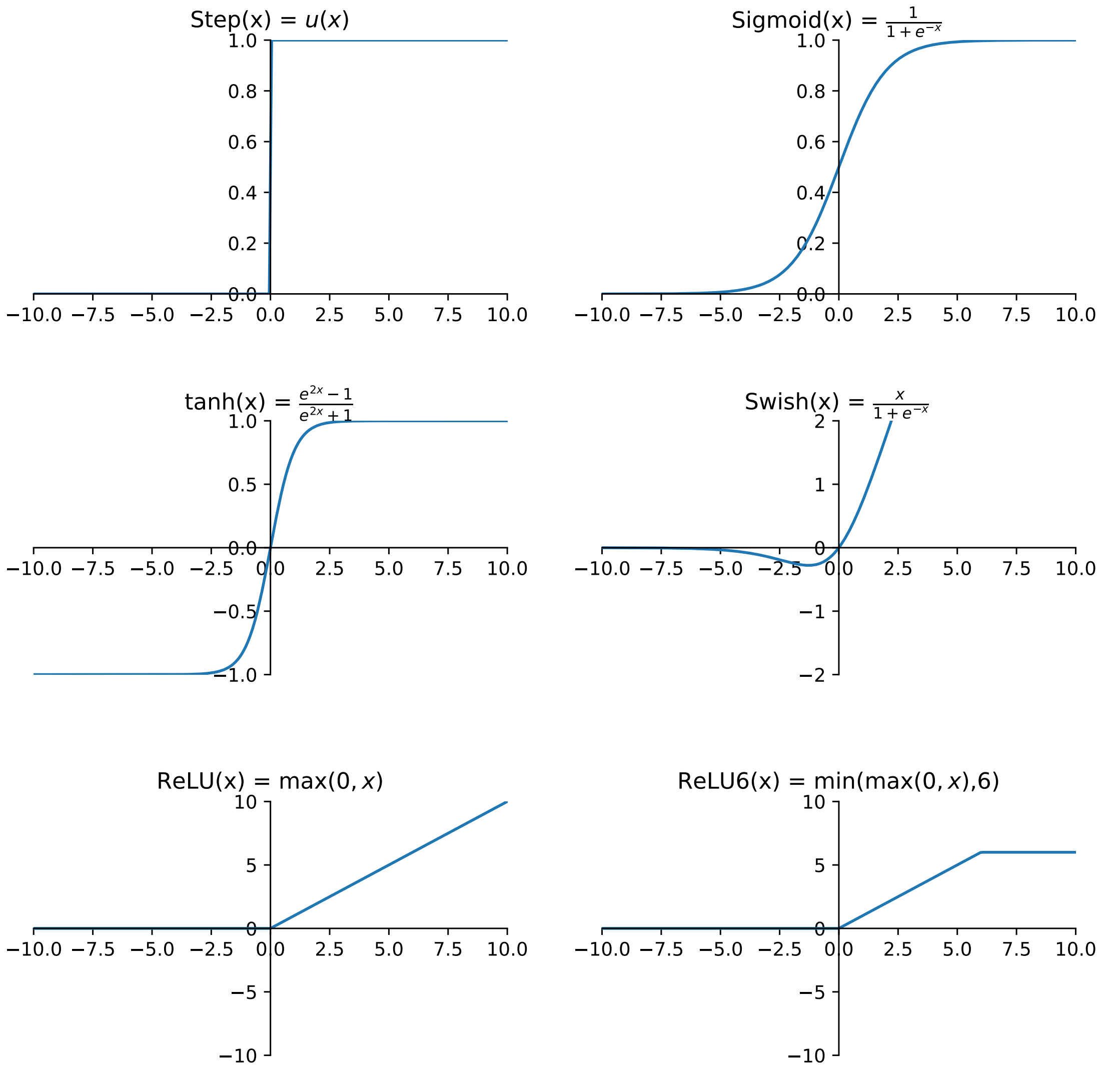


Figure 8: Selected Activation Functions

3.4.1 Descending the Loss Landscape

At the core of training a neural network is minimization of the loss function. The most common way to do this is through the algorithm known as *gradient descent*. Gradient descent aims to find the minima of the loss function by taking steps in the decreasing direction of the function's gradient ∇J, evaluated at some data point [28].

The size of the step taken is determined by the learning rate, η. The algorithm is nicely summarized by Equation 33. At initialization ($t = 0$), the weights for all layers $\mathbf{W} = \{\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(L)}\}$ are set randomly. At each iteration, the weights are updated based on η and ∇J.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla J(\mathbf{W}_t) \tag{33}$$

A typical implementation of this is *stochastic gradient descent*, which performs weight updates based on a single data point $\mathbf{x}$ at a time. When multiple examples are used at once (a *mini-batch*), it is then called *mini-batch gradient descent*. Finally, when the weight update is only performed after all of the training examples are evaluated, it is called *batch gradient descent*.

The learning rate is one of the most important hyperparameters to choose when training a network. If we think about learning rate in terms of how big of a step is taken when traversing a landscape characterized by the loss function, taking too big of a step can result in missing a minimum. Conversely, if the learning rate is too low, then it could take an unreasonable amount of iterations to reach a minimum, if at all. A common practice is to utilize cross validation to choose the most suitable learning rate for a specific dataset. It is also common to reduce the learning rate over time. This is called *annealing*, or learning rate scheduling.

To avoid some of the difficulties which are presented by choosing the best learning rate and schedule, some use optimizers with adaptive learning rates. There are many variants of these, but some popular ones include *Adam* and *Adagrad*. These methods can provide competitive results when compared to SGD, but results vary based on the dataset.

In order to compute the error gradients, a commonly used method in neural networks is *Backpropagation* [28]. It works by first randomly initializing the network weights and computing a forward propagation through the network. Then, starting with the output layer L, the gradient in error with respect to the input signal is evaluated at each layer. This is called the *sensitivity*. The sensitivity at any layer l is defined as:

$$\boldsymbol{\delta}^{(l)} = \frac{\partial \mathbf{e}^{(l)}}{\partial \mathbf{s}^{(l)}} \tag{34}$$

With the error, $\mathbf{e}$, defined as:

$$\mathbf{e} = (\mathbf{x}^{(L)} - y)^2 = (\theta(\mathbf{s})^{(L)} - y)^2 \tag{35}$$

For this case, the squared error function is used, and $\mathbf{s}$ is the output signal at a particular layer. Using the chain rule, the sensitivity at the output layer can be calculated:

$$\begin{aligned} \boldsymbol{\delta}^{(L)} &= \frac{\partial \mathbf{e}^{(L)}}{\partial \mathbf{s}^{(L)}} \\ &= \frac{\partial}{\partial \mathbf{s}^{(L)}} (\mathbf{x}^{(L)} - y)^2 \\ &= 2(\mathbf{x}^{(L)} - y) \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \\ &= 2(\mathbf{x}^{(L)} - y)\theta'(\mathbf{s}^{(L)}) \end{aligned} \tag{36}$$

After which, the remaining sensitivities can be calculated back to layer $l = 1$. Note that, as layer $l = 0$ is the input layer, which does not have weights, it is not necessary to compute the sensitivity. For a layer l, the sensitivity $\boldsymbol{\delta}^{(l)}$ can be calculated:

$$\boldsymbol{\delta}^{(l)} \leftarrow \theta'(\mathbf{s}^{(l)}) \otimes \left[\mathbf{w}^{(l+1)}\boldsymbol{\delta}^{(l+1)}\right]_1^{d^{(l)}} \tag{37}$$

Where $\otimes$ represents the element-wise Hadamard product. Algorithm 1 provides a complete description for computing sensitivities for a single data point and corresponding label. Algorithm 2 defines backpropagation, with the weights updated through gradient descent.

Algorithm 1 Compute Sensitivities for each layer l

```
Input: a data point, $(\mathbf{x}, y)$
Run forward propagation on $\mathbf{x}$ to compute and save:
        $\mathbf{s}^{(l)}$ for $l = 1, \ldots, L$;
        $\mathbf{x}^{(l)}$ for $l = 1, \ldots, L$.
Initialize: $\boldsymbol{\delta}^{(L)} = 2(\mathbf{x}^{(L)} - y)\theta'(\mathbf{s}^{(L)})$
        $\theta'(\mathbf{s}^{(L)}) = \begin{cases} 1 - (\mathbf{x}^{(L)})^2 & \theta(\mathbf{s}) = tanh(\mathbf{s}); \\ 1 & \theta(\mathbf{s}) = \mathbf{s}. \end{cases}$
for $L = l - 1$ to 1 do
    Let $\theta'(\mathbf{s}^{(l)}) = \left[1 - \mathbf{x}^{(l)} \otimes \mathbf{x}^{(l)}\right]_1^{d^{(l)}}$
    Compute the sensitivity $\boldsymbol{\delta}^{(l)}$ from $\boldsymbol{\delta}^{(l+1)}$:
        $\boldsymbol{\delta}^{(l)} \leftarrow \theta'(\mathbf{s}^{(l)}) \otimes \left[\mathbf{w}^{(l+1)}\boldsymbol{\delta}^{(l+1)}\right]_1^{d^{(l)}}$
end
```

Algorithm 2 Backpropagation

```
Initialize all weights $w_{i,j}$ at random
for $t = 0, 1, 2 \ldots$ do
    Pick $n \in \{1, 2, \ldots, N\}$
    Forward: Compute all $x_i^{(l)}$
    Backward: Compute all $\delta_j^{(l)}$
    for $l = 1 \ldots L$ do
        for $i = 0 \ldots N$ do
            Update the weight at layer $l$, neuron $i$: $w_{i,j}^{(l)} \leftarrow \mathrm{w}_{i,j}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
        end
    end
    Iterate to the next step until it is time to stop
end
```

3.4.2 Batch Normalization

In many modern architectures, *batch normalization* is used in between layers. Batch normalization essentially applies a transform to data by subtracting from the mini-batch the mini-batch mean, $\mu_{\mathcal{B}}$, and dividing by the mini-batch variance, $\sigma_{\mathcal{B}}^2$. This helps improve the training speed of the network by reducing the *internal covariate shift*, which is the change in the distribution of network activations due to the change of network parameters when training [33]. Since training examples are processed by the network in conjunction with other members of the mini-batch, batch normalization also helps with regularization by reducing deterministic patterns in examples which may be learned by the network.

3.5 Convolutional Neural Networks

In this section, a brief overview of *Convolutional neural networks* (CNNs) is provided, as these networks are used heavily in this thesis for classifying images of food waste.

CNNs are a type of feed-forward neural network which utilize convolution to extract various types of descriptive features from a given input. The history of CNNs reaches back to the development of the perceptron by Frank Rosenblatt in 1958 [34]. The perceptron was a machine designed for image recognition which made use of photocells and electric motors to learn weights. It did have some success, but was ultimately only able to perform linear separation of patterns. One famous example is the inability to learn the XOR function [35]. Later on, non-linear activation functions would be introduced to help overcome this limitation. Experiments conducted by Hubel and Wiesel in the early 1960s [36] in part helped lead to the idea of applying filters to images to extract local features, such as lines and edges. These experiments involved connecting electrodes to a cat's visual cortex and observing how the neurons are locally sensitive and orientation sensitive. Decades later, one of the earliest and major successes in CNNs was the development of LeNet-5 [11], which was a convolutional neural network applied to recognize handwritten digits at over 99% accuracy. In recent years, success of CNNs has exploded, as we are able to train deeper, more complex networks on arrays of GPUs.

Through convolution, CNNs can reduce a sequence or an information rich image into characteristic patterns which are more amenable to classification. This is one reason that they are much more accurate than vanilla feed-forward networks at many computer vision tasks, although they are also used for regression and other purposes. Each convolutional layer is composed of an arbitrary number of filters, with weights learned through optimizing on a specific dataset. For each filter, a corresponding activation map is produced. Similar to a basic feed-forward network, these maps become the input to the next layer, and are typically followed by an activation function. The activated maps are then subsampled using *pooling* operations. These terms will be described in more detail in the following subsections. If classification is the goal, the final features are then pooled and flattened into a single vector which is fed into a fully connected layer. In Figure 9, a typical CNN architecture is depicted. The features learned are shown by the square "feature maps" in each layer. The convolutional kernels are depicted as the square filters scanning each feature map. The final features used for classification are shown in the "fully connected" layer, which learns a mapping between these features and the output class.

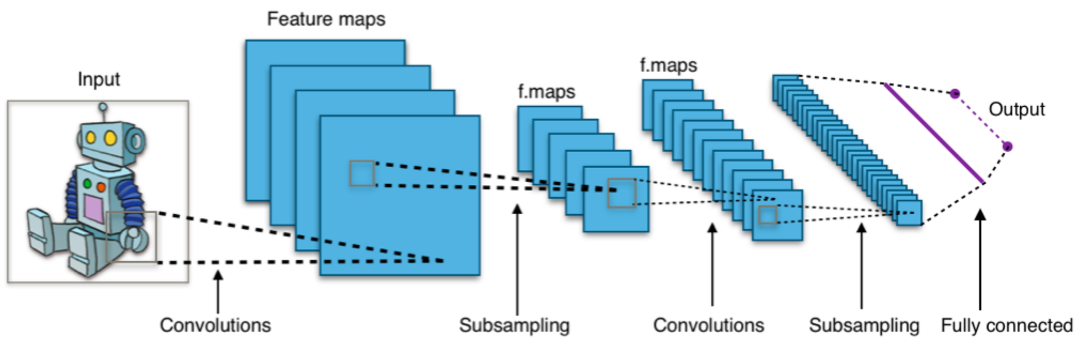


Figure 9: Typical CNN Architecture [2]

Most modern CNNs are composed of several layers, and are considered to be "deep" networks, part of a field called *deep learning*. A deep network can be any network consisting of multiple layers, and not restricted to just those which utilize convolutional layers. Some networks, such as LeNet-5 [11], which only consists of seven layers, are considered to be "shallow". Others, such as ResNet-152 [37] which consists of 152 layers, are considered very deep. Deeper networks are mostly inspired by how CNNs learn to detect features. Take face detection as an example [38]. In the beginning layers of the network, kernels are learned which detect basic components, such as lines and corners of a face. As we go deeper into the network, more complex shapes are learned, such as eyes and noses. Some of the deepest layers are capable of learning the shape of the face, distinguishing between expressions. It has been found that these learned abstractions are more amenable to classification than previous methods.

Not only are convolutional layers good at detecting patterns in data, but they also consume less space in memory than a feed-forward network through *parameter sharing*. Parameter sharing means that the filter weights are shared to detect features across an entire input, rather than utilizing a single parameter per feature such as in a typical feed-forward network.

In practice, the "convolution" in a CNN is actually cross-correlation. This essentially checks the similarity between the features encoded in the kernel and the feature map which it is correlated with. High similarity will result in detected features. Recall that 2-D correlation between a square $N \times N$ kernel, K, and input feature map, I can be defined as:

$$(K \star I)(x, y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} K(i, j) I(x + i, y + j) \tag{38}$$

which is almost identical to convolution, except the input feature map is not flipped.

3.5.1 Pooling

After performing convolution and applying the activation function to each feature map, a *pooling* operation is applied. Pooling essentially performs sub-sampling on a feature map to obtain a more compact, summarized representation [8]. This allows a reduction of overall parameters in a network, which means less space in required in memory. A pooling layer implements a specific pooling function over activation map patches of size $n \times n$. Typically, $n = 2$. Some examples of pooling functions include: Average pooling, Max pooling, and Global pooling.

Average pooling simply averages the values in each patch. Similarly, *max pooling* extracts the maximum value in each patch. Finally, *global pooling* can apply either the average or max functions to the entire image, rather than individual patches. This results in one value to summarize an entire activation map. An example of each pooling function applied to a 4×4 grid of values can be seen in Figure 10. Note that the global max pooling would result in a scalar value of 5.0, as that is the maximum value on the layer. Applying global average pooling would result in a scalar value of 3.0.

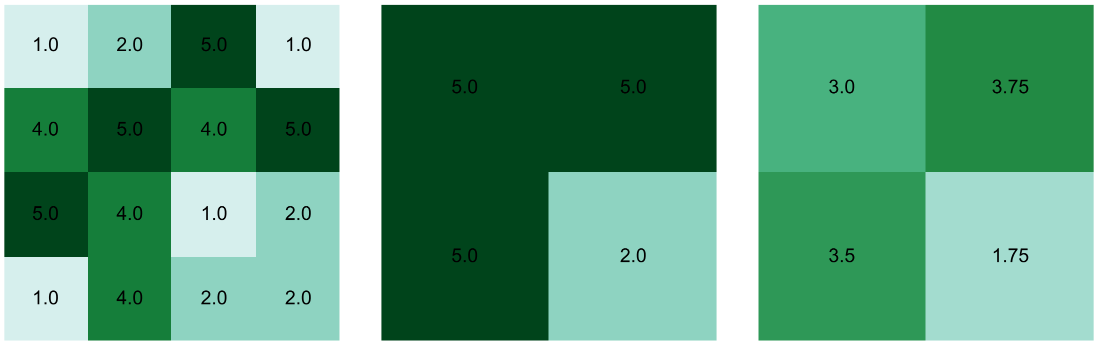


Figure 10: Left: Original 2×2 layer. Middle: Downsampled with 2×2 Max Pooling. Right: Downsampled with 2×2 Average Pooling.

3.5.2 Hyperparameters

Key hyperparameters specific to a CNN include the number of feature maps, kernel size, stride, number of layers, and zero padding. The hyperparameters described in Section 3.4 also still apply. It is important to note that a change in any of these hyperparameters can affect the overall training and performance of the network. For example, selecting more feature maps at a given layer means that the network can learn to distinguish a larger variety of features. This also increases the overall parameter count and volume of the network.

The *kernel size* determines the size of the window which is convolved with the input feature map. The height typically matches the width. Larger kernel sizes result in smaller output feature maps, and vice versa. *Stride* refers to the step size of the kernel when convolved with the image. With a default setting of 1, the kernel will slide 1 pixel at a time. Larger values will result in skipping pixels. *Zero padding* refers to the practice of adding zeros around the border of an image or map. This can be useful for increasing the output size of a convolution, or preserving the size throughout the network [39].

3.5.3 Depth-wise Separable Convolution, Mobile Inverted Bottlenecks, and EfficientNet

Now that the core concepts behind CNNs have been introduced, the model chosen for this thesis, EfficientNet [25], will be discussed. In Section 2.2, EfficientNet is introduced as a family of CNNs which is able to achieve or match state-of-the-art performance on many benchmark datasets. The motivation behind developing this family of models was to find a better way to scale up neural network architectures. It has been a common practice to scale the depth (number of layers), width (size of the convolutional kernel), or input image size (resolution) of the network independently in order to increase accuracy. The EfficientNet paper aims to carefully balance network depth, width, and resolution to produce an efficient and highly accurate network. In this sub-section, EfficientNet will be discussed at length to give the reader a better understanding of its inner workings.

Bottleneck layers were first introduced by He, et al [37] in their ResNet paper. In a bottleneck layer, the input tensor's depth is reduced by a 1×1 point-wise convolution. A spatial 3×3 convolution is then applied on the reduced feature set, which is less expensive to compute. Then, the original dimension is recovered with another 1×1 convolution. In short, the bottleneck layer compresses the input, performs the spatial convolution, and then expands it again.

The main building block of many "efficient" architectures, such as EfficientNet, is the *mobile inverted bottleneck*. These are sometimes abbreviated as MBConv, and represent a block of layers. Two variants are the building blocks of EfficientNet —MBConv1 and MBConv6. These are shown in detail in Figure 13, and seem to have originated in the MnasNet architecture [40].

In order to understand mobile inverted bottlenecks, we need to explain *depth-wise separable convolution*. Suppose we want to apply the Sobel filter (see: Section 3.3) to a multi-channel input, like an RGB image. A depth-wise separable convolution will yield the same result as a standard convolution, but with a reduced computational cost achieved by splitting the convolution into two separate stages [41]. The first stage convolves a filter across each image input channel. Then, a single 1×1 point-wise convolution is applied depth-wise, to combine the output of each channel. This type of convolution is used heavily in networks like EfficientNet and has proven to reduce model size, while maintaining high accuracy. A visualization is shown for a $(k \times k \times 3)$ 3D-convolution in Figure 11 and the equivalent separable convolution in Figure 12.

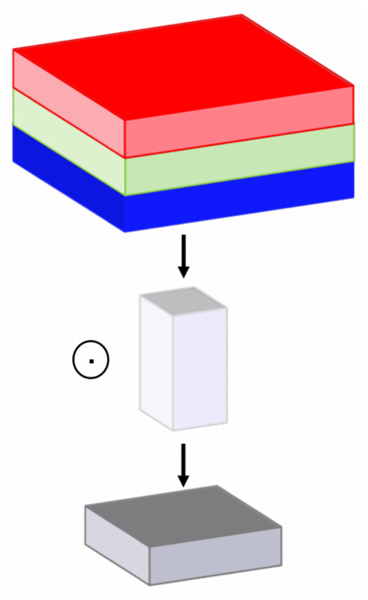


Figure 11: 3D Convolution With $(k \times k \times 3)$ Kernel.

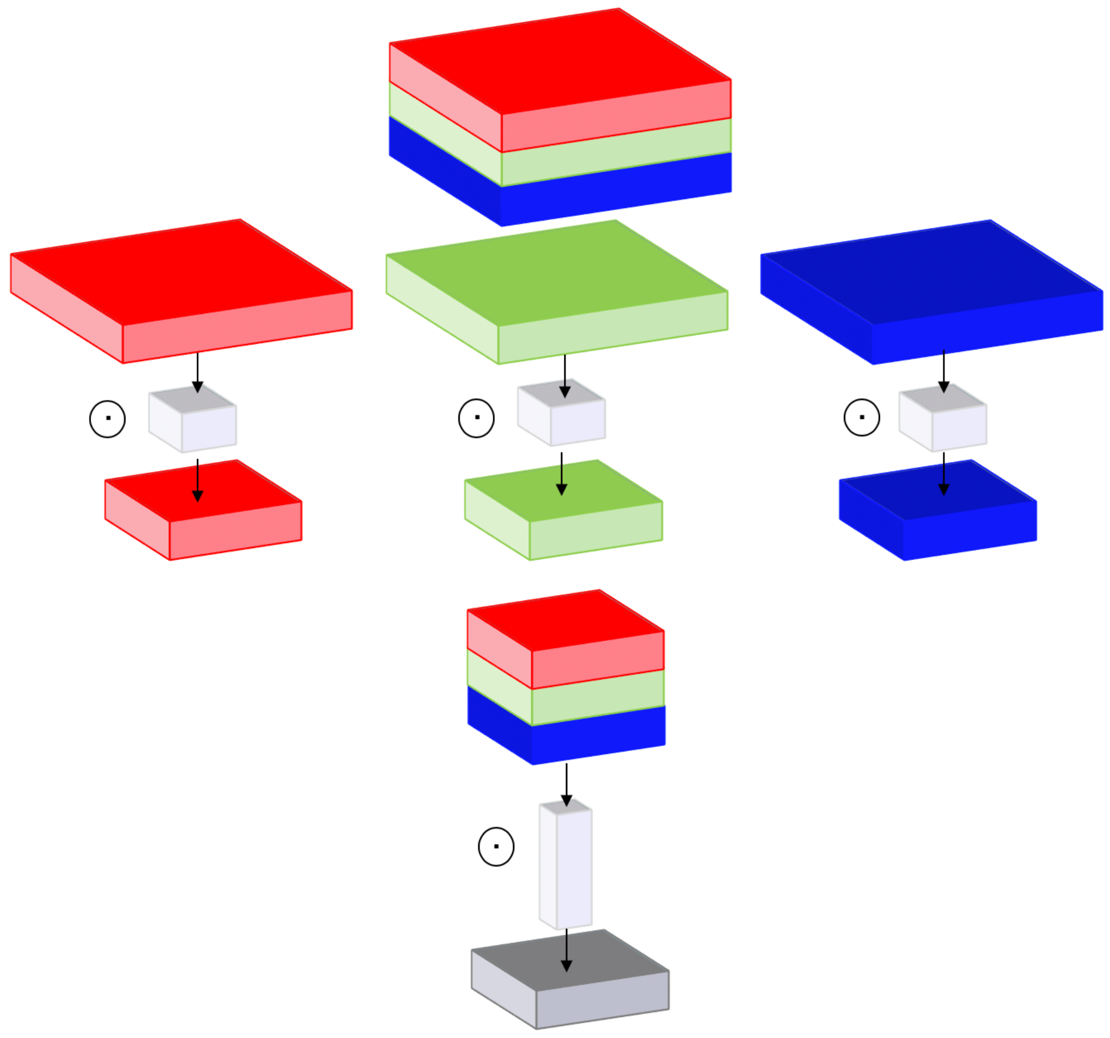


Figure 12: Depth-wise Separable Convolution with Three $(k \times k)$ Kernels and One $(1 \times 1 \times 3)$ Kernel.

Now we can return to explaining mobile inverted bottlenecks. Mobile inverted bottlenecks are very similar to a bottleneck layer, but instead first use a 1×1 convolution to expand the input. A spatial convolution is then applied with a $k \times k$ kernel, and the image is then compressed to the original number of feature maps with another 1×1 convolution. These two stages are the depth-wise separable convolution. In between each convolution, batch normalization and ReLU activation is applied. In Figure 13, both the MBConv6 and MBConv1 are described in this manner. Notice that MBConv1 does not include the initial 1×1 expanding convolution, and also has an added Squeeze-Excite block. The addition of a Squeeze-Excite block allows the network to weight the features learned from each channel differently, rather than uniformly.

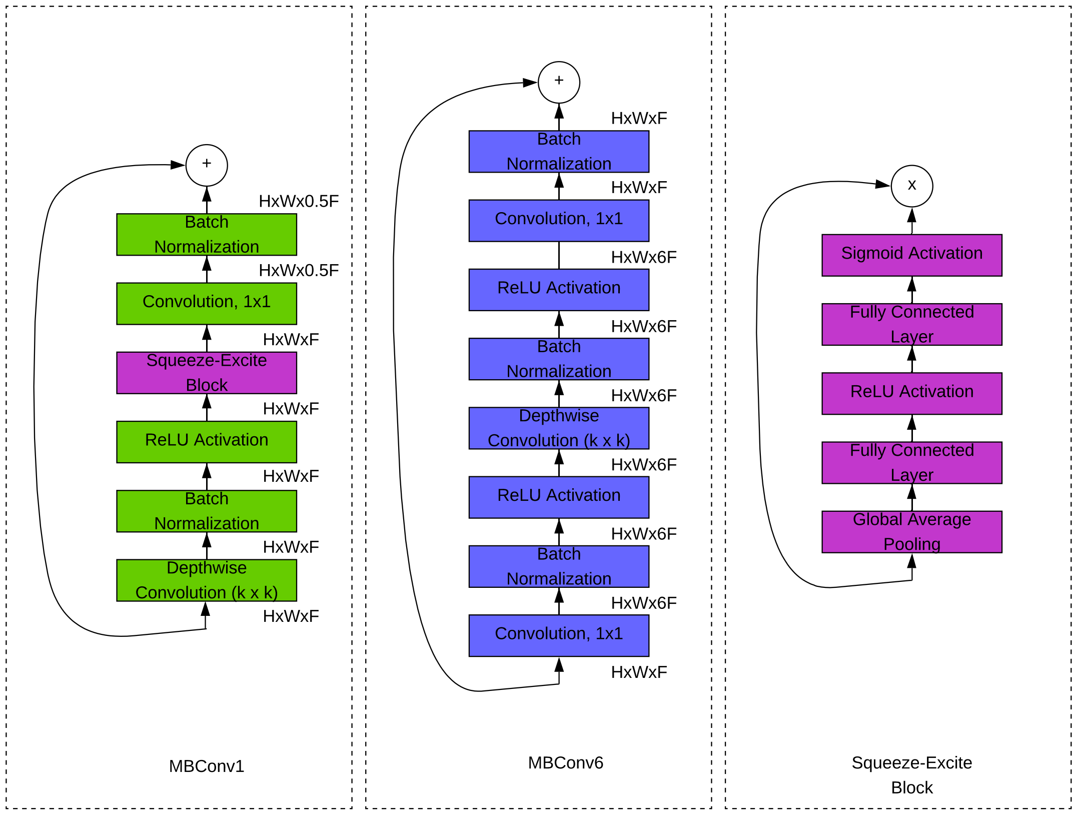


Figure 13: Mobile Inverted Bottleneck Blocks. HxWxF Refers to Input Tensor Size in Terms of Height, Width, and Channels.

The EfficientNet architecture was found by performing a *neural architecture search* as done in [40], with added constraints on FLOPS (floating point operations per second)

Stage (i)	**Operator** ($\hat{\mathcal{F}}$)	**Resolution** ($\hat{H}_i \times \hat{W}_i$)	**#Channels** ($\hat{C}_i$)	**#Layers** ($\hat{L}_i$)
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 1: EfficientNetB0 Architecture

and network parameter count. The FLOPS target for this work is 400M. Through this search, the EfficientNet-B0 baseline architecture was obtained as shown in Table 1. In this table, each stage (group of layers) is denoted as the index i. Each $\hat{\mathcal{F}}$, $\hat{H}_i$, $\hat{W}_i$, $\hat{C}_i$, and $\hat{L}_i$ are the estimated parameters arrived at through the neural architecture search.

Next, the dimensions of each network layer are optimized. They begin by defining the network as:

$$\mathcal{N} = \bigodot_{i=1...s} \mathcal{F}^{L_i} \left(X_{<H_i, W_i, C_i>} \right) \tag{39}$$

Where H, W, and C represent the input image size to the network $\mathcal{N}$, corresponding to height, with, and number of channels. The $\odot$ operator is used to denote the network as being a list of composed layers (L_1, L_2, ..., L_s) which the input X is propagated throug each layer. With this notation, it is possible for each layer to have a unique configuration. In order to reduce the search space, a constraint is placed so that all layers must be scaled uniformly. The search is then formulated as an optimization problem:

$$\begin{aligned}\underset{d,\, w,\, r}{\arg\max} \quad & \text{Accuracy}\big(\mathcal{N}(d,\ w\ ,r)\big) \\ s.t. \quad & \mathcal{N} = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}^{d\cdot\hat{L}_i}\left(X_{<r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i>}\right) \\ & \text{Memory}(\mathcal{N}) \leq \text{target\_memory} \\ & \text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}\end{aligned} \tag{40}$$

where $w,\ d,\ r$ are coefficients that scale the network's width, depth, and resolution. A compoound scaling method is proposed, which utilizes a coefficient ϕ to uniformly scale each dimension:

$$\begin{aligned}\text{depth:} \quad & d = \alpha^{\phi} \\ \text{width:} \quad & w = \beta^{\phi} \\ \text{resolution:} \quad & r = \gamma^{\phi} \\ \text{s.t.} \quad & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1\end{aligned} \tag{41}$$

The constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ is added to ensure that for any new ϕ, the total FLOPS will approximately increase by 2^{ϕ}. In order to find suitable coefficients, the following two step procedure is applied:

1. They fix ϕ =1, assuming twice more resources available, and do a small grid search of α, β, γ based on Equations 40 and 41. They found the best values for EfficientNet-B0 are $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$, under the constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.

2. α, β, γ are then fixed as constants, and the baseline network is scaled up according to different ϕ as defined in Equation 41. Through this, EfficientNet-B1 through B7 are obtained.

While performance results in [25] yield state-of-the-art accuracy using the largest model, EfficientNet-B7, the experiments defined in Section 5.2 utilize the baseline model, EfficientNet-B0. The baseline model is used due it its small footprint and modest input size, (224×224), which requires less computing resources to train. Our dataset is also less complex than Food-101 [9]. In Appendix B, results are shown using a larger model.

3.5.4 Image Classification

As touched on in Section 1.1.2, image classification refers to a process which yields a list of predicted categories for a given input image. While in that section, feature extraction and classification were described as separate processes, CNNs typically combine the two steps into one network. The convolutional layers are responsible for detecting features by propagating the input forward through each layer, which are then fed into a fully connected layer at the head of the network. Image classification is useful for identifying if a single object is present in a given example, and is the main focus of this thesis. Specifically, in each case a CNN is used to both extract relevant features in an image and perform classification with a fully connected layer at the head of the network.

3.5.5 Object Detection

While image classification is great for identifying whether a single object of interest is present, sometimes we want to identify and localize several objects in one image. This can be accomplished by *object detection* [42]. Object detection refers to localizing an object in a given image by drawing a bounding box around it, and then assigning it to a class. In order to train a model capable of object detection, it is necessary to have a set of annotated bounding boxes defining the regions of interest and their respective classes. The set of annotations is referred to as the *ground truth*. There are many different architectures which are open-sourced and available to perform object detection on custom datasets. Many of these utilize image classification models as backbone feature extractors.

3.5.6 Semantic Segmentation

Beyond drawing bounding boxes, *Semantic Segmentation* attempts to not only classify and localize an object in an image, but also identify boundaries containing all related pixels. Similar to object detection, models which perform Semantic Segmentation often utilize other CNN architectures as a backbone to predict *masks* for an input image [42]. A mask is simply a set of values which can be overlaid the original image, corresponding to a class assignment for each pixel location.

3.5.7 Instance Segmentation

Through object detection, we are able to predict bounding boxes, and through semantic segmentation, we are able to classify which pixels belong to a given object. What if we want to do both? This is where *instance segmentation* comes in handy

[42]. Many of the same architectures which perform semantic segmentation can be extended to predict both a bounding box and classify the pixels for a given object. The goal of a model performing instance segmentation is to identify which instance the segmented pixels belong to. This type of segmentation could be especially useful for object counting. While it is not the main focus of this thesis, some sample instance segmentation results on the food waste dataset are shown in Appendix C.

3.5.8 Gradient-Weighted Class Activation Mapping

When evaluating the predictions of a CNN, it can sometimes be difficult to interpret how the model arrives at a particular confidence score for a given input image. In an attempt to demystify this process, make CNNs more transparent, and build trust in these architectures, *Grad-CAM* was developed [43]. Given a class activation, Grad-CAM examines the gradient information flowing into the last layer of the CNN and traces it back to the original feature maps. This is accomplished through calculating neuron importance weights, α_k:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{42}$$

which is simply the global average pooling of the gradients obtained via backpropagation for class c, output y^c (before softmax), and with respect to feature maps A^k of a convolutional layer. The quantity Z represents the number of pixels in the feature map. The activation maps are then weighted by each neuron importance, followed by a ReLU to only retain maps which have a positive influence on the class:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k\right) \tag{43}$$

A consequence of this is that the resulting heat-maps are the same size as the feature maps in the last layer of the network. In order to localize this information in the same spatial coordinates as the input image, $L^c_{\text{Grad-CAM}}$ is upsampled using bi-linear interpolation and fused with Guided Backpropagation via pointwise multiplication.

One of the advantages of Grad-CAM is that it can be used with any CNN architecture. As shown in subsequent sections, this is a valuable tool for troubleshooting a CNN's learned feature extraction capabilities. In this thesis, Grad-CAM visualizations are used to help better understand where in an image a model is extracting relevant features when making a classification.

3.6 Transfer Learning

Building and training a model from scratch can be very computationally expensive, especially when a large dataset is used. What if the process could be enhanced by leveraging knowledge gained from learning a different task? This is the idea behind *transfer learning*. Transfer learning refers to the practice of utilizing a model which has already been trained on a dataset on an entirely new problem. The model, in this case, is called a *pretrained model*, and the retraining process is referred to as *fine-tuning* the model. Fine-tuning can be a bit of an art, and there are many different techniques employed by practitioners. Typically, the final classification layer on the pretrained model is removed and replaced with a fresh layer sized for the new task. The model is then trained and evaluated on the new dataset. In this process, layers of the CNN can be selectively *frozen*, such that their weights are not updated during backpropagation. Sometimes, the entire network is frozen, except for the final classification layer, to allow for initialization. Then, from the penultimate layer back, layers are selectively unfrozen. The best approach varies depending on the data.

Sometimes, instead of fine-tuning, the CNN is used as a feature extractor. In this case, all of the pre-trained model's weights are frozen except for the classification layer, which learns patterns in the extracted features. It is also possible to feed these feature maps into another algorithm for classification, such as Random Forests or SVM.

For image classification, the most common dataset used in transfer learning is *ImageNet* [44]. In this thesis, transfer learning is used to accelerate model training speed and boost overall classification accuracy. Specifically, the model is loaded with and fine-tuned with ImageNet pretrained weights.

3.7 Performance Metrics and Best Practice

Before applying a learning algorithm to a dataset, it is critical to first understand the data. This entails manually reviewing examples, building a working knowledge of the class distributions, checking for outliers and anomolies, and getting a good grasp for any patterns which might be present. Classes with too few images may need more data to learn from, or may need to be eliminated if more data cannot be gathered. Classes with too many examples may need to be pruned to rectify imbalance. Raw data can often suffer from class noise as well, the process of correcting this may modify the class distributions.

Once the data is well understood, it can be partitioned into training, validation, and test sets. The training and validation sets are important, as these facilitate tuning various hyperparameters for the model and *cross validation.* Bias can be introduced indirectly when building a model if performance on the test set influences chosen hyperparameters. Cross validation allows for the model hyperparameters to be tuned on validation data, but ultimately reserves some data for testing for which performance has yet to be seen. For this reason, performance on the validation set can be observed, after which the training and validation sets merge, the model is trained on both, and then scored against the test set. This allows for an unbiased estimate of the out of sample error.

Typically, the primary performance metric for classification on major benchmark datasets is *accuracy.* For the binary case, accuracy can be defined as [30]:

$$\text{Accuracy}(\%) = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} * 100 \tag{44}$$

For which, TP represents the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives. Accuracy can be simply described as a summary of how often the model predicts the correct labeled class, out of all labeled examples. This can be a useful metric when errors in prediction are equally important. Some other useful metrics such as *precision* can be evaluated:

$$\text{Precision}(\%) = \frac{\text{TP}}{\text{TP} + \text{FP}} * 100 \tag{45}$$

and *recall*:

$$\text{Recall}(\%) = \frac{\text{TP}}{\text{TP} + \text{FN}} * 100 \tag{46}$$

Precision is the ratio of how many times a class is correctly predicted over all predictions for that class. Recall is the ratio of how many times the class was predicted over all instances of that class. For a security system utilizing iris recognition, precision may be more important than accuracy because of concern for authenticating the wrong person (false positive). However, for an e-mail spam filter, recall may be more important due to false classification of an item which is not actually spam. It is important in practice to find a balance between both of these metrics.

In scenarios with class imbalance, the precision-recall curve is useful for evaluating the skill of a classifier. The precision-recall curve is very similar the more common receiver operating characteristic (ROC) curve, but there are some important differences. The precision-recall curve plots precision (also known as the positive predictive value) against recall (also called the true positive rate) for a chosen class, with precision on the y axis and recall on the x axis. The ROC plots false positive rate against recall, which is useful in a setting with balanced class sizes. For the precision-recall curve, a naive classifier with no skill would be represented as a horizontal line at $y = 0.5$, with skill evaluated by examining the trade-off between precision and recall. This is done by varying a threshold on the classifier's positive prediction score for the given class from 0 to 1.0. By raising this threshold, we require that the classifier has more confidence in a positive prediction. This results in less false positives, but can result in more false negatives, which means that precision increases while recall decreases. The converse is also true —as the threshold decreases, there are less false negatives, but more false positives. As precision considers false positives, it will decrease, but recall ignores these and will increase. The precision-recall curve can be extended to provide a single curve in a multi-class scenario through *macro* or *micro averaging*. For precision, a micro average can be defined as [45]:

$$\mathrm{PRE}_{\mathrm{micro}} = \frac{\mathrm{TP}_1 + \mathrm{TP}_2 + \ldots + \mathrm{TP}_k}{\mathrm{TP}_1 + \ldots + \mathrm{TP}_k + \mathrm{FP}_1 + \ldots + \mathrm{FP}_k} \tag{47}$$

for k classes. Notice that the micro average takes into account the amount of true and false positives from each individual class. Macro averaging precision can be calculated as follows:

$$\mathrm{PRE}_{\mathrm{macro}} = \frac{\mathrm{PRE}_1 + \ldots + \mathrm{PRE}_k}{k} \tag{48}$$

which is simply an average precision score over each class.

When precision and recall are both important, the *F-Score* can be used to combine the two into a single performance measure. There are variations of the F-Score which can weight either precision or recall differently, but here it is assumed that they are equally important. The F-Score can be calculated with the following equation:

$$F_1 = \frac{2 * \mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{49}$$

Another useful method for evaluating model performance is the *confusion matrix*.

The confusion matrix is a tabular representation of true/false positives and true/-false negatives. Values obtained from the confusion matrix can be used to calculate precision and recall as previously defined. It is extremely useful for identifying and troubleshooting a model's tendency to classify examples.

4 Food Waste Datasets

The primary dataset used in this thesis consists of $55,023$ images of food waste collected from four customer sites. These sites were deemed (anecdotally) as having "high quality metadata", relative to other potential sites. In total, there are 52 distinct waste item names and 55 distinct container classes in the metadata. The Waste Item Name corresponds to common food groups, such as Vegetables, Beef - Ground, Melons, and so on. Similarly, the container Name corresponds to commonly used kitchen containers, such as different sizes of hotel pans and lexan containers. Other information is available, such as the name of the container the food is in, unique waste event identifiers, unique image paths, and more. For the purposes in this thesis, the main focus will be on the *container name* and *waste item name*. Several subsets were created to train different models, these are listed in Table 2.

Another dataset was created based on different customer sites, to test transfer learning capabilities. This is simply referred to as the location X dataset. Two different models are trained on this dataset to observe learning from a network which was pretrained on ImageNet, and learning on a network which was also trained on the waste item dataset. The aim is to highlight the usefulness of fine tuning on an intermediate dataset which is more closely related to the intended classification task.

4.1 Data Cleaning

Real world data is often noisy. In a classification scenario, this can mean that many of the assigned labels do not agree with the true class. In this subsection, the data cleaning process is described to help rectify some of these differences. Data cleaning is important, especially in the test set, because we can inaccurately judge a classifier's performance with noisy test data. As previously mentioned, some training data noise is tolerable, but extreme noise can have a negative effect on model learning.

It was necessary to perform some cleaning on the dataset before training the model. The original subset contained metadata for $67,238$ images of food waste events. However, some of these images were corrupt and could not be opened. These were filtered out by loading the metadata into a data frame and using a multi-threaded try-catch image open routine. Any image which could not be opened was removed from the data set, reducing the usable size to $61,051$ examples.

The subset was then partitioned into training and test sets using an 80/20 split, respectively. Images in the test set were removed if they were misclassified (the metadata does not match the actual class), heavily mixed with another class (example: vegetables and fruit mixed in a bin), were too blurry or occluded to reasonably identify the image content, or contained a blank image of the device's scale (taken by mistake or after product was weighted). Entire classes were removed if they did not contain enough actual test images after cleaning. The training set was left noisy, but classes removed from the test set were also removed from the training set, along with classes with less than 30 examples. This process reduced the size of the dataset to $55,023$ examples. Randomly selected samples from the test and train sets are shown in Figure 14. Notice how in the training set, some images are blank or do not correspond to the labeled class.

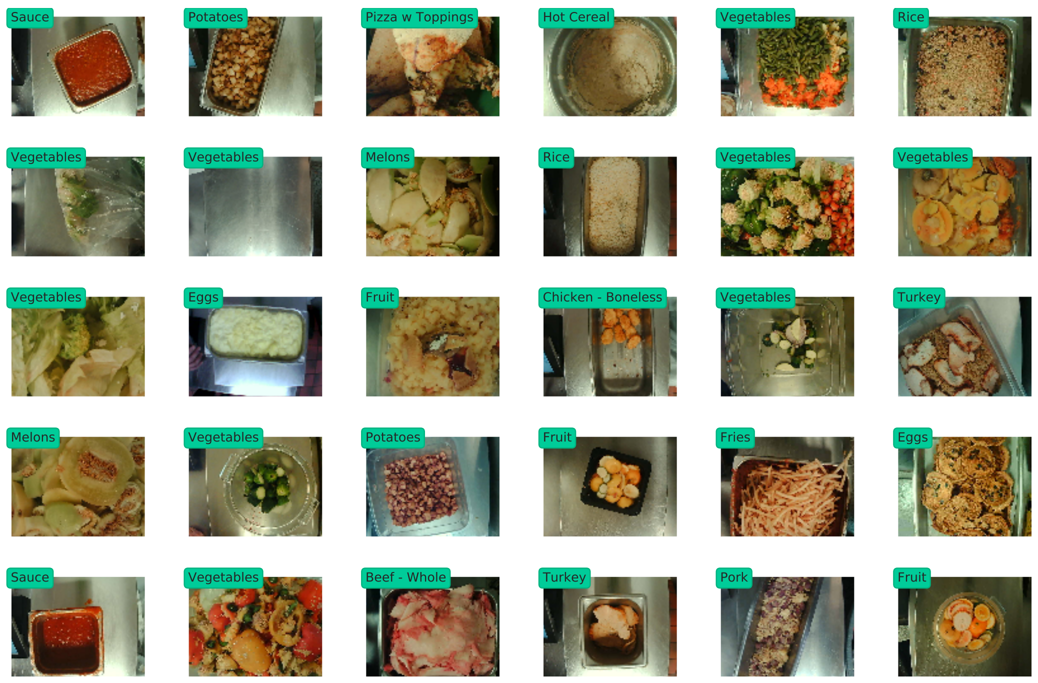
Dataset	**Number of Classes**	**Training Set Size**	**Test Set Size**
Food Waste	52	49829	5194
Container	53	42343	4007
Grouped Container	24	40434	3869
Container / No Container	2	18020	2027
Location X	36	2878	733

Table 2: Datasets and Partition Sizes

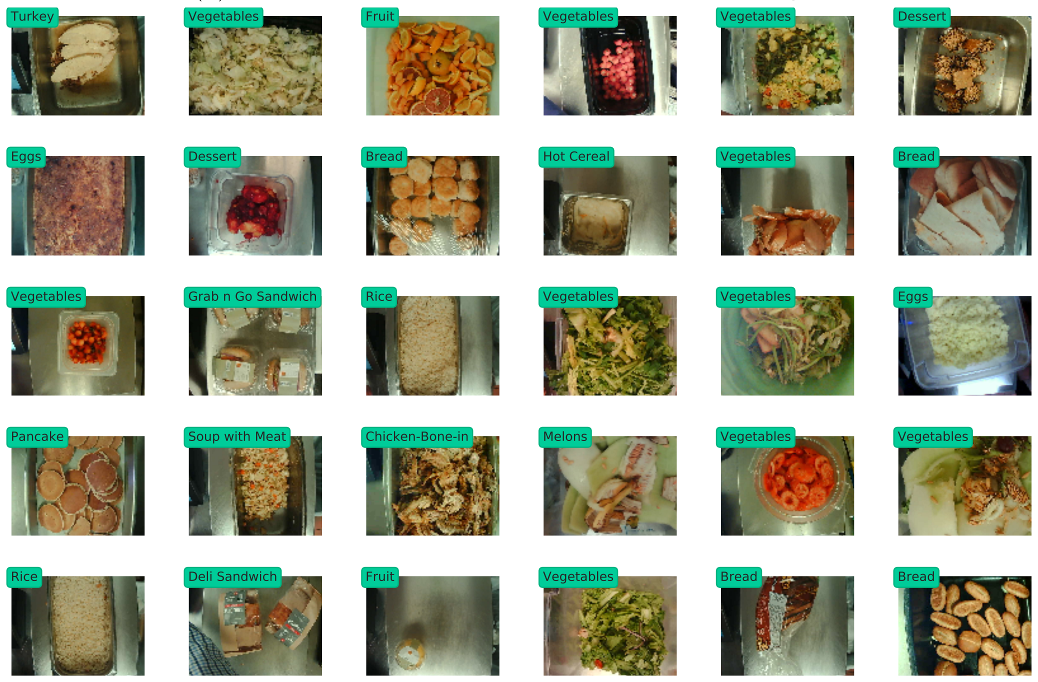
For containers, the same subset was cleaned a second time to remove class noise. Misclassified containers were removed from the data set, along with those with less than 50 examples. Two separate subsets were created —the first separates containers in terms of depth and the second ignores depth. The second is referred to as a grouped container dataset. In addition, the “No Pan” category was separated to create another container/no container dataset.

4.2 Model Architecture

For both the food and container classification models, the PyTorch implementation of EfficientNet-B0 architecture was used [25]. PyTorch is a popular framework used for building and training neural networks, which is written in Python. It allows for the use of GPUs when training neural networks, which greatly reduces overall training time compared to CPUs. The EfficientNet architecture was chosen because it offers high performance when benchmarked against the Food-101 dataset [9] (88% top 1) with a modest parameter count (5.29M). The model was fine-tuned using the pre-trained ImageNet weights [46] in order to take advantage of the overall accuracy boosts and training time reductions gained through transfer learning (this is explored in Appendix A). By default, EfficientNet-B0 accepts a $(3, 224, 224)$ input tensor rep-

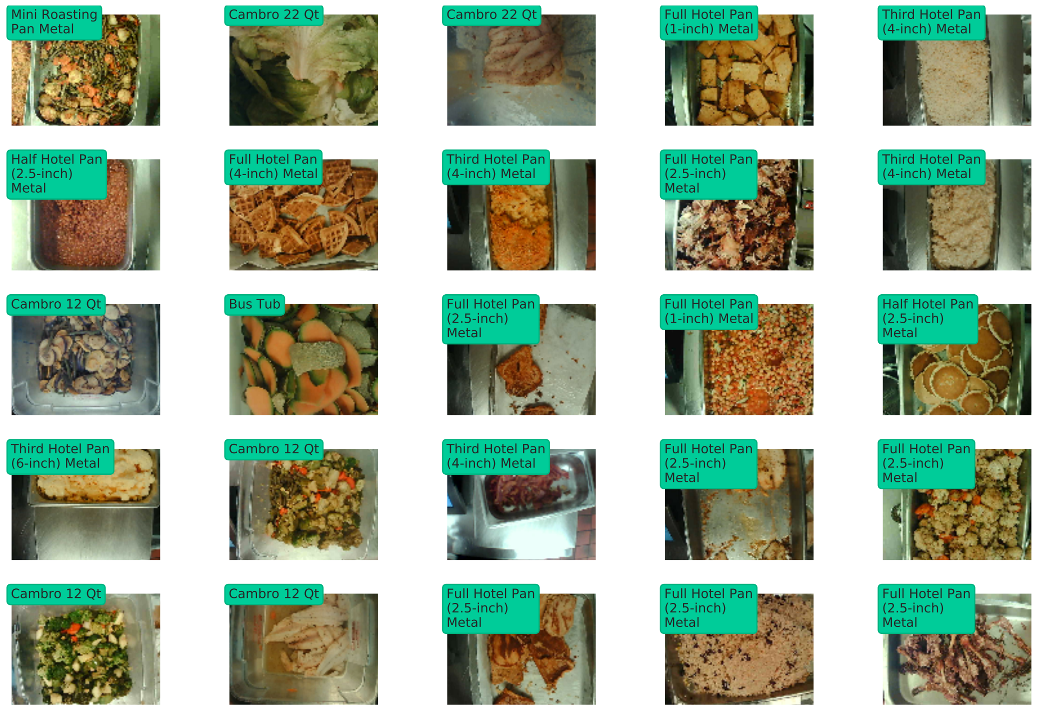


(a) Random Samples from Waste Item Training Set

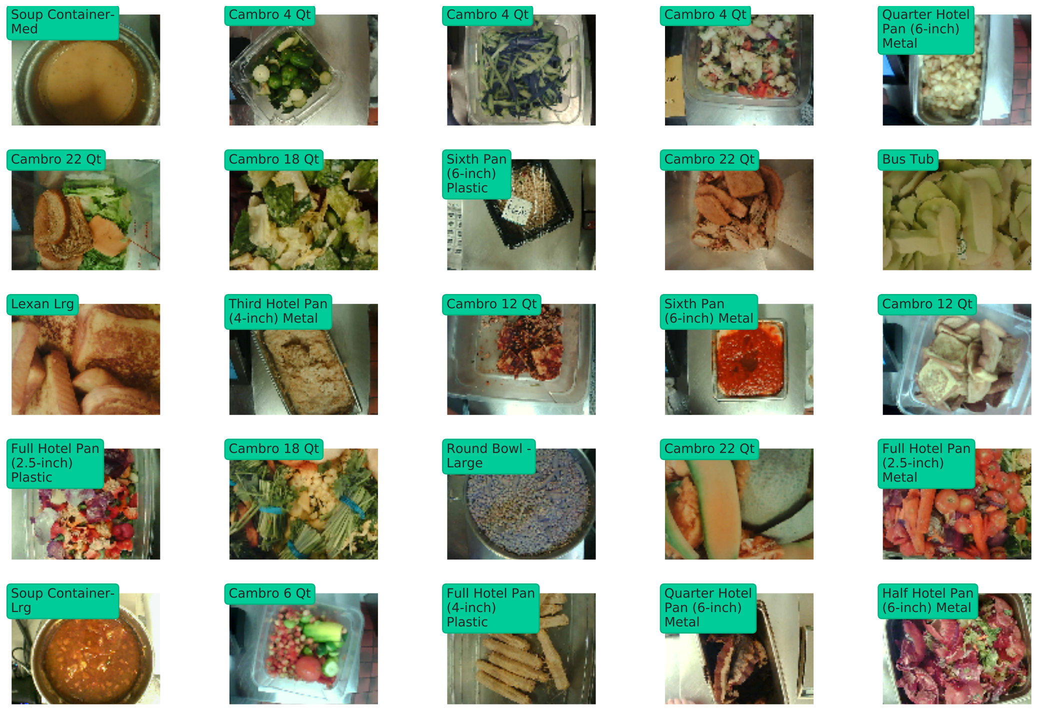


(b) Random Samples from Waste Item Test Set

Figure 14: Randomly Selected Waste Item Samples


Mini Roasting Pan Metal
Cambro 22 Qt
Cambro 22 Qt
Full Hotel Pan (1-inch) Metal
Third Hotel Pan (4-inch) Metal
Half Hotel Pan (2.5-inch) Metal
Full Hotel Pan (4-inch) Metal
Third Hotel Pan (4-inch) Metal
Full Hotel Pan (2.5-inch) Metal
Third Hotel Pan (4-inch) Metal
Cambro 12 Qt
Bus Tub
Full Hotel Pan (2.5-inch) Metal
Full Hotel Pan (1-inch) Metal
Half Hotel Pan (2.5-inch) Metal
Third Hotel Pan (6-inch) Metal
Cambro 12 Qt
Third Hotel Pan (4-inch) Metal
Full Hotel Pan (2.5-inch) Metal
Full Hotel Pan (2.5-inch) Metal
Cambro 12 Qt
Cambro 12 Qt
Full Hotel Pan (2.5-inch) Metal
Full Hotel Pan (2.5-inch) Metal
Full Hotel Pan (2.5-inch) Metal

(a) Random Samples from Container Training Set


Soup Container-Med
Cambro 4 Qt
Cambro 4 Qt
Cambro 4 Qt
Quarter Hotel Pan (6-inch) Metal
Cambro 22 Qt
Cambro 18 Qt
Sixth Pan (6-inch) Plastic
Cambro 22 Qt
Bus Tub
Lexan Lrg
Third Hotel Pan (4-inch) Metal
Cambro 12 Qt
Sixth Pan (6-inch) Metal
Cambro 12 Qt
Full Hotel Pan (2.5-inch) Plastic
Cambro 18 Qt
Round Bowl - Large
Cambro 22 Qt
Full Hotel Pan (2.5-inch) Metal
Soup Container-Lrg
Cambro 6 Qt
Full Hotel Pan (4-inch) Plastic
Quarter Hotel Pan (6-inch) Metal
Half Hotel Pan (6-inch) Metal

(b) Random Samples from Container Test Set

Figure 15: Randomly Selected Container Samples

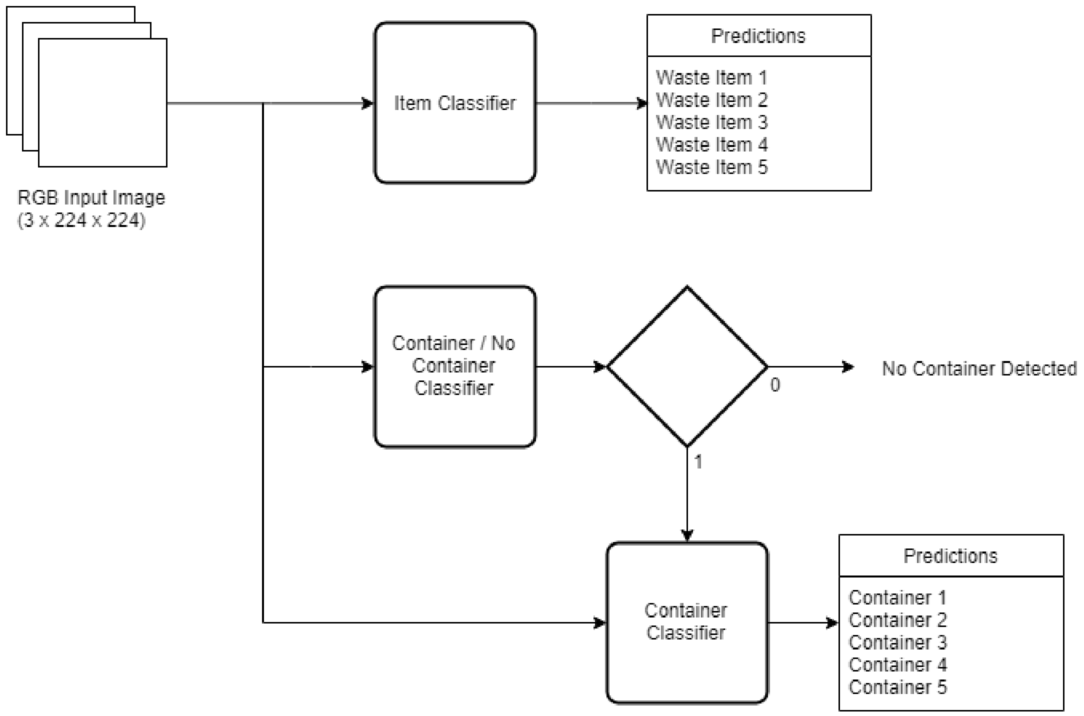


Figure 16: Waste Item and Container Classification Process

resenting a RGB image. In this case, the first dimension of the tensor corresponds to the image channel (red, green, blue), the second dimension corresponds to the height of each channel, and the third dimension corresponds to the depth of each channel. In this thesis, CNNs are used to both detect features and produce probabilities for each class. Therefore, the output of the network is an array of normalized probabilities, representing the prediction score for each class. Network architecture is described in Table 1, where each stage in the network is described in terms of operations, input resolution, the number of channels, and the number of layers in each stage. The MBConv layers are *mobile inverted bottlenecks* as described in Section 3.5.3.

4.2.1 Preprocessing

In order to prepare the images for training and testing, the metadata for each set was handled by a custom Dataset class, written to simply open an image from disk given the index in the data frame. The indices were selected by PyTorch's RandomSampler, which samples randomly without replacement for a given batch size in each epoch.

Next, data augmentation through a series of torchvision transforms was performed on the training images. Data augmentation uses training data to create synthetic data, a powerful tool to help mitigate over-fitting while still allowing the network to learn to meaningful features. The choice in suitable transformations is very data dependent, so there are no concrete rules applied to any given dataset. A few tips can be provided here, however. For example, if the object of interest may be presented at any given angle, it is a good idea to apply a transformation which randomly rotates the image or provides a perspective shift. If the object may be presented at different orientations, it is sometimes a good idea to perform a vertical or horizontal flip. Another case is variation of brightness, hue, and saturation —there are transforms which can augment images to simulate these as well.

For the training data in this thesis, the first transform used is the RandomResizedCrop, an image sampling method which was shown to be successful in leading Google's Inception network architecture to convergence in training on ImageNet [47]. RandomResizedCrop randomly crops a section of the image ranging from 0.08 to 1.0 times the original size, with a random aspect ratio ranging from 3/4 to 4/3. The crop is then resized, in this case to $(224, 224)$, which is the input size of the network. Next, to help introduce invariance to orientation of the subject, the image is randomly flipped horizontally with probability $p = 0.5$ using RandomHorizontalFlip. This is followed by a RandomPerspective, using default settings, which distorts the image's perspective with probability $p = 0.5$. While there are many other transforms available which can be used to create larger sets of synthetic data (such as RandomVerticalFlip to flip vertically or RandomJitter to adjust brightness, hue, and contrast), the images in the dataset are varied with respect to rotation and placement already. Applying random crops and flips build on this variation. Only the aforementioned transforms were chosen for the experiments in this thesis, but it is possible in future cases that it may be beneficial to vary features such as color space values to account for differences in brightness, saturation, and so on, which sometimes result from camera malfunction. Finally, the image is converted from a PIL image to a tensor, and normalized with the mean and standard deviation from ImageNet. Using ImageNet statistics is standard practice for fine-tuning on models pretrained with ImageNet, as much of the learning has already been performed on the dataset, which consists of roughly 14 million examples. The datasets used in this thesis are much smaller in comparison. An example of the transformed training data is shown in Figure 17.

Sometimes, augmentation is performed on test data to boost accuracy in prediction. One form of this is called Test Time Augmentation (TTA). A common augmentation is the n-crop testing, which splits an example into n image patches and averages the prediction over each patch. TTA is not performed in these experiments, to keep the test performance as close as possible to actual performance on the real image. For testing in this thesis, each image is simply downsampled to $(224, 224)$ and normalized with the ImageNet mean and standard deviation.

4.2.2 Hyperparameters, Optimizer, and Loss Function

Following the advice of Goodfellow, et al. [8],

> The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate. (p. 417)

Grid search was performed to select the learning rate, learning rate schedule, and mini-batch size for the waste item and container classification models. The learning rate schedule is included in the search because it directly affects the learning rate and allows us to quickly descend to minima in the loss landscape, while reducing the step size over time to help prevent overstepping a minima. The batch size was also included in the search, as it has been suggested in [48] that this should also be optimized due to the slight interactions between this hyperparameter and learning rate. For weight decay (the L2-regularization hyperparameter), a small value of $\lambda = 0.0001$ was used to aid in regularization. Values for this hyperparameter vary depending on the network examined. The EfficientNet paper [25] follows the procedure used by Kornblith, et al. [49] for fine-tuning, which establishes a search space of 7 points, logarithmically spaced from 10^{-6} to 10^{-3} and including zero. The best found parameters are not provided. Due to this ambiguity, and to reduce the computational resources required to run grid search, weight decay was simply chosen within the suggested search space of [49] and kept constant.

The Cross Entropy loss function is commonly used to judge the performance of classification models. Its popularity is due to its equivalence to Maximum Likelihood Estimation, which can be shown to be the best estimator asymptotically and is a consistent estimator under certain conditions [8]. It is considered the default loss function for classification. For these reasons, it is also used as a parameter estimator in this work, as we characterize the performance surface of the network with respect to a set of parameters with this loss function. For optimization, Stochastic Gradient Descent was used with Nesterov Momentum = 0.9. Each model was fine-tuned for 30 epochs.

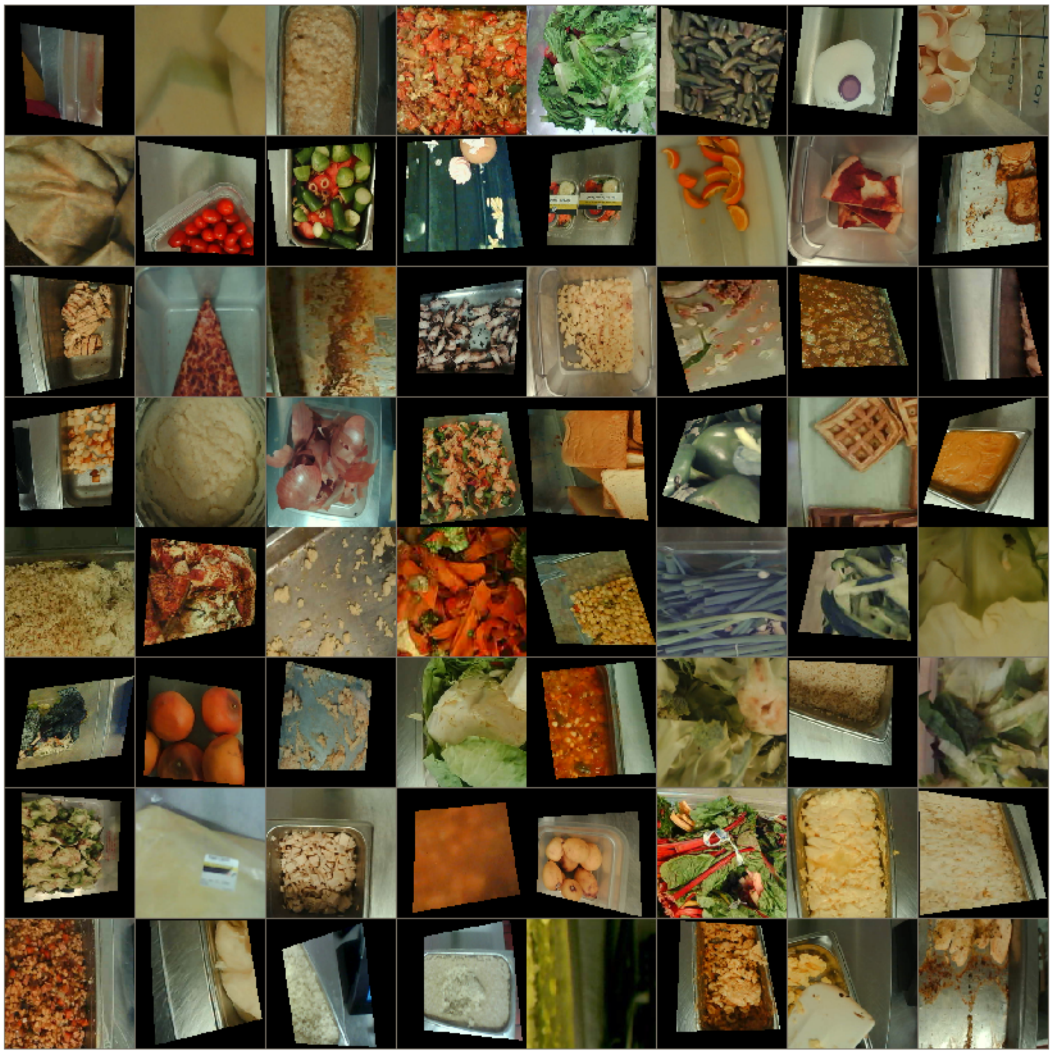


Figure 17: Example Transforms

5 Experimental Design

5.1 Design Constraints

5.1.1 Ground Truth Accuracy

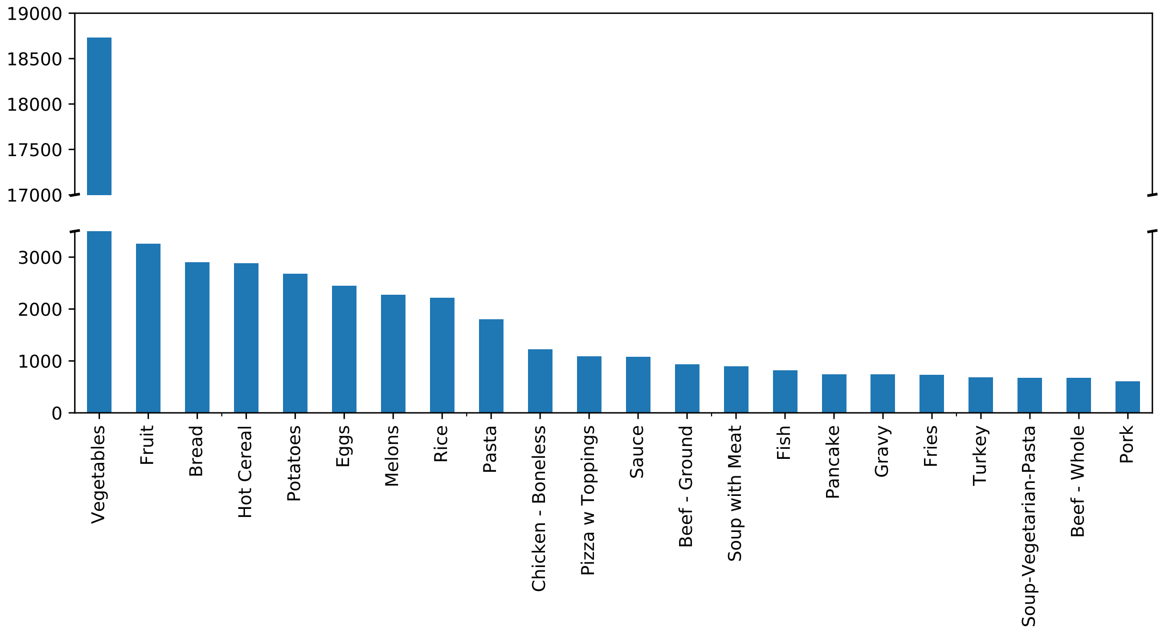
One of the biggest challenges in training a model on each dataset has been making decisions regarding class noise. For the food waste item dataset, categories such as mixed plate, mixed protein, and mixed starch are heavily utilized by the end user, but appear to have strayed from their intended uses. For example, mixed plate may have been intended to be used for cafeteria-style plates with mixed food items, but many photos contain examples from every category along with mixed plates and containers with mixed food items. The same problems are present in the other mixed categories. By inspection, the sheer amount of noise in these categories outweigh the usable examples, so these classes were removed from training and testing. It is possible that with heavy cleaning, the mixed classes may be reinstated, but this task is left for future work.

The container data has similar problems with class noise across all categories, but were chronic with respect to the No Pan, Cambro, and Lexan classes. The No Pan category had many examples with both containers and without containers, in addition to ambiguous cases such as styrofoam, dishes, and salad containers. Roughly 50% of the test sample could be considered as having a pan. Due to the sheer amount of class noise, this category was removed and individually sorted into its own dataset such that only official kitchen containers and plates would be considered a container. It is possible that this interpretation does not fit that of every customer site. For the

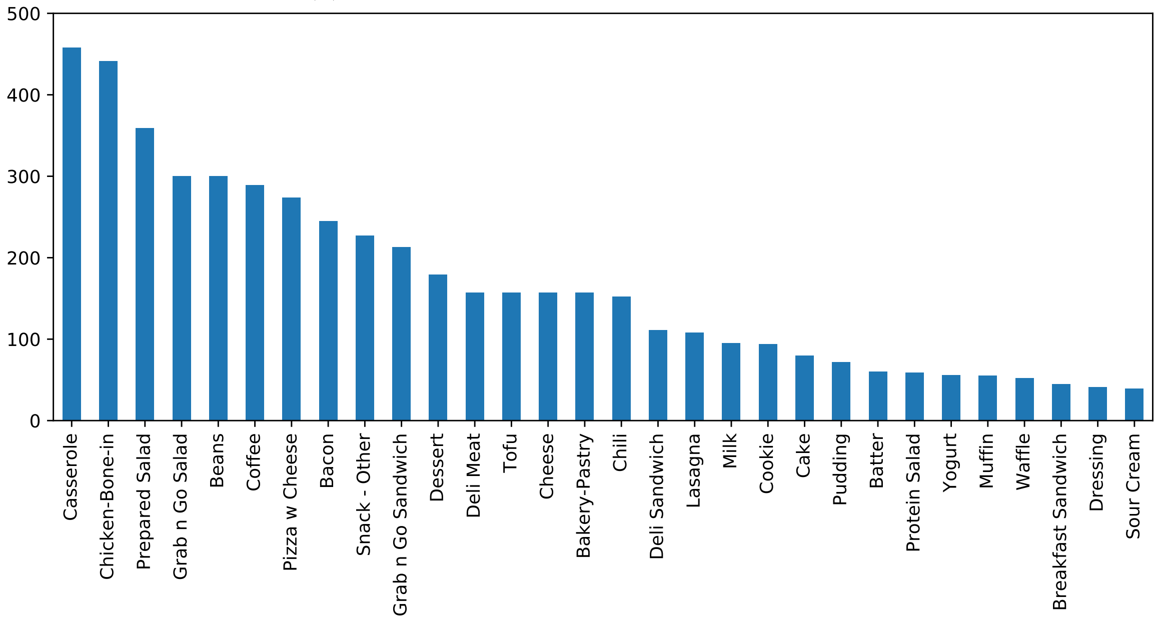
Cambro and Lexan categories, it is unclear at times which container truly belongs to which class. Cambro is a brand name of container, and Lexan is a type of plastic. It is possible that a Cambro may be made of Lexan, but the converse may not be true. Containers designated as a type of Cambro made up a more significant portion of the overall population compared to Lexan. For this reason, a decision was made to remove Lexan from the dataset. It is possible that not all end users would agree with this choice, so results may vary across sites.

5.1.2 Class Imbalance

Another challenge presented by this dataset is related to imbalance between class sizes. This is particularly noticeable in the waste item dataset, in which the largest class (vegetables) outnumbers the second largest class (fruit) by more than a factor of 5. The smallest class included (sour cream) is outnumbered by a factor of 300. This is not the worst imbalance, as the original metadata contained some classes which had less than 10 examples. As the data was originally pulled over all of 2018 from four sites, this points to very low usage on these categories. By not including very low represented classes, this means that the end user will have to manually classify these events as they occur. It is possible that this could be remedied by pulling data across all customer sites, but this is left for future work.



(a) Waste Item Dataset Classes, Size > 500



(b) Waste Item Dataset Classes, Size < 500

Figure 18: Waste Item Dataset Class Sample Sizes

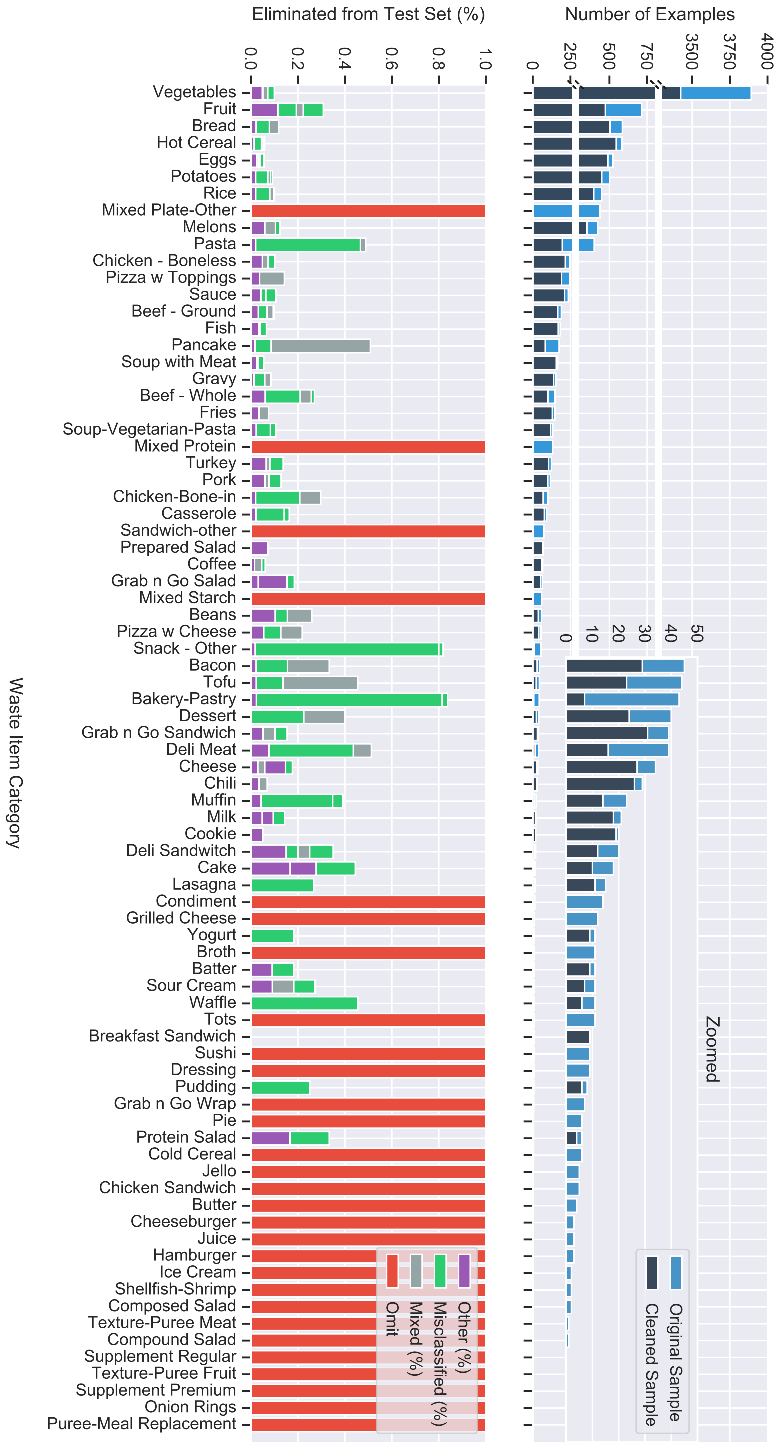


Figure 19: Left: % Eliminated from Test Set, with Reason. Right: Number of Examples in Each Category.

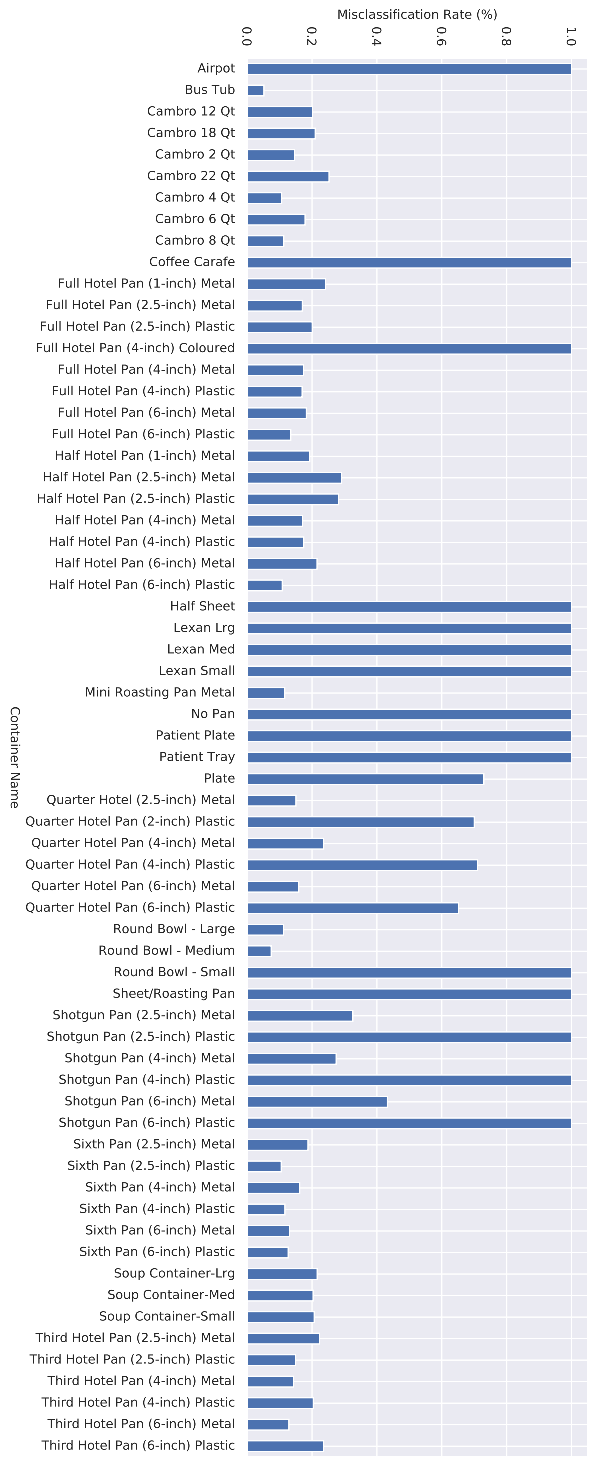


Figure 20: Left: % Eliminated from Test Set, with Reason. Right: Number of Examples in Each Category.

5.2 Experiments

For the waste item and container datasets, the original test partitions created in the train/test splits were split into dev/test partitions. Grid search was performed on the train/dev sets in order to obtain the best learning rate, batch size, and learing rate decay to reasonably achieve the performance goal. The search varied learning rate over the interval $\eta = [0.1, 0.01, 0.001]$, learning rate decay by a factor of 10 in steps of $s = [7, 10, 15]$, and batch size $\mathcal{B}_m$ for $m = [16, 32, 64]$. To begin, the train/dev sets were concatenated and shuffled, and then split into 5 folds. For each combination of hyperparameters, a model was trained for 30 epochs. The 30 epoch limit was selected to encourage choosing hyperparameters which lead to high accuracy in a short period of time. In a typical k-fold cross validation, the search would have been repeated across each combination of folds. Due to the computational resources required to perform cross validation, the search was not repeated across folds. The same hyperparameters found through searching the waste item dataset were also used in training the Location X model. Similarly, the hyperparameters found searching over the container dataset were also used in training the container/no container and grouped container models. This was another choice which was made to conserve computational resources. The images are similar in content, so this choice proved reasonable to achieve performance criteria, but it is possible that further optimization can lead to better results.

Once the most suitable hyperparameters were discovered through grid search, the train/dev partitions were concatenated and mixed together to form a single training partition. The final models were obtained by training on their respective partitions over 30 epochs and evaluating on the test set. The test scores in Table 3 are the best results obtained, but not necessarily at epoch 30.

To evaluate the waste item model's capability as a feature extractor, the convolutional layers fine-tuned for waste item classifier were used on the smaller Location X dataset. When evaluating as a feature extractor, this simply means that the convolutional layers of the network are frozen such that they cannot learn new features and the weights cannot change. Classification is still performed by the network, but the previous classification layer is replaced with a new one with randomly initialized weights. By freezing these layers, the model is only allowed to detect features learned through fine-tuning on the waste item dataset. This is also benchmarked against a model which was only pretrained on ImageNet, to highlight the importance of fine tuning a model on data which is more closely related to the problem of interest.

To measure performance of each classifier, the top-1 accuracy, top-5 accuracy, micro-averaged precision-recall Curve, and F1-Score were measured. The top-5 accuracy is considered the main measure of performance due to the UI design of the product for which this is intended for implementation. However, due to class imbalance, only considering accuracy of a classifier can marginalize the perception of performance on small classes. For this reason, precision-recall curves and F1-Scores are also considered. Precision is important because it provides a metric for understanding how many items of a specific class were correctly classified out of all predictions for that class. Recall is important because it provides a measure of how often the classifier attempts to predict a class of all labeled examples of that class in the dataset. The precision-recall curve allows for visualizing how the two metrics interact over different thresholds, and can be summarized with the average precision which is simply the area under the curve. The F1-Score allows us to calculate the harmonic mean of precision and recall, thus providing a summarizing statistic. More detail on these metrics is provided in Section 3.7. To examine anomalies and edge cases, normalized confusion matrices are also considered. The confusion matrix is especially helpful for insight into misclassifications and understanding where the model has a tendency to classify. For better understanding model attention for a specific example, the Class Activation Maps [43] are also examined. The code for generating these maps can be found in [50] and easily adapted to a custom PyTorch model. Finally, to show the power of fine-tuning a pretrained feature extractor on the custom dataset, these metrics will be compared to a classifer which only utilizes ImageNet weights.

All experiments were conducted on Google Cloud Platform, using JupyterLab Notebooks with Python 3.7 kernels. Google Cloud Platform (along with other cloud services) provides access to additional resources in remote server farms, which allow running experiments with larger batch sizes at higher speeds. This enables access to high end computing resources, for a small fee, without having to commit to purchasing expensive deep learning "rigs" with dedicated GPUs. An advantage to using Jupyter notebooks is that they make model prototyping and collaboration easy. Once the model architecture and training scripts are finalized, they can also be easily converted into a single python script for automation and retraining on future data. For building and training neural networks, the PyTorch 1.1 framework was used, although there are EfficientNet implementations in other frameworks, such as Tensorflow/Keras. Choice of framework is very much user preference. A standard compute instance was built with 8 vCPUs and 52 GB RAM, which was often more than sufficient to train each model. A NVIDIA P100 GPU was used to train each model due to its compromise between speed, memory (16 GB) size, and price.

6 Results and Discussion

Final results from evaluating each model on the respective test partitions are shown in Table 3. The following sections will describe the strengths and deficiencies for each model, along with the top-1 accuracy, top-5 accuracy, average precision, and F1-score. For justification of these metrics, please see Section 5.2.

Dataset	**Top-1 Test Acc. (%)**	**Top-5 Test Acc. (%)**	**AP**	**F1-Score**
Item	90.3	98.8	0.96	0.90
Container/No Container	96.7	N/A	0.97	0.98/0.94
Container	70.2	94.6	0.77	0.70
Grouped Container	84.6	98.6	0.91	0.84
Location X	81.6	96.3	0.88	0.80

Table 3: Dataset Test Results

6.1 Waste Item Dataset

6.1.1 Hyperparameter Grid Search

The results from performing grid search on the waste item dataset are shown in Table 4. Performance was judged in terms of highest top-5 accuracy on the validation partition. While there is a tie for the top-5 accuracy score (95.425%), referring to the top-1 accuracy as a tie-breaker shows a clear winner. As a result, a mini-batch size of $\mathcal{B}_m = 64$, learning rate of 0.01, and step decay of a factor of 10 every 15 epochs was selected for final model training.

$\mathcal{B}_m$	η	s	**Top-1 Validation Accuracy (%)**	**Top-5 Validation Accuracy (%)**
16	0.001	7	80.376	94.203
16	0.001	10	81.187	94.674
16	0.001	15	81.488	94.844
16	0.01	7	75.541	95.054
16	0.01	10	75.961	93.622
16	0.01	15	75.921	93.442
16	0.1	7	70.835	90.378
16	0.1	10	72.026	90.378
16	0.1	15	70.455	90.378
32	0.001	7	79.626	93.712
32	0.001	10	80.687	94.273
32	0.001	15	81.318	94.724
32	0.01	7	82.859	95.344
32	0.01	10	82.689	95.425
32	0.01	15	82.699	95.374
32	0.1	7	77.753	93.602
32	0.1	10	79.575	94.203
32	0.1	15	78.314	94.023
64	0.001	7	77.423	92.781
64	0.001	10	78.474	93.402
64	0.001	15	80.437	94.163
64	0.01	7	82.379	95.124
64	0.01	10	82.679	95.404
64	**0.01**	**15**	**82.859**	**95.425**
64	0.1	7	81.878	95.054
64	0.1	10	82.329	95.244
64	0.1	15	81.698	94.984

Table 4: Waste Item Grid Search Cross Validation Results

6.1.2 Train and Test Results

After training the model on the combined training and validation partitions with hyperparameters from 6.1.1, the model was evaluated on the test partition. A random subset of example predictions can be found in Figure 21 The accuracy and loss for each epoch for the waste item dataset is shown in Figure 22. Notice that the training was run for 30 epochs, and the best top-5 accuracy was obtained at epoch 21. For consistency, each experiment was stopped at epoch 30. It is possible that slightly higher accuracy could be achieved by running longer, but due to the learning rate annealing schedule, the returns from doing so would be diminishing.

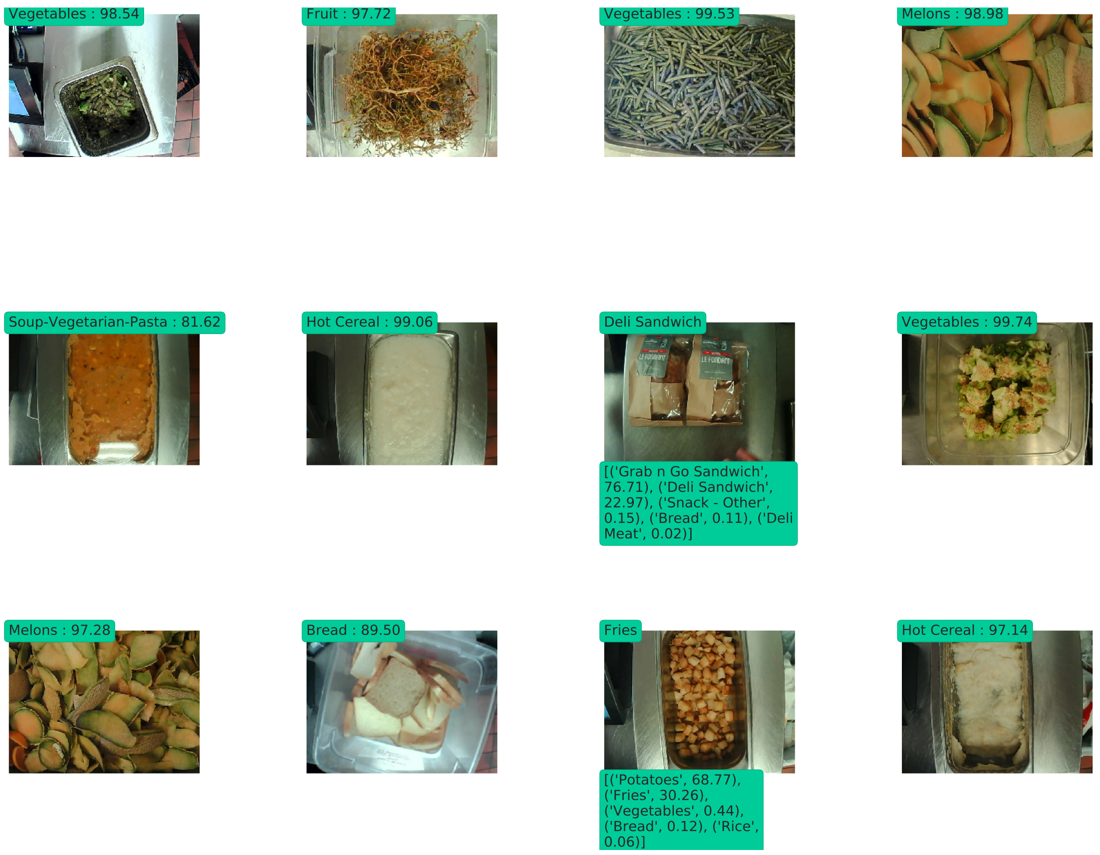

Vegetables : 98.54
Fruit : 97.72
Vegetables : 99.53
Melons : 98.98
Soup-Vegetarian-Pasta : 81.62
Hot Cereal : 99.06
Deli Sandwich
[('Grab n Go Sandwich', 76.71), ('Deli Sandwich', 22.97), ('Snack - Other', 0.15), ('Bread', 0.11), ('Deli Meat', 0.02)]
Vegetables : 99.74
Melons : 97.28
Bread : 89.50
Fries
[('Potatoes', 68.77), ('Fries', 30.26), ('Vegetables', 0.44), ('Bread', 0.12), ('Rice', 0.06)]
Hot Cereal : 97.14

Figure 21: Waste Item Classifier Random Sample Predictions

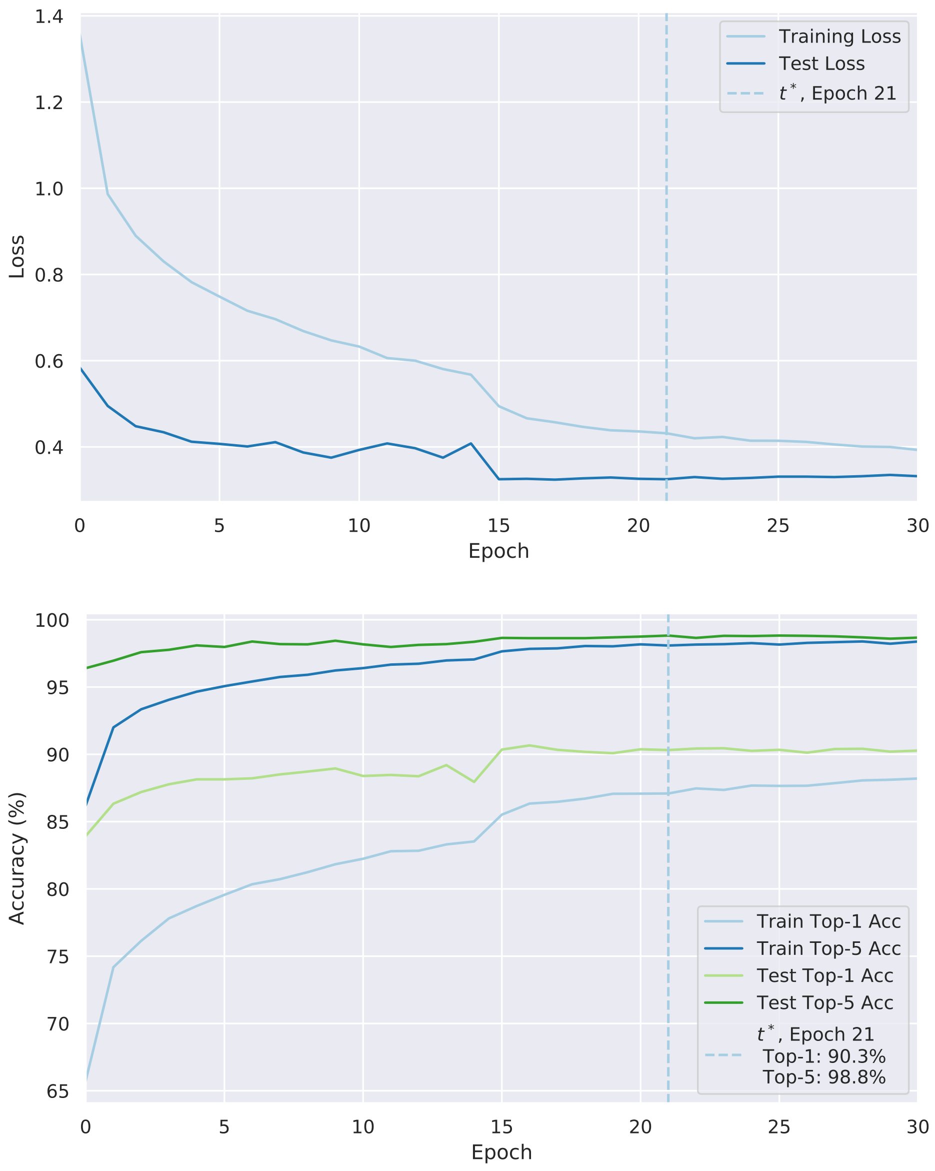


Figure 22: Waste Item Model Performance. Loss, Accuracy versus Epoch

The precision-recall curve in Figure 23 shows the micro-averaged precision vs recall for the corresponding waste item classes, evaluated on the test set, as defined in Section 3.7. Recall that a naive classifier would be represented by a horizontal line at $y = 0.5$. A perfect classifier would have an average precision of 1.0. The waste item classifier shows high precision and recall up until roughly $x = 0.95$. The average precision is 0.96.

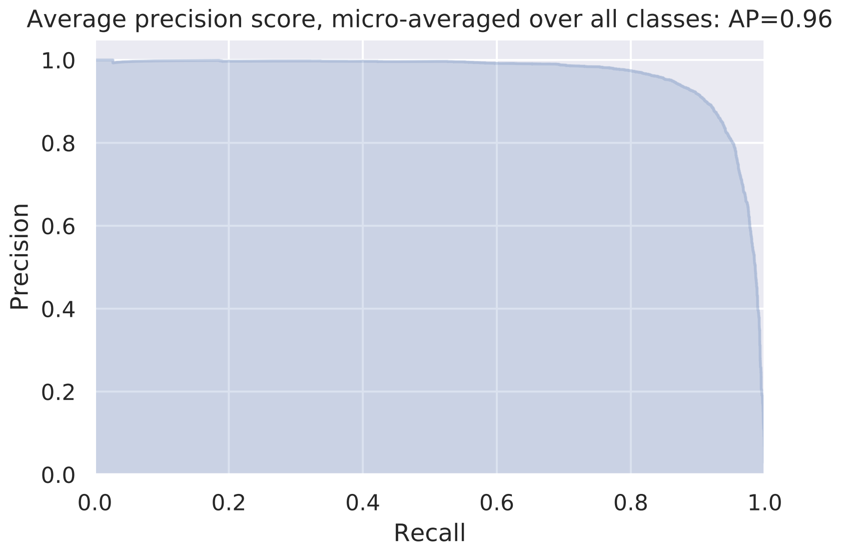


Figure 23: Waste Item Classifier Precision-Recall Curve

The classifier scored well in terms of AP and accuracy, but for multi-class scenarios, the confusion matrix is very helpful for visualizing individual class performance. For the waste item dataset, the *normalized confusion matrix* is shown in Figure 24. This is similar in spirit to the typical confusion matrix, but normalizes each class by its size to range from 0 to 1 instead of the population counts for ease of visualization. A perfect classifier would have only values of 1.0 on the diagonal, and 0 elsewhere.

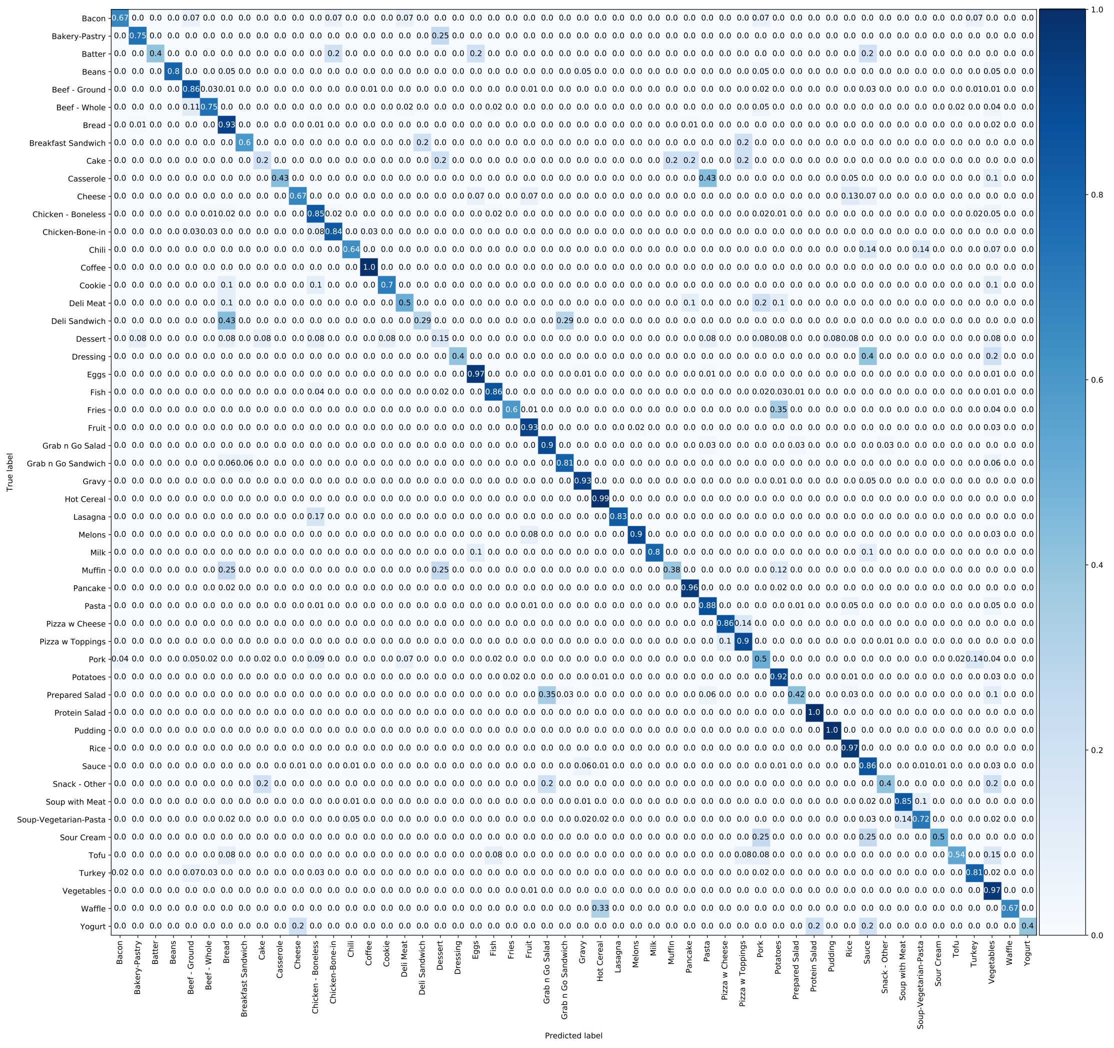
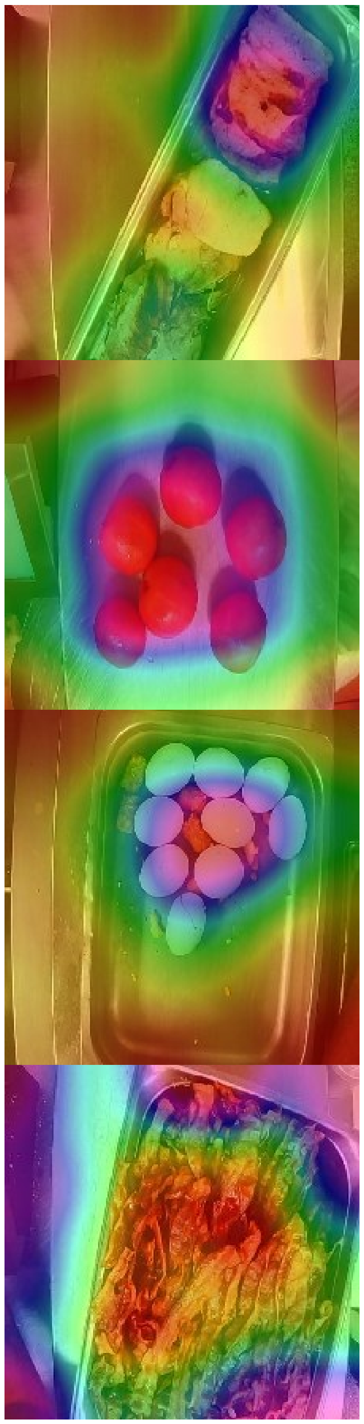


Figure 24: Waste Item Confusion Matrix

Some interesting information is encoded in the confusion matrix. For example, the predictions for the cake category are distributed equally among cake, dessert, muffin, pancake, and pizza with toppings classes. This could mean that there are examples with similar features present in each of the training categories, and may warrant further exploration. For the purposes of this application, however, cake should be among the top 5 predictions, so this may still be satisfactory for the product. Similarly, predictions for the casserole class are split between casserole, pasta, rice, and vegetables. It is likely that there is some ambiguity between theses classes, depending on the mixture of ingredients and operator classification at the time of the waste event. Some other categories show very strong performance, such as: eggs, rice, hot cereal, protein salad, pudding, rice, and vegetables. The classifier is able to assign over 97% of available examples to the true classes in these examples. Note that some of these are classes for which more training data is available for.

Finally, we can examine the Grad-CAM visualizations for a few of the classes. Figure 25 shows the class activation maps for examples of pork, vegetables, eggs, and bacon. Notice that in this example, all of top-1 predictions align with the true class. For pork, the features which contribute significantly to this prediction seem to be localized in the upper right cluster of meat in the image, while the other slices of pork do not factor into the decision as much. The image of vegetables shows clear localization of the tomatoes, as with the eggs. The prediction for bacon, however, clearly hinges on the corners of the container it is in. This could be due to the fact that for the collection of customer sites used to build this dataset, bacon is commonly in this type of pan. In order to help the classifier learn to truly recognize the texture and shape of bacon, it may be necessary to introduce additional examples from other environments.



True Class: **Pork**

Top 5 Predictions:

Pork 65.8%
Beef-Whole 20.7%
Chicken-Boneless 6.3%
Turkey 6.3%
Beef-Ground 0.01%

True Class: **Vegetables**

Top 5 Predictions:

Vegetables 99.9%
Eggs 0.01%
Fruit 0.0%
Beans 0.0%
Bread 0.0%

True Class: **Eggs**

Top 5 Predictions:

Eggs 96.2%
Chicken-Boneless 3.3%
Snack-Other 2.7%
Vegetables 0.1%
Tofu 0%

True Class: **Bacon**

Top 5 Predictions:

Bacon 99.3%
Pork 0.7%
Vegetables 0%
Pasta 0%
Lasagna 0%

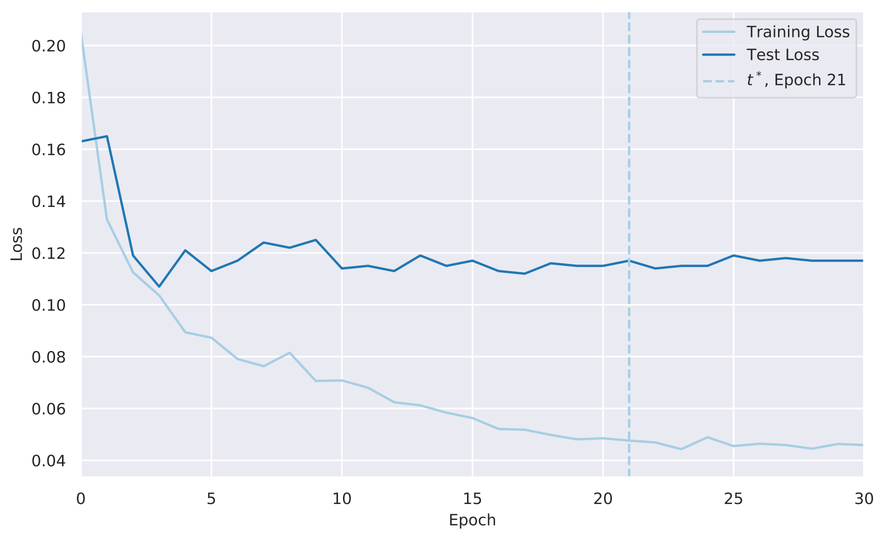
Figure 25: Selected Grad-CAM Visualizations for Food Waste Model

6.2 Container/No Container Dataset

For the container/No container model, Loss and Accuracy vs Epoch performance is shown in Figure 26. Note that as this is a binary classifier, only top-1 accuracy is considered. At epoch 21, the classifier was able to achieve maximum classification accuracy of 96.6%. The corresponding precision-recall curve and normalized confusion matrix can be found in Figure 27. The PR curve shows high skill for this classifier for all values or recall, with an average precision of 0.97.

One of the challenges in building this dataset is determining what constitutes a container. Some clear members of this class include the Cambro, Bus Tub, and Hotel Pan types. However, should a plastic bag, napkin, or a styrofoam container be considered a pan? Since these are typically of negligible weight, they were considered to fall under the “no container” class. In addition, when there are no container edges present in an example (such as a close up image of food in a bus tub), this is also considered to not have container.

Given the above assumptions, this classifier is significantly better than manual user entry. Based on examining the test partition prior to building this dataset, user error was approximately 50%. Of course, this will vary depending on the site evaluated. It is possible that classification accuracy could be improved further with the collection of more data.



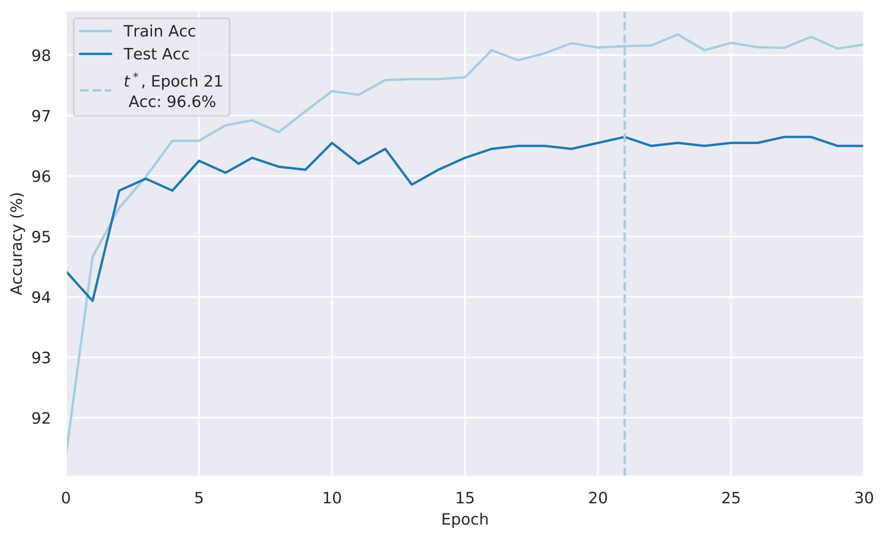
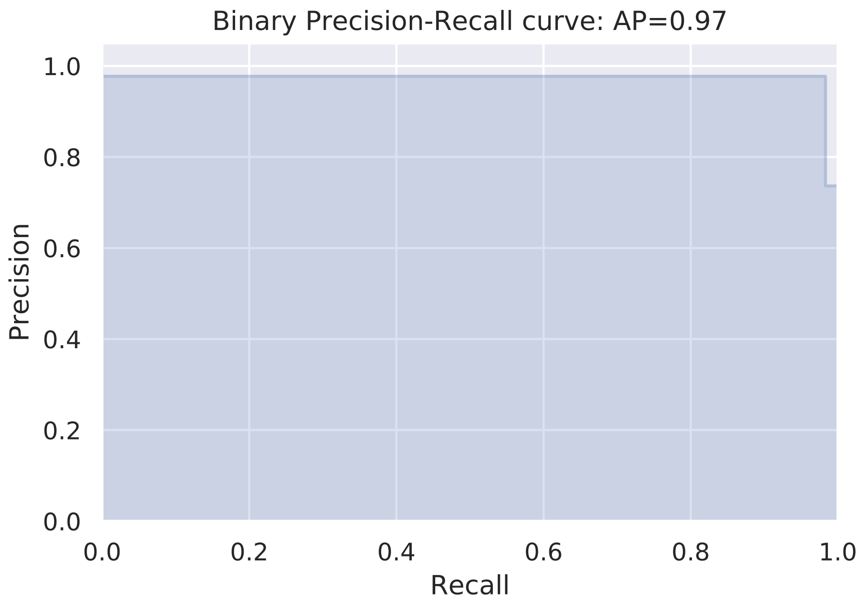


Figure 26: Container/No Container Model Performance. Loss, Accuracy versus Epoch



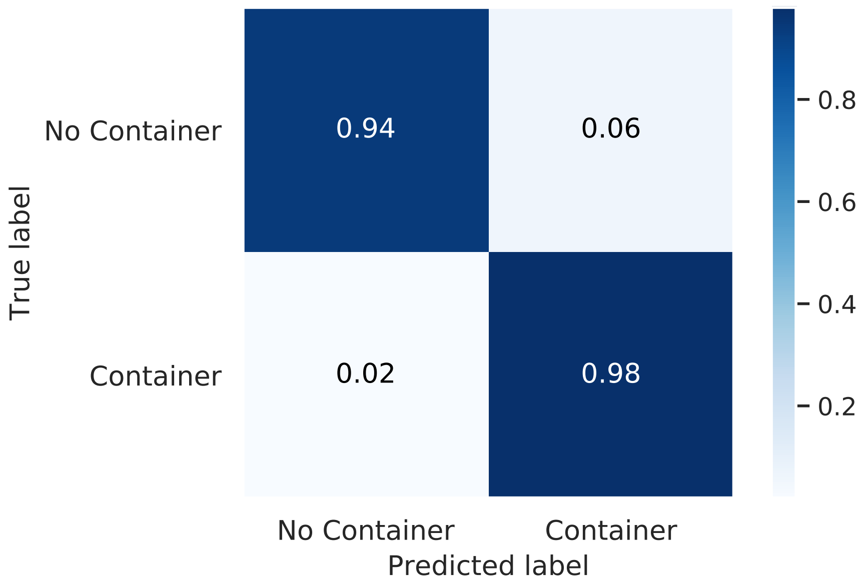


Figure 27: Binary Container/No Container Model PR Curve and Confusion Matrix

6.3 Container Dataset

6.3.1 Hyperparameter Grid Search

The results from performing grid search on the container dataset are shown in Table 5. The search space was narrowed to exclude a smaller mini-batch size of 16, unlike the waste item experiment which included this value. Performance was judged in terms of highest top-5 accuracy on the validation partition. Through this experiment, a mini-batch size of $\mathcal{B}_m = 64$, learning rate of 0.1, and step decay of a factor of 10 every 10 epochs was selected for final model training.

B_m	η	s	**Top-1 Validation Accuracy (%)**	**Top-5 Validation Accuracy (%)**
32	0.001	7	53.374	79.134
32	0.001	10	53.949	80.237
32	0.001	15	56.015	80.941
32	0.01	7	57.329	82.044
32	0.01	10	57.435	82.138
32	0.01	15	58.209	82.232
32	0.1	7	53.116	79.216
32	0.1	10	54.360	80.190
32	0.1	15	55.616	80.554
64	0.001	7	50.158	76.622
64	0.001	10	52.165	78.183
64	0.001	15	53.656	79.392
64	0.01	7	57.212	81.810
64	0.01	10	57.282	82.244
64	0.01	15	58.139	82.244
64	0.1	7	57.622	82.361
64	**0.1**	**10**	**58.197**	**82.385**
64	0.1	15	57.446	82.349

Table 5: Container Cross Validation Results

6.3.2 Train and Test Results

The train and test curves for the container classifier can be viewed in Figure 29. A random sample of predictions can be found in Figure 28. The maximum Top-5 classification accuracy obtained for this model was only 95.2%, which is lower than the goal of 99%. Top-1 classification accuracy was also low at 70.2%. For perspective, there is clearly learning occuring here for most categories, as a “no skill” classifier would only have a Top-1 accuracy of 2%. The PR curve along with average precision of this model is shown in Figure 30. The average precision is 0.77, which is still higher than 0.50, but much lower than the previous two models listed. While generally better than a naive classifier, the performance still does not meet the standards of this thesis.

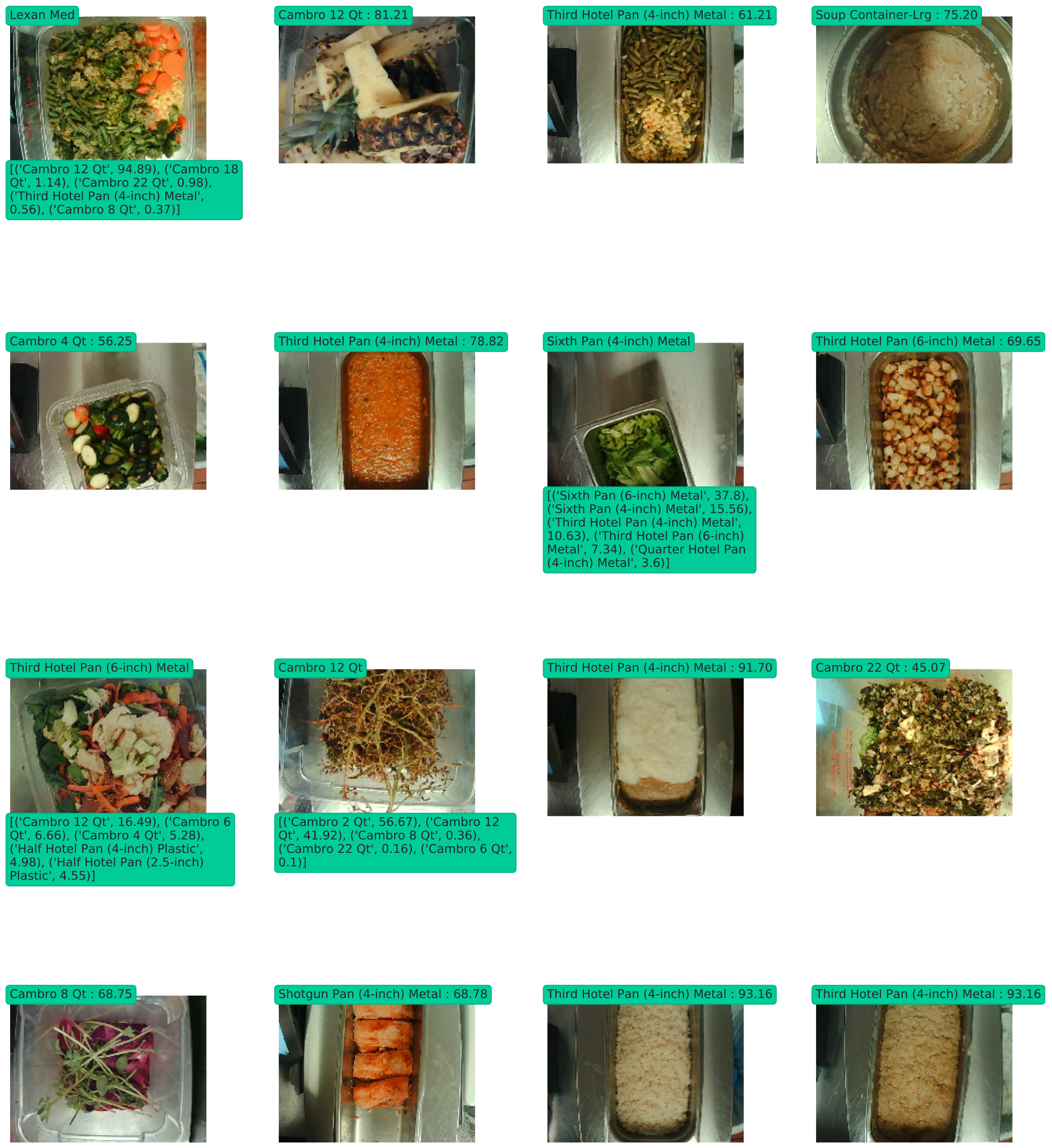
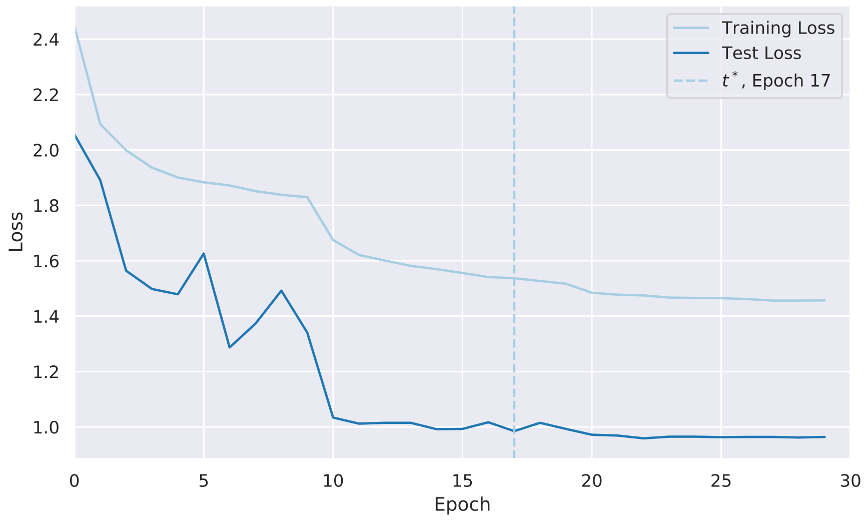

Lexan Med
[('Cambro 12 Qt', 94.89), ('Cambro 18 Qt', 1.14), ('Cambro 22 Qt', 0.98), ('Third Hotel Pan (4-inch) Metal', 0.56), ('Cambro 8 Qt', 0.37)]
Cambro 12 Qt : 81.21
Third Hotel Pan (4-inch) Metal : 61.21
Soup Container-Lrg : 75.20
Cambro 4 Qt : 56.25
Third Hotel Pan (4-inch) Metal : 78.82
Sixth Pan (4-inch) Metal
[('Sixth Pan (6-inch) Metal', 37.8), ('Sixth Pan (4-inch) Metal', 15.56), ('Third Hotel Pan (4-inch) Metal', 10.63), ('Third Hotel Pan (6-inch) Metal', 7.34), ('Quarter Hotel Pan (4-inch) Metal', 3.6)]
Third Hotel Pan (6-inch) Metal : 69.65
Third Hotel Pan (6-inch) Metal
[('Cambro 12 Qt', 16.49), ('Cambro 6 Qt', 6.66), ('Cambro 4 Qt', 5.28), ('Half Hotel Pan (4-inch) Plastic', 4.98), ('Half Hotel Pan (2.5-inch) Plastic', 4.55)]
Cambro 12 Qt
[('Cambro 2 Qt', 56.67), ('Cambro 12 Qt', 41.92), ('Cambro 8 Qt', 0.36), ('Cambro 22 Qt', 0.16), ('Cambro 6 Qt', 0.1)]
Third Hotel Pan (4-inch) Metal : 91.70
Cambro 22 Qt : 45.07
Cambro 8 Qt : 68.75
Shotgun Pan (4-inch) Metal : 68.78
Third Hotel Pan (4-inch) Metal : 93.16
Third Hotel Pan (4-inch) Metal : 93.16

Figure 28: Container Classifier Random Sample Predictions



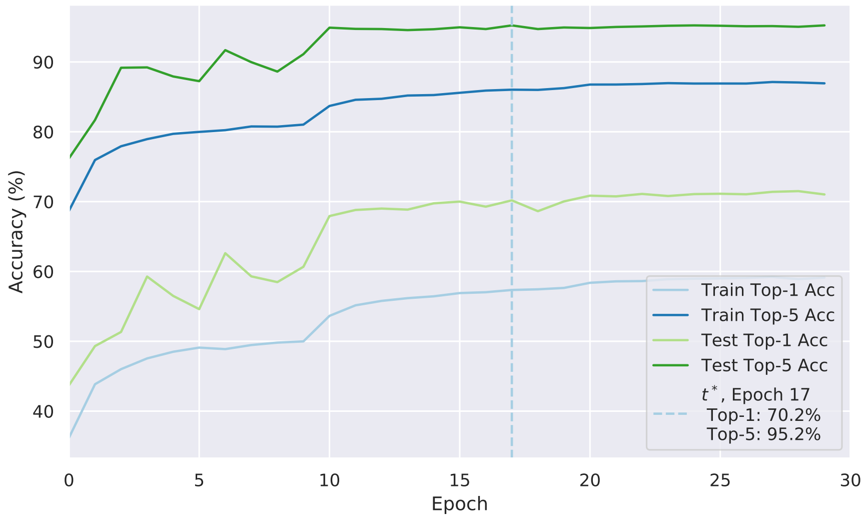


Figure 29: Container Model Performance. Loss, Accuracy versus Epoch.

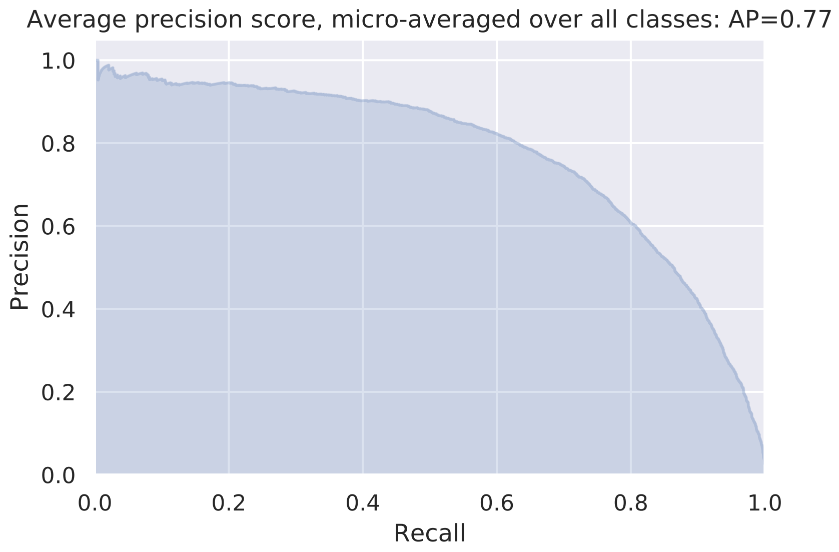


Figure 30: Container Classifier Precision-Recall Curve

To better understand the classifier's deficiencies, it is helpful to once again look at the confusion matrix. Several classes fall under the same family of containers (Full Hotel Pan, Half Hotel Pan, Quarter Hotel Pan), but are ultimately unable to distinguish between depths. There is also confusion between families of pans, such as the Quarter Hotel Pan (6-inch) Plastic and the Sixth Pan (6-inch) Plastic and the Cambro and Lexan classes. Some of these issues are due to persistent class noise, which can be remedied by further data cleaning or customizing to a particular customer's needs. This may also help with depth perception, but since all of the images are taken from the bird's eye view, distinguishing between these types of containers may still be difficult. As this model has reached 94% Top-5 accuracy, utlizing it would still be better than the alternative of manual classification which has been shown to be prone to user error.

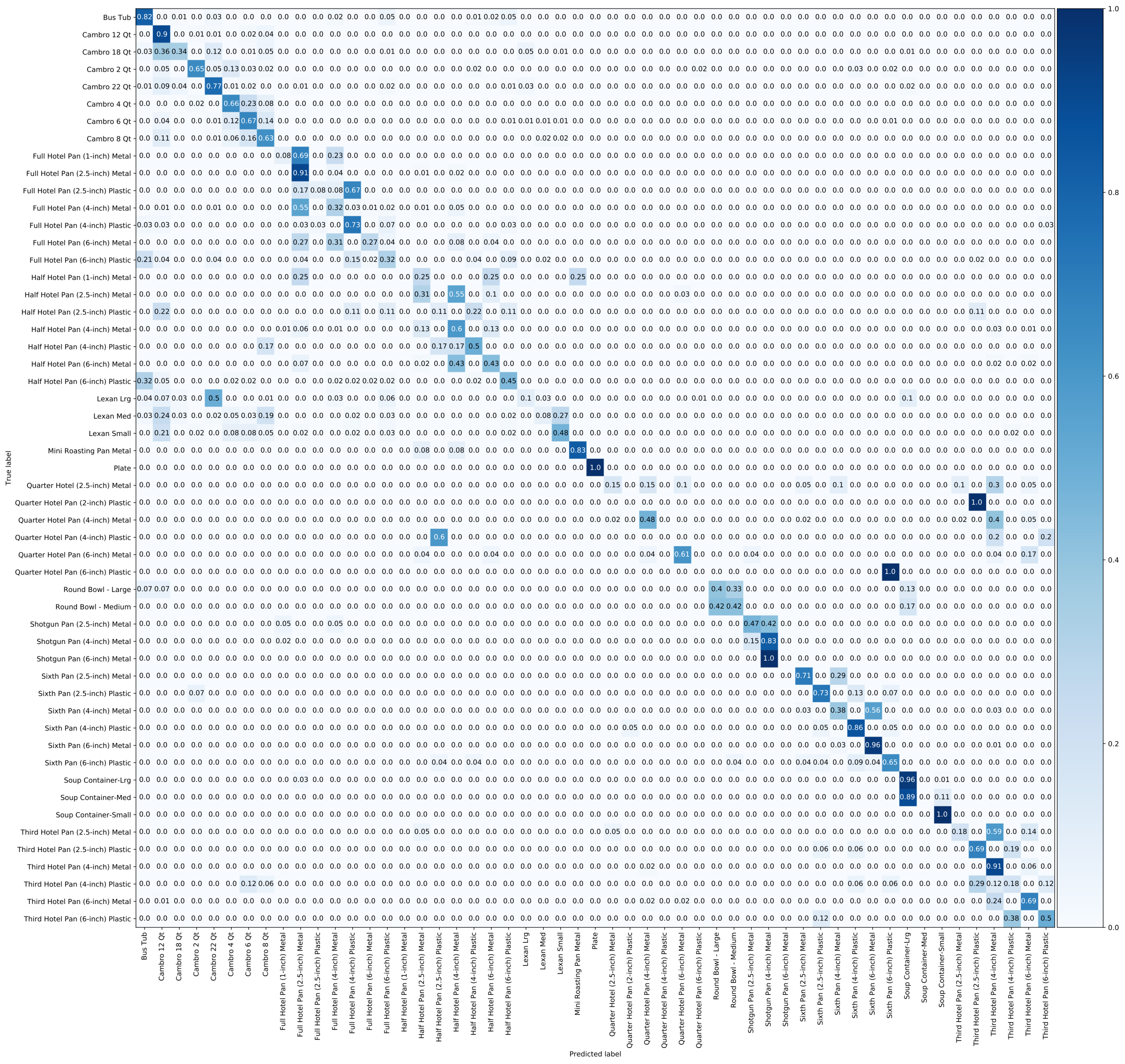
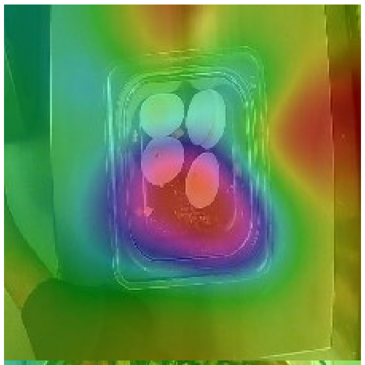


Figure 31: Container Confusion Matrix

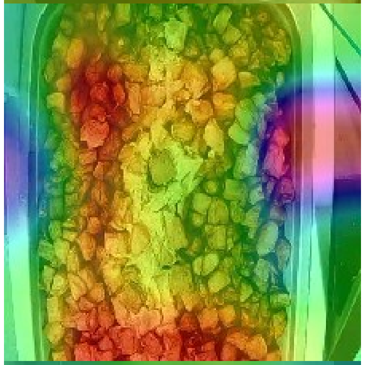
Some selected Grad-CAM visualizations are shown in Figure 32. For each example, the true class appears within the top-5 predictions, but not necessarily the top-1. It is clear that the model is utilizing features derived from the edges of the containers, but it seems to have some difficulty with container depth.



True Class: **t Pan - nc Plas c**

Top 5 Predictions:

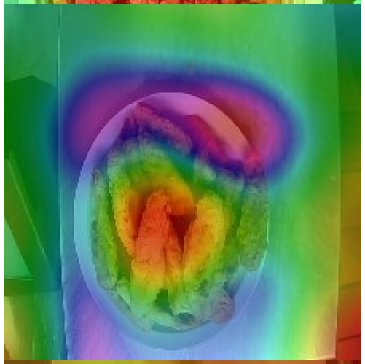
Si th Pan 2.5-inch Plastic 2.5%
Shotgun Pan 6-inch Metal 7.7%
t Pan - nc Plas c 2.9%
uarter otel Pan 2-inch Plastic 1.6%
uarter otel Pan -inch Plastic 1.3%



True Class: **ll otel Pan 2.5- nc Metal**

Top 5 Predictions:

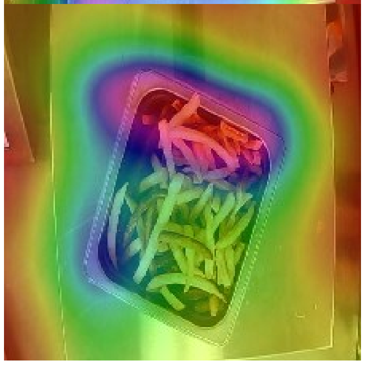
alf otel Pan -inch Metal 3 .3%
ll otel Pan 2.5- nc Metal 23. %
Full otel Pan -inch Metal 20. %
Full otel Pan 1-inch Metal 5.1%
alf otel Pan 2.5-inch Metal .2%



True Class: **Plate**

Top 5 Predictions:

Plate 5 . %
Third otel Pan -inch Metal 12.2%
Ca bro 2 t 6. %
Full otel Pan 2.5-inch Metal 3.2%
Full otel Pan 6-inch Plastic 2.5%



True Class: **t Pan - nc Metal**

Top 5 Predictions:

Si th Pan 2.5-inch Metal 6. %
Shotgun Pan -inch Plastic .7%
t Pan - nc Metal .3%
uarter otel Pan 2.5-inch Metal 1.1%
uarter otel Pan -inch Metal 1.1%

Figure 32: Selected Grad-CAM Visualizations for Container Model

6.4 Grouped Container Dataset

Grouping the container classes together by parent class (Half Hotel Pan, Full Hotel Pan, etc...) and material (Metal or Plastic), while ignoring depth yields the grouped container dataset. The best top-5 accuracy was observed in epoch 22 at 98.6%, which is very close to the goal of 99%. Top-1 accuracy at this epoch was 84.6%. The training and test curves can be found in Figure 33.

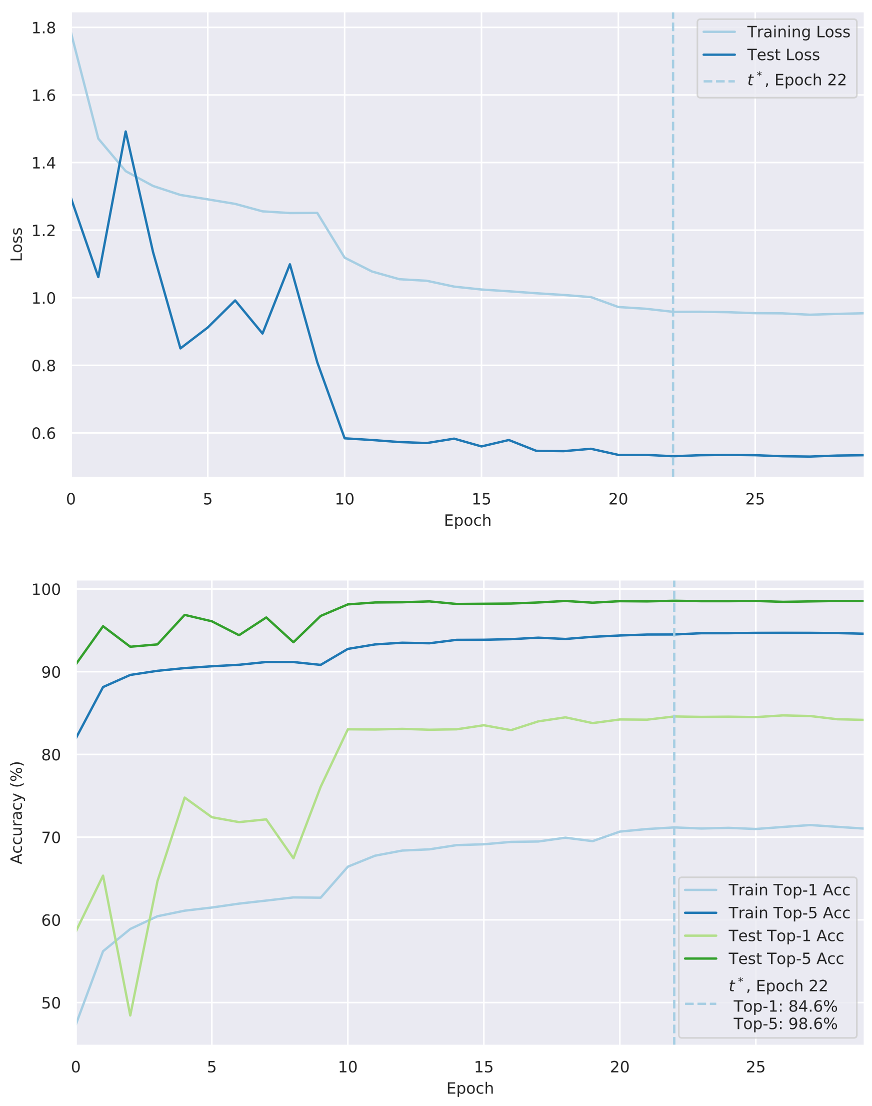


Figure 33: Grouped Container Model Performance. Loss, Accuracy versus Epoch.

The micro-averaged precision-recall curve is shown in Figure 34. The AP is 0.91, which is very close to a maximum of 1.0. The Fisher score for this classifier is 0.84, which is a significant improvement from the model which considered container depth.

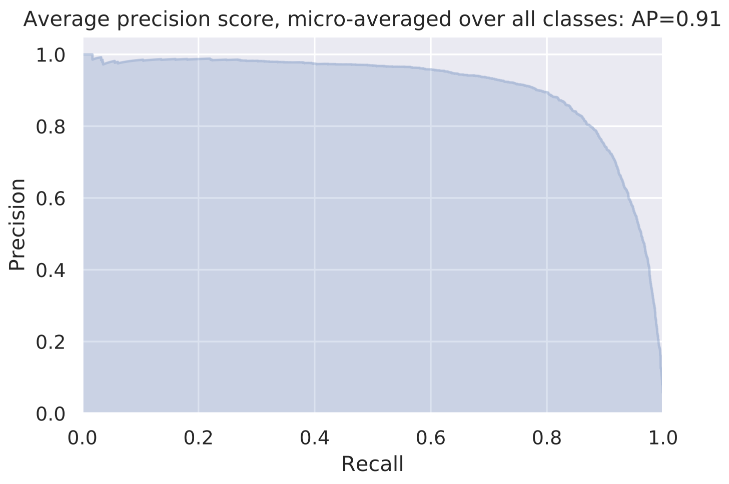
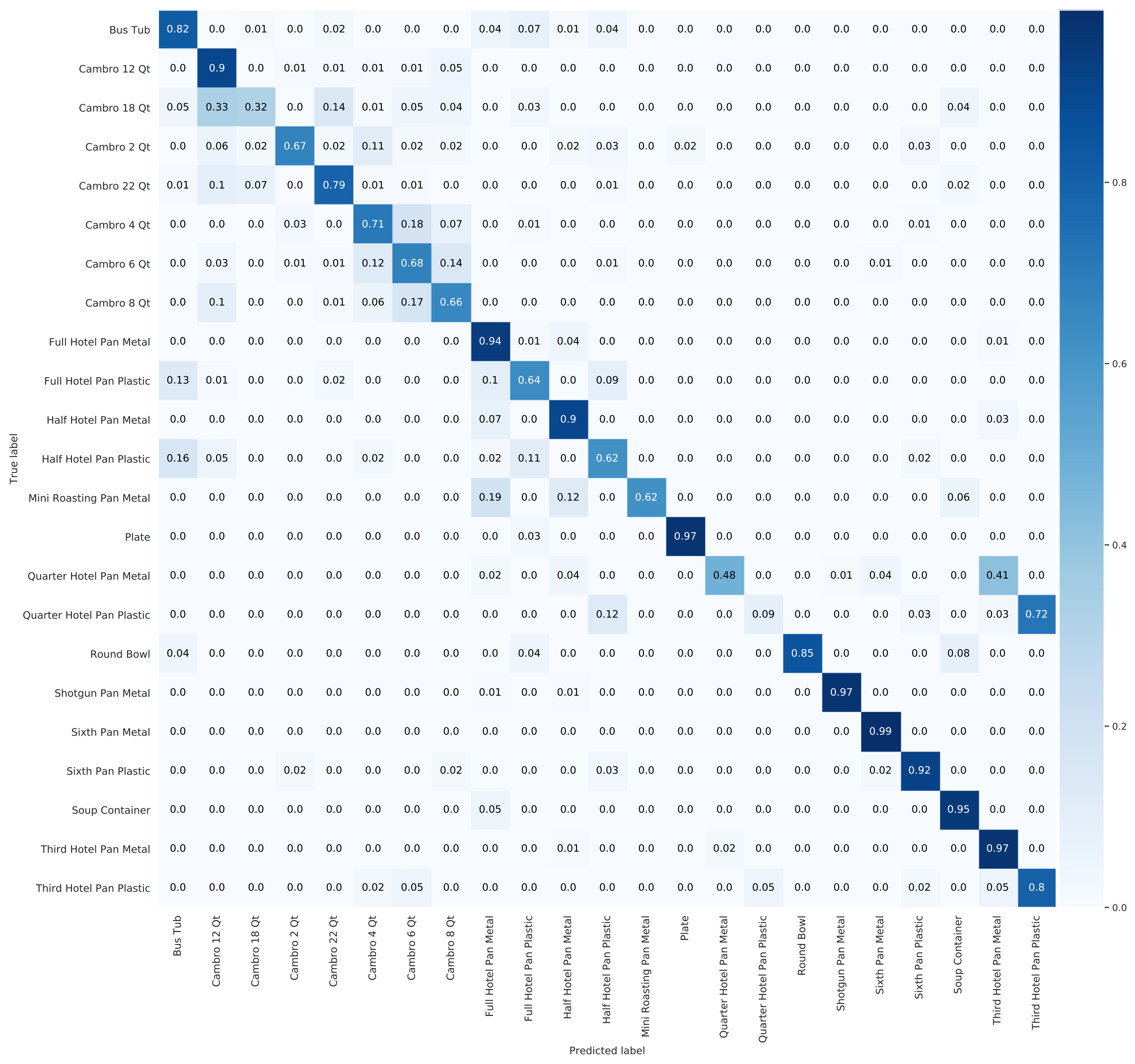


Figure 34: Grouped Container Precision-Recall Curve

Examining the confusion matrix in Figure 35, it is clear that the deficiencies are worse for certain categories, such as Quarter Hotel Pan Plastic and Cambro 18 Qt. Taking a closer look at the examples reveals that many images of the Quarter Hotel Pan Plastic are very similar to the Third Hotel Pan Plastic, which may explain why 72% of the test set examples are classified to that category. There is also a tendency to classify these examples as Half Hotel Pan Plastic and Sixth Pan Plastic. For the Cambro group of containers, the classifier has a tendency to guess the correct family but has some difficulty assigning the exact size. Examining the individual examples reveals that the ground truth is quite noisy, and further cleaning may be necessary to improve performance. The model does learn to distinguish metal from plastic, which is critical to ensuring the correct weight is subtracted for a container. This model is a good choice if container depth is not process critical.



	Bus Tub	Cambro 12 Qt	Cambro 18 Qt	Cambro 2 Qt	Cambro 22 Qt	Cambro 4 Qt	Cambro 6 Qt	Cambro 8 Qt	Full Hotel Pan Metal	Full Hotel Pan Plastic	Half Hotel Pan Metal	Half Hotel Pan Plastic	Mini Roasting Pan Metal	Plate	Quarter Hotel Pan Metal	Quarter Hotel Pan Plastic	Round Bowl	Shotgun Pan Metal	Sixth Pan Metal	Sixth Pan Plastic	Soup Container	Third Hotel Pan Metal	Third Hotel Pan Plastic
Bus Tub	0.82	0.0	0.01	0.0	0.02	0.0	0.0	0.0	0.04	0.07	0.01	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cambro 12 Qt	0.0	0.9	0.0	0.01	0.01	0.01	0.01	0.05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cambro 18 Qt	0.05	0.33	0.32	0.0	0.14	0.01	0.05	0.04	0.0	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.04	0.0	0.0
Cambro 2 Qt	0.0	0.06	0.02	0.67	0.02	0.11	0.02	0.02	0.0	0.0	0.02	0.03	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.03	0.0	0.0	0.0
Cambro 22 Qt	0.01	0.1	0.07	0.0	0.79	0.01	0.01	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02	0.0	0.0
Cambro 4 Qt	0.0	0.0	0.0	0.03	0.0	0.71	0.18	0.07	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.0
Cambro 6 Qt	0.0	0.03	0.0	0.01	0.01	0.12	0.68	0.14	0.0	0.0	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.0	0.0
Cambro 8 Qt	0.0	0.1	0.0	0.0	0.01	0.06	0.17	0.66	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Full Hotel Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.94	0.01	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0
Full Hotel Pan Plastic	0.13	0.01	0.0	0.0	0.02	0.0	0.0	0.0	0.1	0.64	0.0	0.09	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Half Hotel Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.07	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.03	0.0
Half Hotel Pan Plastic	0.16	0.05	0.0	0.0	0.0	0.02	0.0	0.0	0.02	0.11	0.0	0.62	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02	0.0	0.0	0.0
Mini Roasting Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.19	0.0	0.12	0.0	0.62	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06	0.0	0.0
Plate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.03	0.0	0.0	0.0	0.97	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Quarter Hotel Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02	0.0	0.04	0.0	0.0	0.0	0.48	0.0	0.0	0.01	0.04	0.0	0.0	0.41	0.0
Quarter Hotel Pan Plastic	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.12	0.0	0.0	0.0	0.09	0.0	0.0	0.0	0.03	0.0	0.03	0.72
Round Bowl	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.85	0.0	0.0	0.0	0.08	0.0	0.0
Shotgun Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.97	0.0	0.0	0.0	0.0	0.0
Sixth Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.99	0.0	0.0	0.0	0.0
Sixth Pan Plastic	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.02	0.92	0.0	0.0	0.0
Soup Container	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.95	0.0	0.0
Third Hotel Pan Metal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.0	0.97	0.0
Third Hotel Pan Plastic	0.0	0.0	0.0	0.0	0.0	0.02	0.05	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.0	0.0	0.0	0.02	0.0	0.05	0.8

Figure 35: Grouped Container Confusion Matrix

Finally, some sample Grad-CAM visualizations are shown in Figure 36. These are the same samples which were also evaluated on the previous container dataset. Notice that each example's top-1 prediction aligns with the true class, and each visualization shows that the classifier is utilizing features extracted from the edges of the containers.

True Class: **t Pan Plas c**

Top 5 Predictions:

t Pan Plas c 93.5%
Si th Pan Metal 5.1%
uarter otel Pan Metal 0.3%
Third otel Pan Plastic 0.2%
alf otel Pan Metal 0.2%

True Class: **ll otel Pan Metal**

Top 5 Predictions:

ll otel Pan Metal 56. %
alf otel Pan Metal 3 .2%
Mini oasting Pan Metal .5%
uarter otel Pan Metal 3.3%
alf otel Pan Plastic 0.5%

True Class: **Plate**

Top 5 Predictions:

Plate 8 . %
Third otel Pan Metal .6%
Full otel Pan Metal 2. %
Si th Pan Metal 2. %
Full otel Pan Plastic 1. %

True Class: **t Pan Metal**

Top 5 Predictions:

t Pan Metal 8 . %
uarter otel Pan Metal . %
Third otel Pan Metal 1.7%
Si th Pan Plastic Metal 0. %
Full otel Pan Metal 0. %

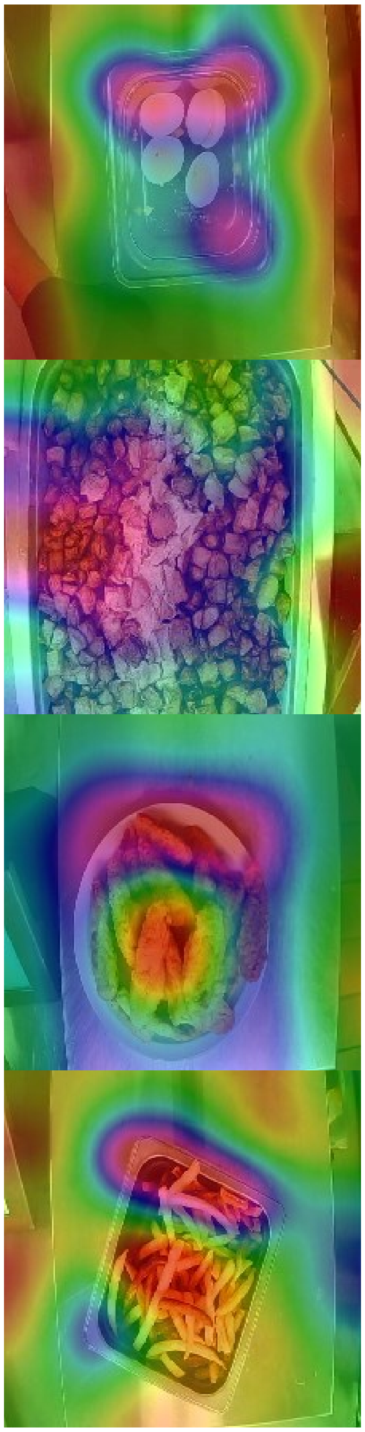


Figure 36: Selected Grad-CAM Visualizations for Grouped Container Model

6.5 Location X Dataset

For the waste item classification model trained on Location X data, the train and test performance for loss and accuracy can be found in Figure 37. As mentioned in section 3.4, two models were evaluated: one which was only pretrained on ImageNet, and the fine-tuned waste item model from section 6.1. The model which was fine-tuned on the other location's waste item data achieved a Top-1 accuracy of 81.6% and a Top-5 accuracy of 96.3%, while the ImageNet model achieved a Top-1 accuracy of 71.6% and a Top-5 accuracy of 91.7%. This suggests that there is a clear advantage utilizing data from other customer sites when learning to extract features in a CNN. Note that as all weights were frozen except the classification layer, the model was only able to extract features learned from training on the waste item dataset.

The confusion matrix in 39 shows poor classification performance for new classes, such as gravy, lasagna, noodles, and shellfish-shrimp. It is possible that the model may learn these classes by unfreezing some of the convolutional layers and training on a larger subset of data. Given that this dataset was much smaller than those from prior experiments, these results show promise for transfer learning.

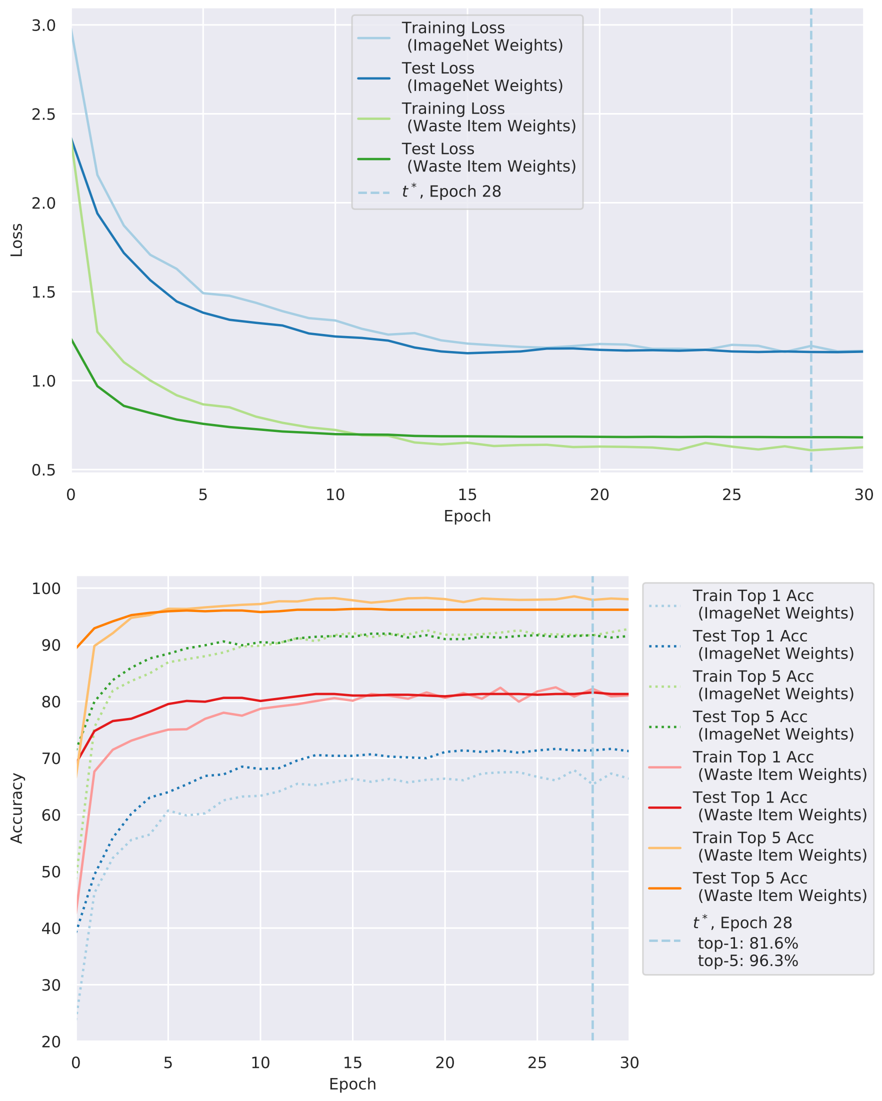


Figure 37: Location X Waste Item Model Performance. Loss, Accuracy versus Epoch.

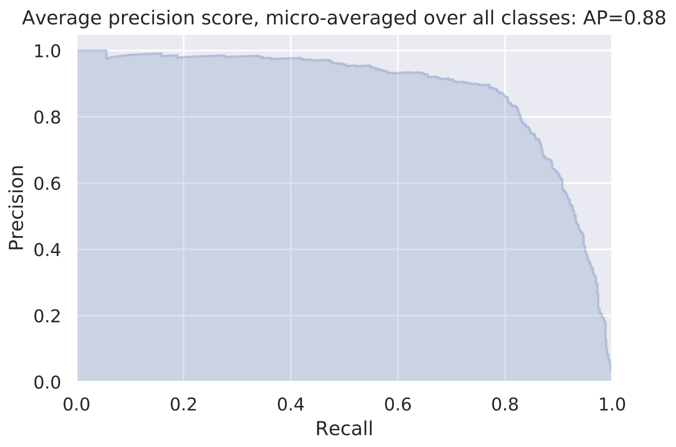


Figure 38: Location X Classifier Precision-Recall Curve

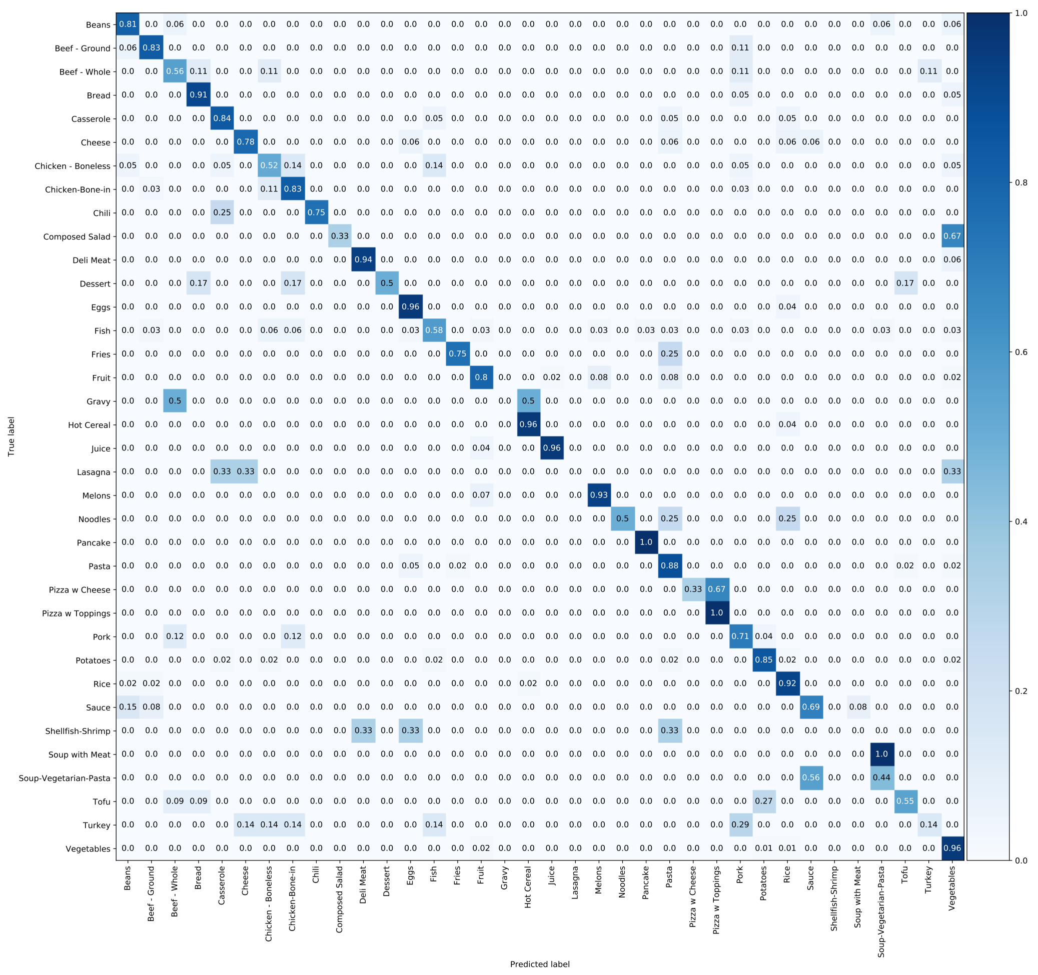


Figure 39: Location X Confusion Matrix

7 Summary and Conclusion

In this thesis, a novel dataset was utilized to train five distinct classifiers for the purposes of automating food waste tracking and reduction. The procedure used for data preparation and decision making was documented, along with the detailed design decisions and their rationale. A concept is proposed for several of these models to work together to form a robust food waste and container recognition system. The intention is that by introducing automation into food waste tracking, businesses utilizing this product will be able to more accurately track wasted food and cost while kitchen staff focuses on other important tasks. Increased accuracy in tracking food waste may also help the environment, as corrective action can be taken when efforts are focused on reducing waste in the correct food categories.

An important contribution from this work is to highlight best practices and design decisions from reputable sources and apply them in training a classifier on a novel dataset. One of these best practices is critical in judging model performance on unseen data —leaving the test partition untouched until the very end, to obtain a less biased estimate of the out of sample error. Another best practice conveyed is the importance of cleaning the test data. This is critical in order to accurately judge model performance, as shown by the large gap between the training and test performance.

Another important contribution was to convey the benefits of using transfer learning on this dataset. By utilizing transfer learning, top 5 accuracy was shown to converge within 30 epochs. Specifically, the food waste model achieved 98.8% top 5 accuracy, and the grouped container model achieved 98.6% top 5 accuracy. Without applying transfer learning, it can take several hundred epochs for error to converge at a satisfactory level, as shown in Appendix A. The experiments on the location X dataset showed the benefits of fine-tuning using a dataset which is closer to the actual task, in addition to initial training using ImageNet weights.

7.1 Limitations

The limitations of this work should be acknowledged. While many best practices for design decisions were outlined from various sources, these were not benchmarked against the Food-101 dataset to obtain similar results as those in [25]. This is a deficiency in verifying that they were good design decisions.

7.2 Future Work

There are many opportunities to improve these models through future work. Learning may be improved on some of the classes with poorer performance by gathering more examples and balancing the training data. For the food waste model, classes which display learning of correlated but undesirable features through Grad-CAM may need to be bolstered with more diverse examples. This can be accomplished by gathering and cleaning more data from other customer sites to create a more general model. One of the biggest challenges when cleaning the container data is verifying the accuracy of the depth and size. Since the pictures are taken from a “bird’s eye” view, it is difficult for a human to distinguish the depth. It may be unreasonable to expect an algorithm to do so as well. Introducing another image of the container at a different angle could help with this issue. There is also great potential to go beyond image classification and into the realm of Semantic Segmentation. Semantic Segmentation would be helpful in scenarios where a mixture of food is present, or learning the components of a specific dish. To support this work, many of the generalized classes would need to have increased granularity, such as Vegetables or Fruit, to identify what kinds of vegetables or fruit are present. An opportunity for semi-supervised learning also exists, in which trained models classify more examples and are retrained over time.

Another opportunity for future work exists in exploring the usefulness of fine-tuning a classifier on another benchmark dataset such as Food-101. It is possible that the Food-101 dataset could be used to learn to detect features which are better for classification in the waste item dataset. This would be similar to the experiment performed on the location X classifier.

As these models are planned for implementation on a mobile architecture, network parameter count and latency are of concern. Some work suggests that further reduction of parameters and network speedup may be possible through deep compression [51] and network/weight pruning [52] [53]. Future work should explore some of these methods of network parameter reduction and speeding up inference time.

While no model is perfect, a hope is that deploying these models into production will lead to a long-term increase in accuracy of collected data. When data is clean, models can be more accurately trained and scored, and customers have a better idea of what food is actually wasted. Efforts to reduce waste will hopefully back-propagate and result in a reduction in environmental effects over time.

References

[1] H. Eraqi, “Bag of visual words for image classification (caltech101 - surf features - matlab code),” Mar 2017. [Online]. Available: http://heraqi.blogspot.com/2017/03/BoW.html

[2] Aphex34, “Creative commons share-alike 4.0 license,” Creative Commons. [Online]. Available: https://creativecommons.org/licenses/by-sa/4.0/deed.en

[3] S. FOOD, “Global initiative on food loss and waste reduction,” *Key facts on food loss and waste you should know*, pp. 01–02, 2015.

[4] F. F. W. Footprints, “Impact on natural resources. summary report.(2013).”

[5] R. F. W. through Economics, “Data (refed), 2016. a roadmap to reduce us food waste by 20 percent.”

[6] “Unpacking the sustainability landscape,” Sep 2018. [Online]. Available: https://www.nielsen.com/us/en/insights/report/2018/unpacking-the-sustainability-landscape/

[7] S. Finn, “40 million pounds of food waste prevented in 5 years,” Jul 2019. [Online]. Available: https://blog.leanpath.com/40-million-pounds-of-food-waste-prevented-in-5-years

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101 – mining discriminative components with random forests,” in *European Conference on Computer Vision*, 2014.

[10] D. Garcia-Gasulla, F. Parés, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, “On the behavior of convolutional nets for feature extraction,” *Journal of Artificial Intelligence Research*, vol. 61, pp. 563–592, 2018.

[11] Y. LeCun, C. Cortes, and C. J. Burges, “The mnist database of handwritten digits, 1998,” *URL http://yann. lecun. com/exdb/mnist*, vol. 10, p. 34, 1998.

[12] Y. Kawano and K. Yanai, “Automatic expansion of a food image dataset leveraging existing categories with domain adaptation,” in *Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*, 2014.

[13] A. Singla, L. Yuan, and T. Ebrahimi, “Food/non-food image classification and food categorization using pre-trained googlenet model,” in *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management.* ACM, 2016, pp. 3–11.

[14] Y. Kawano and K. Yanai, “Foodcam-256: a large-scale real-time mobile food recognitionsystem employing high-dimensional features and compression of classifier weights,” in *Proceedings of the 22nd ACM international conference on Multimedia.* ACM, 2014, pp. 761–762.

[15] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[16] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, “Image classification with the fisher vector: Theory and practice,” *International journal of computer vision*, vol. 105, no. 3, pp. 222–245, 2013.

[17] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of keypoints,” in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, no. 1-22. Prague, 2004, pp. 1–2.

[18] D. G. Lowe *et al.*, “Object recognition from local scale-invariant features.” in *iccv*, vol. 99, no. 2, 1999, pp. 1150–1157.

[19] W. T. Freeman and M. Roth, “Orientation histograms for hand gesture recognition,” in *International workshop on automatic face and gesture recognition*, vol. 12, 1995, pp. 296–301.

[20] K. Yanai and Y. Kawano, “Food image recognition using deep convolutional network with pre-training and fine-tuning,” in *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW).* IEEE, 2015, pp. 1–6.

[21] G. M. Farinella, D. Allegra, M. Moltisanti, F. Stanco, and S. Battiato, “Retrieval and classification of food images,” *Computers in biology and medicine*, vol. 77, pp. 23–39, 2016.

[22] L. Perez and J. Wang, “The effectiveness of data augmentation in image classification using deep learning,” *arXiv preprint arXiv:1712.04621*, 2017.

[23] Y. Matsuda, H. Hoashi, and K. Yanai, “Recognition of multiple-food images by detecting candidate regions,” in *2012 IEEE International Conference on Multimedia and Expo.* IEEE, 2012, pp. 25–30.

[24] N. Martinel, G. L. Foresti, and C. Micheloni, “Wide-slice residual networks for food recognition,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 567–576.

[25] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *arXiv preprint arXiv:1905.11946*, 2019.

[26] P. McAllister, H. Zheng, R. Bond, and A. Moorhead, “Combining deep residual neural network features with supervised machine learning algorithms to classify diverse food image datasets,” *Computers in biology and medicine*, vol. 95, pp. 217–233, 2018.

[27] C. Saran, “How computer vision powered by nvidia helped winnow tackle food waste,” Jun 2019. [Online]. Available: https://www.computerweekly.com/news/252465110/How-computer-vision-powered-by-Nvidia-helped-Winnow-tackle-food-waste

[28] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data.* AMLBook New York, NY, USA:, 2012, vol. 4.

[29] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning.* Springer, 2013, vol. 112.

[30] A. Burkov, *The Hundred-Page Machine Learning Book.* Andriy Burkov, 2019.

[31] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[32] S. S. Al-Amri, N. Kalyankar, and S. Khamitkar, “Image segmentation by using edge detection,” *International journal on computer science and engineering*, vol. 2, no. 3, pp. 804–807, 2010.

[33] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[34] F. Rosenblatt, “Principles of neurodynamics. perceptrons and the theory of brain mechanisms,” Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[35] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[36] D. H. Hubel and T. Wiesel, “Shape and arrangement of columns in cat’s striate cortex,” *The Journal of physiology*, vol. 165, no. 3, pp. 559–568, 1963.

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

[38] M. Wang and W. Deng, “Deep face recognition: A survey,” *arXiv preprint arXiv:1804.06655*, 2018.

[39] A. Karpathy, “Cs231n convolutional neural networks for visual recognition, 2016,” *URL http://cs231n.github.io/convolutional-networks/*, 2017.

[40] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[41] 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,” *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[42] J. Brownlee, “A gentle introduction to object recognition with deep learning,” *URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/*, 2019.

[43] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” 2016.

[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[45] S. Raschka, “What is the best validation metric for multi-class classification?” Sep 2019. [Online]. Available: https://sebastianraschka.com/faq/docs/multiclass-metric.html

[46] Lukemelas, “lukemelas/efficientnet-pytorch,” Oct 2019. [Online]. Available: https://github.com/lukemelas/EfficientNet-PyTorch

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.

[48] G. B. Orr and K.-R. Müller, *Neural networks: tricks of the trade.* Springer, 2003.

[49] S. Kornblith, J. Shlens, and Q. V. Le, “Do better imagenet models transfer better?” 2018.

[50] U. Ozbulak, “Pytorch cnn visualizations,” https://github.com/utkuozbulak/pytorch-cnn-visualizations, 2019.

[51] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[52] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[53] T. Zhang, S. Ye, Y. Zhang, Y. Wang, and M. Fardad, “Systematic weight pruning of dnns using alternating direction method of multipliers,” *arXiv preprint arXiv:1802.05747*, 2018.

[54] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” 2017.

[55] A. Dutta and A. Zisserman, “The VIA annotation software for images, audio and video,” in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM ’19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3343031.3350535

[56] A. Dutta, A. Gupta, and A. Zissermann, “VGG image annotator (VIA),” http://www.robots.ox.ac.uk/ vgg/software/via/, 2016, version: X.Y.Z.

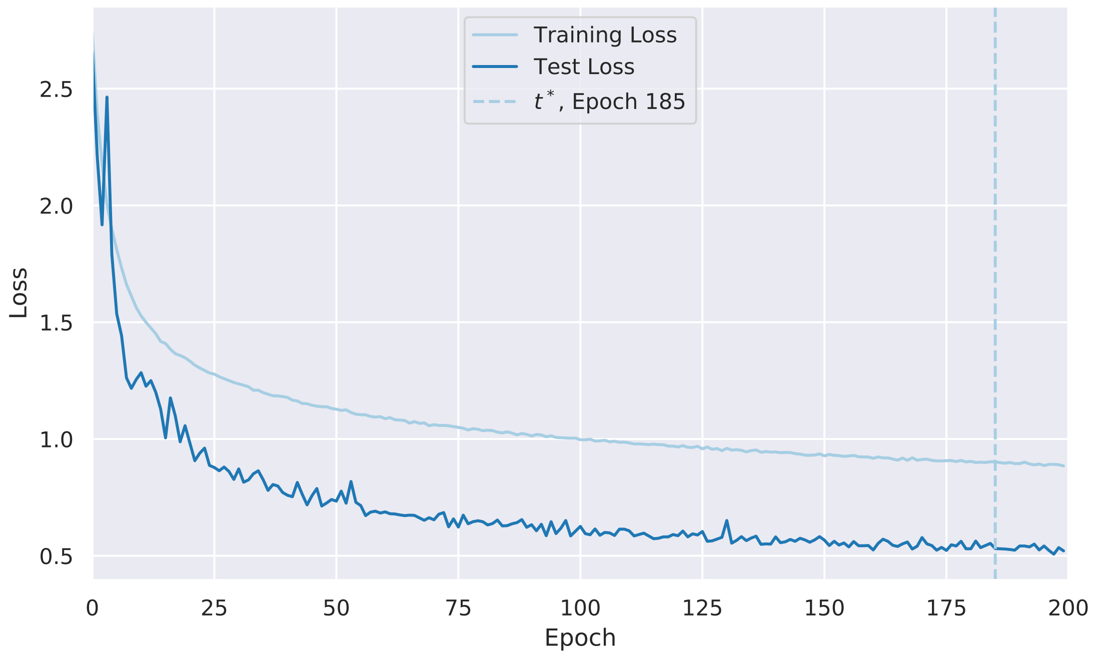
[57] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco: Common objects in context,” 2014.

[58] F. Massa and R. Girshick, “maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch,” https://github.com/facebookresearch/maskrcnn-benchmark, 2018.

A The Advantages of Transfer Learning

The food waste classifier in Section 6.1 achieved a Top-1 accuracy of 90.1%, and a Top-5 accuracy of 98.8% within only 30 epochs. One of the reasons why it was able to learn so quickly is because the network had already learned to extract some features through pretraining on the ImageNet dataset. In order to explore how much time could be saved through utilizing pretrained models, a separate model was randomly initialized and trained from scratch on the food waste dataset. Instead of utilizing cross validation to find the best hyperparameters, the Adagrad optimizer was used with default PyTorch settings. Training was performed for 200 epochs, the results from which are shown in Figure 40.

At epoch 185, the classifier achieved a maximum Top-1 accuracy of 85%, and a maximum Top-5 accuracy of 97.2%. While reasonably good, this is not competitive with the pretrained model, which yields better performance within only 30 epochs. On the Google Cloud Compute Engine, total training time for the pretrained model using a high memory instance with 8vCPUs, 52GB RAM, and NVIDIA P100 GPU is roughly 110 minutes and 20 seconds. Training a model for 200 epochs without pretraining will take over 12 hours.



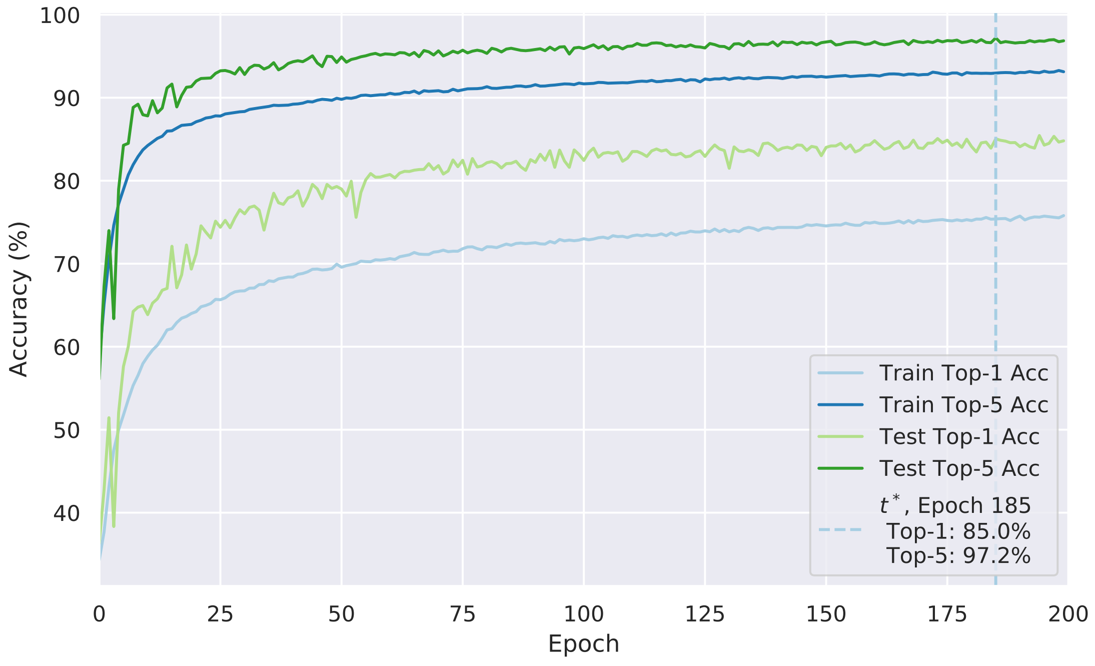


Figure 40: Waste Item Model Performance, Trained from Random Initialization. Loss, Accuracy versus Epoch

B Performance of Various Architectures

As another exercise, the performance of different CNN architectures was compared against our chosen model, EfficientNetB0. The models evaluated are:

- EfficientNetB1
- GoogleNet
- InceptionV3
- MobileNetV2

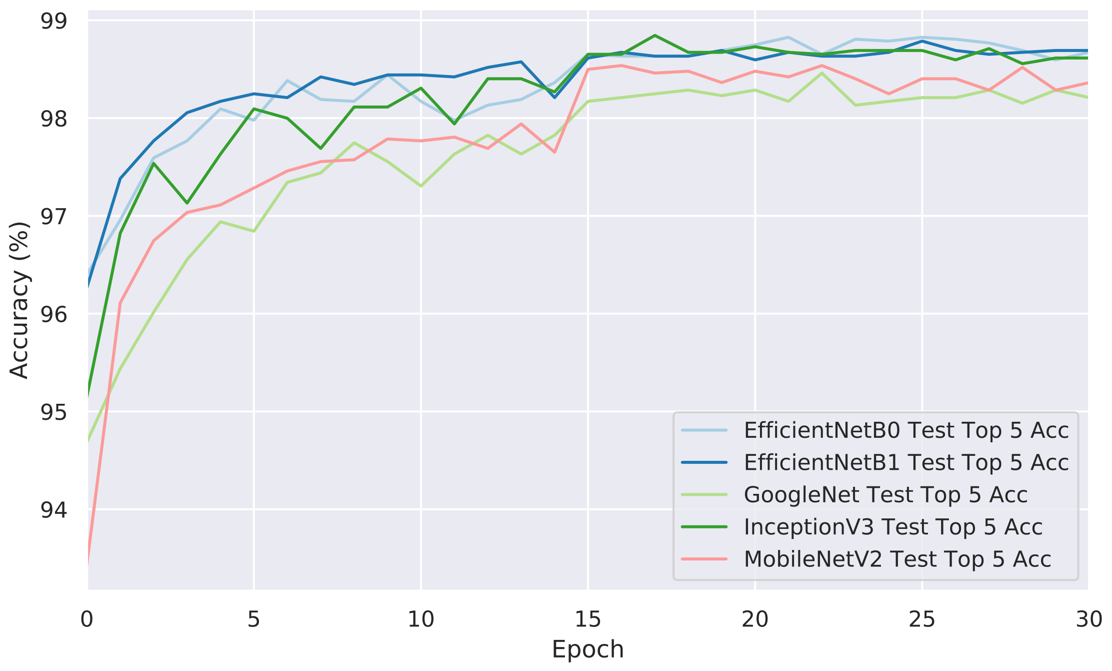
For consistency, each model was trained using the same hyperparameters obtained via cross-validation on with the EfficientNetB0 model. As a result, the test may be biased in favor of our chosen model. A record of the highest accuracies obtained by each model is given in Table 6. The results versus epoch from this experiment are shown in the following Figure 41.

Model	**Best Top-1 Acc. (%), Epoch**	**Best Top-5 Acc. (%), Epoch**	**Parameter Count**
EfficientNetB0	90.7, Epoch 16	98.8, Epoch 21	5.29M
EfficientNetB1	90.7, Epoch 17	98.8, Epoch 25	7.79M
GoogleNet	89.6, Epoch 20	98.5, Epoch 22	6.8M
InceptionV3	90.4, Epoch 20	98.9, Epoch 17	23M
MobilenetV2	89.7, Epoch 26	98.5, Epoch 16	4.52M

Table 6: Comparison of Performance and Size for Various Models.

While the highest Top-5 accuracy was achieved by InceptionV3, all of the results are very close. The results from training the larger EfficientNetB1 model show that the additional parameters do not aid in learning any additional features on our dataset.

From these results, EfficientNetB0 is a good choice due to its modest parameter count and high performance.



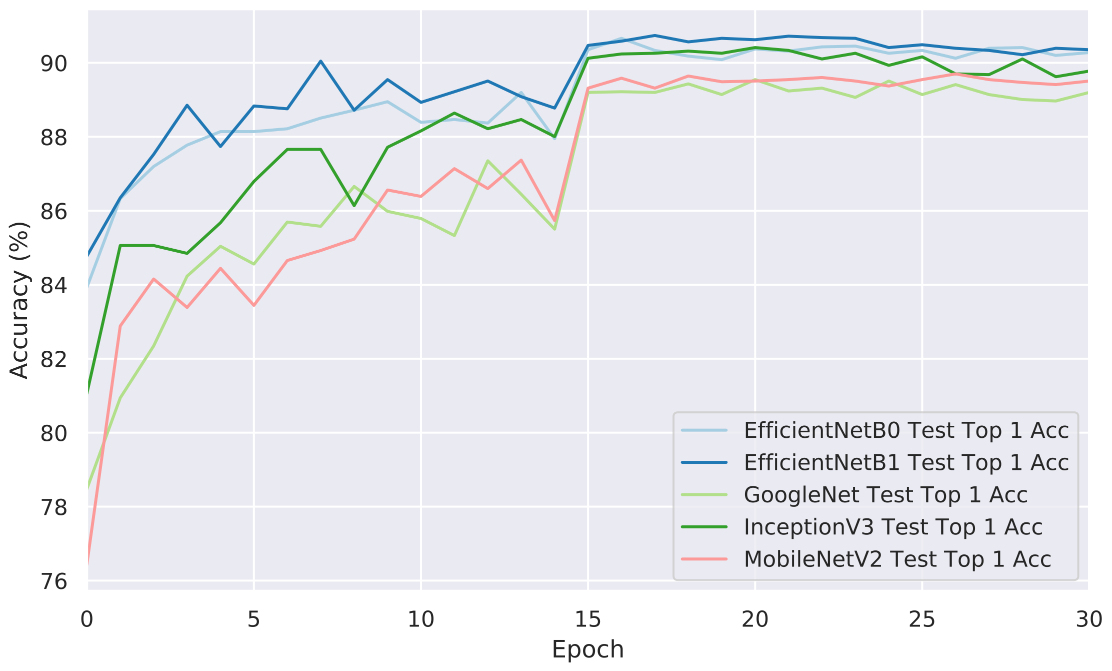


Figure 41: Performance of Various Models Trained on Waste Item Dataset.

C Instance Segmentation: POC

Finally, this thesis is concluded with a demo which outlines the potential for instance segmentation on this data. One of the more popular frameworks for instance segmentation is Facebook AI Research's *Mask-RCNN* [54]. Mask-RCNN essentially utilizes features extracted through a pretrained network, called a *backbone*, to propose regions for bounding box coordinates while also classifying each pixel. The regions are proposed in a branch of the network called the *region proposal network*, and the masks are predicted in parallel head using a fully connected layer. The resulting masks can be filtered by the model's confidence in prediction.

In order to train Mask-RCNN, it must be given data with properly annotated masks. This annotation can be a very slow process. As such, a small dataset of 300 training examples and 30 test examples was created using VIA Image Annotator [55] [56]. There are roughly 11 classes of food in total. The annotations for this data were exported in the popular COCO format [57]. For this demo, the open-sourced code for Mask-RCNN found here [58] was used, with fbnet as the model's backbone (pre-trained on the food waste item dataset).

Some example predictions can be found in Figures 42, 43. 44. While the confidence scores are fairly low and the boundaries are rough, is classifier generally able to distinguish between the types of food in the images. This shows some potential for future success in an instance segmentation model.

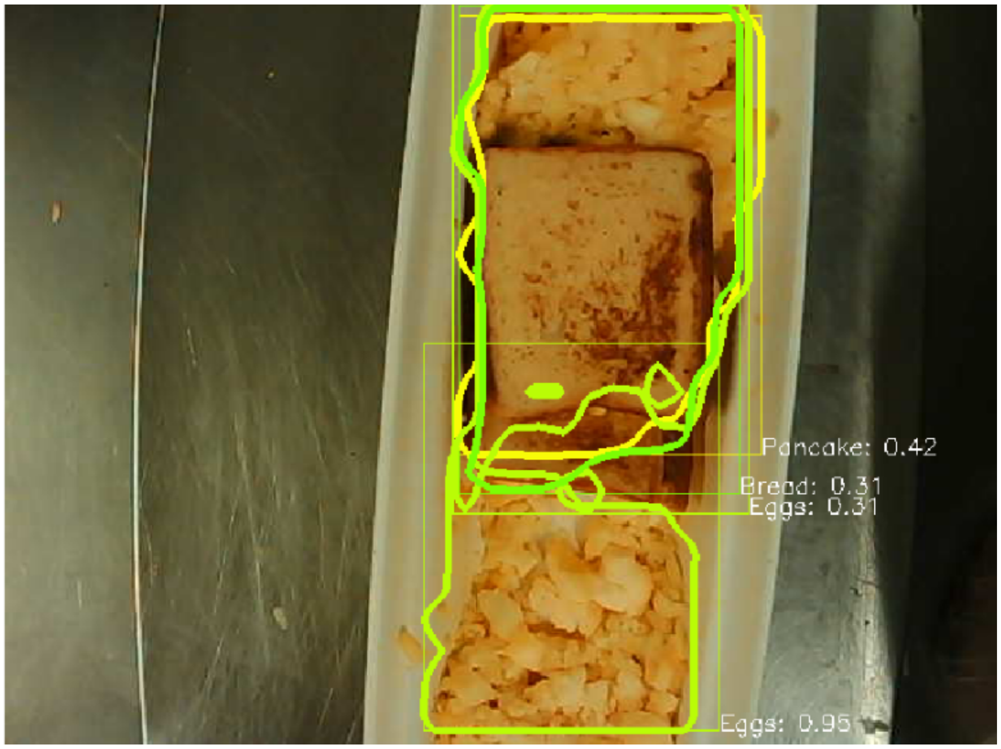


Figure 42: Example with Bread and Eggs.

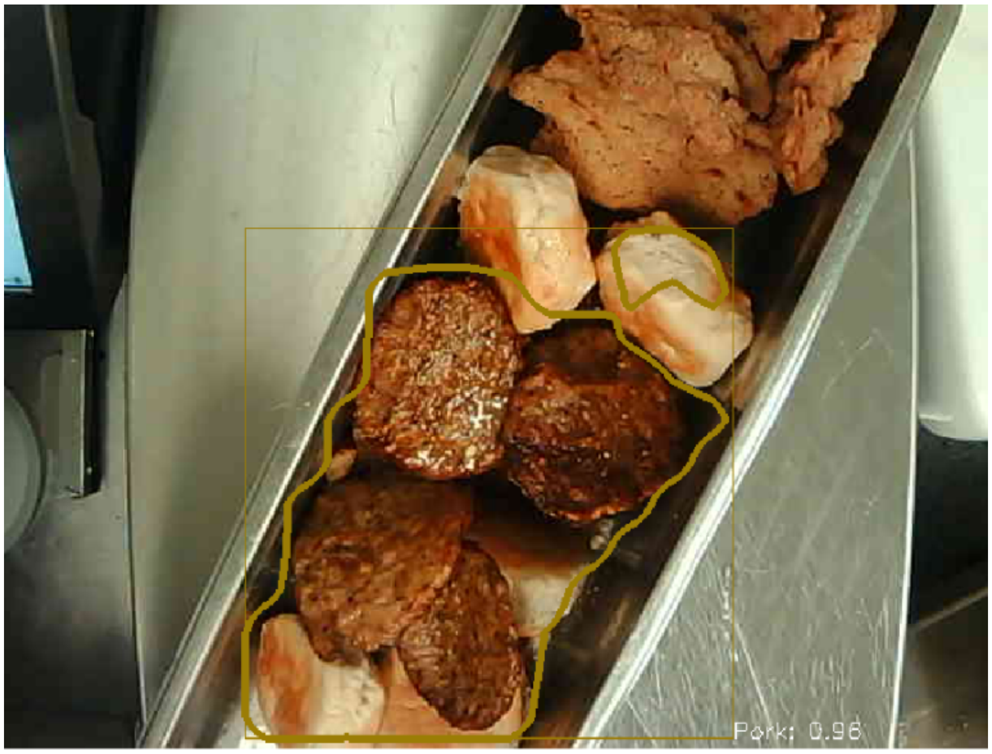


Figure 43: Example with Pork and Bread [Rolls].

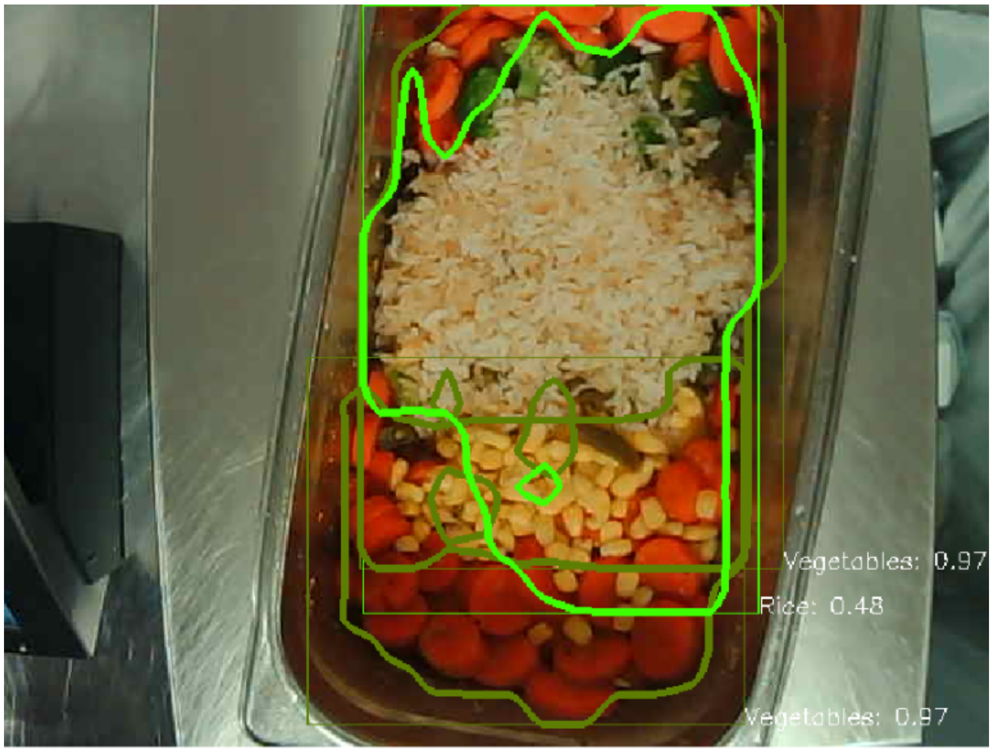


Figure 44: Example with Rice and Vegetables.