Improving Food Recognition for Mobile Applications via Convolutional Neural Networks and User Habits

Mete Aykul

Master of Engineering

Department of Electrical and Computer Engineering

McGill University

Montreal, Canada

2019-07-05

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering

DEDICATION

This document is dedicated to my parents, and to my brother, without whose love and support I would not be where I am today.

ACKNOWLEDGEMENTS

I would like to thank Professor Zeljko Zilic for his supervision and advice, which was crucial in allowing this thesis to come to fruition. I would also like to thank Jianing Sun, a colleague at IML, for her assistance in formulating ideas and experiments. Lastly, I would like to thank all my friends who've either shown me support, or provided me with countless, instrumental pieces of advice in passing.

ABSTRACT

This thesis aims to take a step toward mitigating the growing concern of poor dietary health, and associated conditions like obesity and diabetes; it focuses on using runtime-efficient convolutional neural networks (CNNs) to improve the accuracy and efficiency of food recognition on mobile devices. In general, it is hoped that the methods and findings presented here help facilitate quality-of-life improvements to the mobile dietary health applications that many rely on today. First, the DenseNet and MobileNetV2 architectures are trained on the UEC Food datasets, and achieve up to 82% top-1 single-crop accuracy using significantly-less parameters than similar work. Their performance was found to be state-of-the-art in parameter-efficiency, and their on-mobile advantages in runtime speed and memory-usage were highlighted with respect to competing approaches. In exploring the training space, the success of atypical choices – such as minimal data augmentation and fine-tuning all weights – are highlighted and discussed, particularly the implication that smaller networks may in general be easier and faster to train than larger counterparts. Second, a novel user habit system is proposed, that increases food recognition accuracy by leveraging users' dietary history – or user context – to personalize and augment top-5 classifier predictions. The scheme increased top-1 accuracy by 14% on a proof-of-concept evaluation with respect to no user context, up to 82%, despite increasing classification difficulty through the inclusion of background noise and multi-class images – which the proposed system does not explicitly handle. Third, guidelines for future effort are provided by identifying important, related, orthogonal areas-of-focus

– like calorie estimation –, and deeply exploring the scope of their challenges and approaches – both present and prospective; the findings were then used to generate guidelines through the contribution of domain-specific and -agnostic discussions, that could lead to practical implementations of other core features in a dietary health application. Overall, automatic food recognition may be a promising avenue for developing novel methods to mitigate the growing concerns associated with poor dietary health; the larger hope is that this work provides practical insights into the development of a closer-to-ideal mobile food recognition system that is realizable, and capable of providing benefit to those that need it.

ABRÉGÉ

Cette thèse vis de faire un pas en avant pour atténuer l'inquiétude croissante que suscitent les problèmes de santé liées à l'alimentation et les problèmes associés tels que l'obésité et le diabète; Il se concentre sur l'utilisation de réseaux de neurones à convolution (CNN) efficaces pour l'exécution afin d'améliorer la précision et l'efficacité de la reconnaissance des aliments sur des appareils mobiles. En général, on espère que les méthodes et les résultats présentés ici contribueront à améliorer la qualité de vie des applications de santé diététique mobiles sur lesquelles beaucoup s'appuient aujourd'hui. Premièrement, les architectures DenseNet et MobileNetV2 sont formées sur les jeux de données UEC Food et permettent d'atteindre une précision pouvant aller jusqu'à 82 % top-1 pour une seule culture en utilisant beaucoup moins de paramètres que des travaux similaires. Leurs performances se sont avérées être à la pointe de l'efficacité en termes de paramètres, et leurs avantages sur mobile en termes de vitesse d'exécution et d'utilisation de la mémoire ont été mis en évidence par rapport aux approches concurrentes. Dans l'exploration de l'espace de formation, le succès de choix atypiques - tels que l'augmentation minimale des données et le réglage précis de toutes les pondérations - est mis en évidence et discuté, en particulier le fait qu'il peut être généralement plus facile et plus rapide de former des réseaux plus petits que des homologues plus grands.Deuxièmement, un nouveau système d'habitudes d'utilisateur est proposé, qui augmente la précision de la reconnaissance des aliments en exploitant l'historique alimentaire de l'utilisateur - ou

son contexte - pour personnaliser et augmenter les prédictions du top 5 des classificateurs. Le schéma a permis d'augmenter la précision du premier niveau de 14% sur une évaluation de la validation de principe par rapport à l'absence de contexte utilisateur, jusqu'à 82%, en dépit de la difficulté croissante de classification en raison de l'inclusion du bruit de fond et des images multi-classes - qui le système proposé ne gère pas explicitement. Troisièmement, des lignes directrices pour les efforts futurs sont fournis en identifiant les domaines d'intervention orthogonaux importants, connexes, comme l'estimation des calories, et en explorant en profondeur la portée de leurs défis et leurs approches - actuelles et prospectives; Les résultats ont ensuite été utilisés pour générer des lignes directrices via l'apport de discussions spécifiques à un domaine et diagnostiques, qui pourraient conduire à la mise en œuvre pratique d'autres fonctionnalités essentielles dans une application de santé alimentaire. Dans l'ensemble, la reconnaissance automatique des aliments pourrait être un moyen prometteur de développer de nouvelles méthodes pour atténuer les préoccupations croissantes liées à une mauvaise alimentation. Le plus grand espoir est que ces travaux fournissent des informations pratiques sur la mise au point d'un système de reconnaissance des aliments mobiles plus proche de l'idéal, réalisable et susceptible de procurer des avantages à ceux qui en ont besoin.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

KEY TO ABBREVIATIONS

API Application Programming Interface

BN Batch Normalization

CNN Convolutional Neural Networks

CVPR Conference on Computer Vision and Pattern Recognition

FAO Food and Agricultural Organization of the United Nations

FC Fully Connected

FV Fisher Vector

GPU Graphics Processing Unit

ILSVRC ImageNet Large Scale Visual Recognition Competition

JSON JavaScript Object Notation

MB Megabyte

ms Millisecond

NNAPI Neural Networks API

ReLU Rectified Linear Units

ResNet Residual Neural Networks

RMSProp Root Mean Square Propagation

SIFT Scale Invariant Feature Transform

SVM Support Vector Machine

tanh Hyperbolic Tangent

TPU Tensor Processing Unit

UEC100 UEC Food-100

UEC256 UEC Food-256

USDA United States Department of Agriculture

WHO World Health Organization

WRN Wide Residual Network

XML eXtensible Markup Language

CHAPTER 1
Introduction

Health is an extremely precious commodity that is often overlooked in the busyness of life. Dietary health in particular is easy to neglect; as life constantly hustles people from one thing to another, the prospect of a quick-and-convenient meal often trumps that of a proper-and-balanced one. The unfortunate reality is that a poor diet can lead to obesity, diabetes, and cardiovascular diseases [1], to name a few; it is extremely important to avoid such complications as they can further lead to mortality – for example, diabetes was the 11th leading cause of death in 2002, and projected to be the 7th by 2030 [2]. Many have taken to diet tracking applications in a bid to facilitate a healthier lifestyle [3], and while they have been somewhat effective, they are still often abandoned – prior to meeting set goals – for both unspecified reasons and for a lack of features [3]; coupled with the growing prevalence of diabetes [4], there is a clear need for novel solutions to mitigate this growth [5].

Recently, mobile device-based food recognition has been of interest to the scientific community, for its potential to significantly reduce user-made errors – such as typos or wrong measurements when making manual calorie entries – and improve the quality-of-life of diet tracking applications. With the re-emergence of convolutional neural networks (CNNs) [6] facilitating breakthroughs in virtually all computer vision tasks, food recognition has similarly seen a substantial advancement of its state-of-the-art results across numerous datasets. In addition, the use of mobile devices as

a primary platform is of increasing interest; their growing ubiquity, camera access, portability, and modest computational capabilities, make them an ideal platform in many regards. When considered together, these factors make the inception of an ideal mobile food recognition platform less distant and more realizable.

This thesis takes a step toward an ideal mobile food recognition platform by exploring numerous methods for further improving food recognition performance; specifically, various schemes are scrutinized to maximize accuracy while respecting the memory-and-time constraints a mobile system may impose.

1.1 Context and Motivation

The primary motivation stems from poor dietary health being linked to a myriad of diseases and conditions [1], and that the concern is only growing; current approaches have been demonstrated to be largely ineffective at mitigating the growth [3], signifying a need for new and novel methods. In addition, the numerous, worrying trends and statistics that substantiate the growing concern are staggering, and introduces a sense of urgency in the development of new approaches:

- The number of people with diabetes has grown from 108 million in 1980 to 422 million in 2014; furthermore, there is a more rapid rise in prevalence in middle- and low-income countries [4].
- Diabetes was the 11th leading cause of mortality in 2002, and projected to be the 7th by 2030 [2]; it was already estimated to be the 7th leading cause in 2016 [4].
- A healthy diet can delay or even prevent the onset of type 2 diabetes; if already afflicted, a healthy diet can treat it [4].

- Obese subjects consistently under-report their caloric intake by a substantial degree [7].

More generally, it is the author's opinion that health is among the most precious of commodities a person can possess. Certain conditions can cause a large degree of suffering in those who are afflicted, and may be long-term in duration; treatments can often be very expensive, with some diseases not having any treatment or cure as well. To make matters worse, those close to the individual may be negatively impacted too, where additional stress and worry can be commonly observed. It is hoped that the findings in this thesis can be of use, however significant or marginal, in furthering the efficacy and wide-spread adoption of diet tracking applications; more generally, it is hoped that these findings can demonstrate and advance the perception that technology can have a large, positive impact on health.

A special mention is made that the approach in this thesis was heavily motivated by the contributions in [8]. However, while [8] focused on object detection and segmentation of food items, this work focuses more on image classification techniques, and in boosting the accuracy of single-item recognition; food items can be inherently difficult to classify, especially when using parameter-constrained models on mobile devices.

1.2 Contribution and Thesis Outline

This thesis tackles the problem of poor dietary health by exploring the improvement of on-device food recognition, to further facilitate improvements to the mobile dietary health applications that many currently rely on. There are several primary reasons for this approach: first, mobile devices are an ideal candidate platform on

which food recognition could be performed, and accommodated to a varying degree of lifestyles on a global scale, given its portability, ubiquity, and on-device camera; second, the accurate automation of manual food-entry in diet-tracking applications would not only improve overall quality-of-life, but also accuracy – primarily from the elimination of user-made errors; third, the quality-of-life improvements to existing applications would be non-trivial, and it may have a significant effect on reducing application abandonment for unspecified reasons, or for lack of application features [3]; fourth, the accurate-and-efficient implementation of food recognition may serve as a launchpad for automating other features, such as calorie estimation, that would further non-trivially enhance the quality-of-life of dietary health applications.

A large portion of this thesis focuses on CNNs for improving food recognition, which several other works have also explored; in addition, like this work, several of these also emphasize the importance of using mobile devices. However, they seldom discuss the prohibitively-high computational cost of standard CNNs, focusing most often instead on maximizing solely the accuracy at whatever cost, and rarer-still explore the computational constraints imposed by mobile devices; this work uniquely places a strong consideration on efficiency by exploring the use of DenseNet [9] and MobileNetV2 [10], two relatively-recent architectures that explicitly feature design elements aimed at maximizing both accuracy and computational efficiency – by utilizing components that also aim to minimize memory-usage and inference-time.

Another major contribution of this thesis includes a novel user habit system that utilizes dietary history – or user context – to further increase accuracy, by exploiting the uniquely regular and habitual nature of meal consumption. The proposed

approach leverages each user's dietary history – personalizing predictions according to that user –, and the top-5 classifier predictions, to make a prediction that is most likely for that particular user. The visual properties of food can make them extremely challenging to classify – best exemplified by how *coffee* and *Coca Cola* are indistinguishable – and so the proposed method is very important for addressing the challenges of these cases.

Finally, this thesis proposes guidelines for future effort by considering its approaches within the larger context of a diet tracking system; the goal of advancing food recognition is to, at least in part, enable a superior diet tracking application to what is currently available. In this regard, numerous features, like the aforementioned calorie estimation, are critical to its efficacy; these orthogonal areas-of-focus are deeply explored to identify both the scope of their challenges, and their approaches – present and prospective. Guidelines for future effort are then generated by contributing both domain-specific and -agnostic discussions based on these findings.

Regarding structure, the thesis is organized by chapter as follows:

- Chapter 2 aims to give background on CNNs, and discuss the DenseNet [9] and MobileNetV2 [10] architectures – which are employed in this work – at greater depth.
- Chapter 3 provides background on food recognition, discussing related works and highlighting CNN-based approaches that constitute the state-of-the-art. The class variation properties of food are also discussed in detail, to illustrate

how they make food recognition more challenging, and how they strongly motivate the proposed user habit system. Chapter 3 further discusses diet tracking systems at a higher level, identifying and deeply-exploring a subset of relevant orthogonal topics; while the thesis explores various methods for improving mobile food recognition, it does not discuss much how these tie into an overall food recognition system. This section complements the approach in this thesis by providing a reference starting point for the multidisciplinary effort that the inception of such a system demands.

- Chapter 4 focuses on the methodology of this thesis. It highlights how there is a lack of practical consideration and exploration in choice of CNN for mobile food recognition, and proposes various methods to maximize the accuracy of DenseNet [9] and MobileNetV2 [10]. It also discusses the rationale for the user habit system in greater detail, and highlights the proof-of-concept experiment that evaluates and quantifies its advantages. Finally, the runtime-performance evaluation on mobile devices is discussed for the proposed architectures.
- Chapter 5 provides the corresponding experimental quantification to Chapter 4, along with an appropriate discussion of the results.

CHAPTER 2
Convolutional Neural Networks

2.1 Introduction

Convolutional Neural Networks (CNNs) have seen a resurgence in recent times, outperforming all prior conventional methods in accuracy, and are largely considered state-of-the-art across a myriad of image recognition tasks. The DenseNet [9] and MobileNetV2 [10] architectures in particular are noteworthy in regards to food recognition, due to architectural designs that make them both suitable for mobile deployment and highly accurate at their parameter costs. This section has two primary aims: the first is to provide the reader with background on CNNs at a more general level by discussing architectures, components, and techniques that constitute best practices today – and is motivated by the rapid advancement of the state-of-the-art from year-to-year; the second is to analyze the design of DenseNet and MobileNetV2 at a deeper level, and highlight how it makes them ideal for performing inference on mobile devices. These architectures constitute the core CNN approach proposed in this thesis, and so it is hoped that the background provides sufficient context for their utilization.

2.2 Historical Overview

CNNs are multi-layered computational models that excel at learning representations in raw data, and their unprecedented accuracy has largely established them as state-of-the-art for almost all image classification tasks [6]. While they found their

success recently, the foundations of CNNs were developed decades ago. As the name suggests, CNNs are biologically-inspired pattern recognition architectures with basic principles drawn heavily from the works of Hubel and Weisel in 1962, whose experiments shed light on how brain cells process vision [11]. The findings, particularly regarding pattern recognition, were explicitly stated as inspiration by Fukushima in 1980, who introduced the Neocognitron – one of (if not) the first architectures resembling modern feedforward CNNs – as a means to understand how humans recognize patterns [12]; he created a model that closely resembles the modern CNN in how it is layered and organized, how it supports unsupervised learning, and how it captures patterns in a position-invariant way. Shortly after, in 1989-90, LeCun et al. demonstrated how real image recognition problems can be tackled with such networks; the backpropagation learning algorithm was incorporated, and a system was designed that could read handwritten digits in zip codes [13]. In 1998, gradient-based learning was incorporated in the LeNet CNN, and the system was implemented by banks to read checks following its successful employment in document recognition [14]. While these contributions helped demonstrate the capability of CNNs, the lack of computing power and training data proved to be large hurdles until recently, and so they largely fell out of favor. In 2004, graphics processing units (GPUs) were shown to significantly improve the speed of neural network implementations by efficiently performing parallel operations [15], specifically the vector and matrix multiplications that are critical in CNNs, and helped address the lack of computing power. In 2010, the first ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was ran [16] in which the public was challenged to minimize image classification error on

the large ImageNet dataset [17]; the competition provided a stage to showcase new developments, while the large dataset addressed the lack of training data.

In 2012, the AlexNet architecture [18] of Krizhevskiy et al. marked the first time a CNN was entered in the ILSVRC [16]; by significantly outperforming conventional methods – it had a top-5 error rate of 15.3% versus the runner-up at 26.2% – it played a large role in the renewed interest in CNNs. Since then, not only have results gotten significantly better – 2.3% error rate in 2017 ILSVRC – but the state-of-the-art has also been pushed in other areas such as object detection [19], speech recognition [20], and semantic segmentation [21]. Additionally, large companies like Microsoft [20], Google [22, 23], Facebook [24], and Amazon [25] have shown interest in practical systems with CNNs at the core, and have aided in reinforcing both their commercial and research relevancy.

2.3 Architectures

A quick search reveals that there are a staggering number of distinct CNN architectures in literature, with most advancing the state-of-the-art in some way such as lower error rate and improved memory efficiency. However, arguably the most influential have been the past ILSVRC entries: they provided key insights into how CNNs work, and improved both architecture – leading to record results with less memory and computations – and training techniques – most of which constitute best practices today. The main contributions of several notable entries will be examined to contextualize the development of DenseNet and MobileNetV2; specifically, it is hoped that the discussion demonstrates the evolution of the state-of-the-art, and how the various findings serve as a design foundation for the aforementioned architectures.

2.3.1 ILSVRC Entries

As previously mentioned, AlexNet [18] swept the 2012 ILSVRC, improving the top-5 error rate of the runner-up by 10.9%, and was instrumental in renewing interest in CNNs; as a result, although its architecture is largely outdated now, it is still a basis for many subsequent ones. It promoted the use of several components and techniques that are still effective today, including Rectified Linear Units (ReLU) [26], dropout [27], image augmentation, image pre-processing by subtracting mean activity, and softmax classification with cross-entropy loss. It also granted the base intuition for transfer learning [28] and further visualization efforts, by providing filter visualization for the general nature of shallower learned features.

The following year, Zeiler and Fergus [29] published key findings based on refinements for their 2013 entry. They eased the burden of training by successfully using substantially-less data augmentation, and greatly contributed to filter visualization with the Deconvolutional Network; the provided insight was instrumental to improving filter design, and the authors themselves improved the filter design of the first layer in AlexNet.

In 2014, two notable entries were made in VGGNet [30] – the runner-up, and GoogLeNet [31] – the winner. The former showcased the importance of depth by solely using 3x3 convolutions to maximize it, which in turn demonstrated their efficacy both in reducing parameters and increasing non-linearities for a given receptive field – a concept the authors introduce. Today, a ubiquitous use of 3x3 convolutions can still be seen, and VGGNet is still occasionally used as a core due to its relatively simple, but effective, design. Regarding GoogLeNet, its impact remains

in how it greatly utilized 1x1 convolutions [32] to improve computational efficiency by reducing the number of intermediate feature maps, and reduced the number of parameters by replacing fully-connected layers with global average pooling. It also introduced the inception module, a unique and efficient design that extracts features at different scales, which serves as a basis for high-performing architectures including ever-improving Inception derivatives [33], and Xception [34]. A final note is made that both entries demonstrated the efficacy of scale jittering in image augmentation, improving final accuracies by a notable degree.

The following year saw a huge breakthrough with ResNet [35], marking the first time a single model exceeded human performance [36] with a top-5 error rate below 5%. The authors developed and used residual skip connections to resolve the problem of degrading performance with increasing depth; they showcased the ability to go orders of magnitude deeper than anything to this point – exceeding 1000 layers – and achieved exceptionally-good results with various designs utilizing these skip connections. The motivating principle stems from explicitly re-mapping the function the network is expected to learn; instead of learning an underlying mapping of $H(x)$, it learns the residual. Specifically, the original mapping is reformulated as $H(x) = F(x) + x$, and the network is made to learn $F(x)$. The authors hypothesized that this reformulation may be easier for the network to learn, and possible in that if the former mapping can be asymptotically-approximated by the network, so too should the latter. The performance was further improved using pre-activations [37], especially for the over-1000-layers design; the authors additionally demonstrated the importance of identity mappings in these skip connections, an important finding in

recent literature. Interestingly, despite the ubiquitous use of these skip connections today, the exact reasoning of why they are so effective remains elusive; there are many claims [38, 39, 40] that are contrary to the authors', in particular that improved gradient flow is an unlikely explanation.

The 2016 ILSVRC saw largely refinements relative to the prior year with ResNeXt [41] and Wide ResNet [42]; their main contribution was improving performance through increased width, or the number of filters per layer, due to the diminishing gains of depth. In particular, the former introduced cardinality – splitting the number of filters in a layer evenly into multiple branches, similar to inception – to outperform naïve increases to either depth or width; the authors claim that cardinality allows layers to learn more powerful representations, and show superior performance in non-residual counterparts as well. Finally, in 2017, the winning entry utilized Squeeze-and-Excitation blocks [43] as a modular and cost-effective way of increasing the representational power of a network; the authors incorporated them into architectures like ResNet [35] and MobileNet [44] to achieve state-of-the-art performance at small computational increase, superior to increasing either depth or width.

2.3.2 Components and Methodologies

The namesake of CNNs, the convolutional layer is at the core of any architecture, and learns effective feature extractions using trainable weights – also called kernels or filters – that are slid across the input volume; though it typically involves taking dot products between the weights and the input patch, alternatives such as octave convolutions [45] are being proposed to improve computational efficiency. For filter size, most architectures usually have a larger initial one to extract more expressive

initial feature maps, with the rest of the network using 3x3 filters to maximize depth, non-linearities, and minimize parameter usage for a given receptive field [30]; 1x1 convolutions [32] are also commonly seen to cheaply increase depth [35], or increase computational efficiency by reducing the number of input feature maps [31].

Most architectures halve the height and width of feature maps at deeper layers using either stride-2 pooling [14] or convolutions [35]; the number of filters in subsequent layers are then doubled to maintain a similar level of complexity throughout. Although pooling is ubiquitously seen, and is relatively cheap to compute, there are discussions that pooling may be unnecessary and serve to needlessly complicate network architecture [46]. Another important component to almost all architectures is batch normalization (BN) [47], a trainable layer that performs intermediate normalization; they significantly improve network performance at minimal or no cost, as the learned parameters can often be folded into preceding convolutional layer weights [42]. Global average pooling [31, 32] has also largely replaced fully-connected layers [18, 29] to drastically improve parameter-efficiency, and is usually included once prior to the final softmax layer.

Explicit inclusion of non-linearity is critical to increasing the representational capability of a network, as it is only capable of learning linear features otherwise. The Rectified Linear Unit (ReLU) [26] is by far the most favored due to its often strictly-superior performance [18] over the conventional hyperbolic tangent (tanh) [48] or sigmoid [49]; it also mitigates the exploding and vanishing gradient problems [50, 51, 52] which the latter two are susceptible to, with non-saturation often argued as the primary reason why. While largely advantageous, ReLU neurons can potentially fail

if they begin to output zero, since it would be incapable of updating its weights from the gradient also being zero [53]; thus, alternatives like leaky [53] and parametric [54] ReLU have been suggested. Despite these suggestions, ReLU remains the most popular choice, and has empirically continued to perform reliably and effectively.

Beyond architecture, many training techniques have been proposed to mitigate the difficulty in acquiring the best possible set of weights for a target task; in addition to convergence challenges, CNNs have a tendency to overfit [18], and so significant efforts have been made in ensuring a generalized set of weights. Data augmentation has proven very effective, and is almost always used; common ones include a variety of photometric distortions [18], and scale variation [30, 31]. Image pre-processing is typically minimal, with the most common being the subtraction of mean activity per pixel to "center" the data [18, 35]. Numerous regularization approaches have also been recommended to address overfitting, including dropout [27] – where neurons are randomly deactivated – and L1/L2 regularization [55] – to regulate weight updates. Although it is argued that BN somewhat marginalizes the necessity of regularization techniques, they continue to prove effective, and newer methods have developed such as DropBlock [23] and stochastic depth [56] – which can also substantially reduce training time.

Convergence is another important topic, and without proper initialization, many networks struggle to converge well; as such, schemes have been suggested to avoid poor convergence, or in some cases no convergence at all [51, 54]. Recently, transfer learning – where a network is initialized with previously-trained weights – has proven extremely effective at accelerating and improving convergence [28], and is widely

recommended in all cases that the option is available; it is quite common as there is a high availability of ImageNet pre-trained weights in circulation, particularly for most popular architectures like ResNet [35]. Furthermore, it drastically reduces subsequent training time as the available weights are usually well-converged, as they are usually derived from extensive training sessions; they also usually facilitate a more generalized set of final weights, and the fact that most weights are for a different task is typically advantageous. The base intuition stems largely from visualizations that initial features are often more generic [18, 29], while deeper features are more specific to the original task [57]; as such, while one may expect fine-tuning on the target task to improve performance – especially for the deeper weights, transfer learning has proven effective even in cases where the weights remain frozen, and is sometimes advantageous as the network is more generalizable.

A final thing to discuss is in various training process optimizations; it is ubiquitous to see training data processed in batches and mini-batches for time-efficiency, and the use of an algorithm to optimize the weight update process for better convergence. Updates using vanilla gradient descent, in conjunction with momentum [58], consistently performs well [18, 35], and usually serves as a basic optimizer; adaptive methods in root mean square propagation (RMSprop) [59], AdaGrad [60], and Adam [61] have also seen increased adoption, and success on certain tasks [44]. However, gradient descent remains the most consistent as the adaptive methods have been criticized for having marginal value due to not generalizing as effectively [62].

2.4 DenseNet

The DenseNet [9], which won best paper at the 2017 Conference on Computer Vision and Pattern Recognition (CVPR), is noteworthy not just for its superior accuracy over ResNet [35], but also its superior parameter-efficiency. While some prior approaches considered efficiency, most focused solely on maximizing accuracy [63]; for example, many authors utilized techniques like ensemble methods and multi-crop testing [35] when reporting their results, and although effective, they are highly impractical and expensive. There are several unique design elements to the DenseNet that facilitate its high accuracy and efficiency; this section aims to highlight these, and contextualize the choice of DenseNet-121 for the proposed food recognition approach.

2.4.1 Dense Connectivity

A key contribution to the success of DenseNet is its dense connectivity structure, illustrated in Figure 2–1. The core intuition involves maximizing efficiency through re-using intermediate feature maps as much as possible, and is achieved by concatenating all preceding feature maps at every intermediate layer. The authors were heavily inspired by the success of ResNet [35], specifically with how identity connections were instrumental to its success [37] – a finding that is well-corroborated [39, 40]; by extending the idea, the authors aimed to further improve gradient flow, and so ease the training process. In addition to improving feature re-use and reducing redundancy [56], the dense connections improve parameter efficiency by allowing layers to be made thinner – or in other words, reducing the number of filters per

layer. Finally, the authors note that the resulting architecture has a regularizing effect, and they discuss how it generalizes better on smaller datasets.

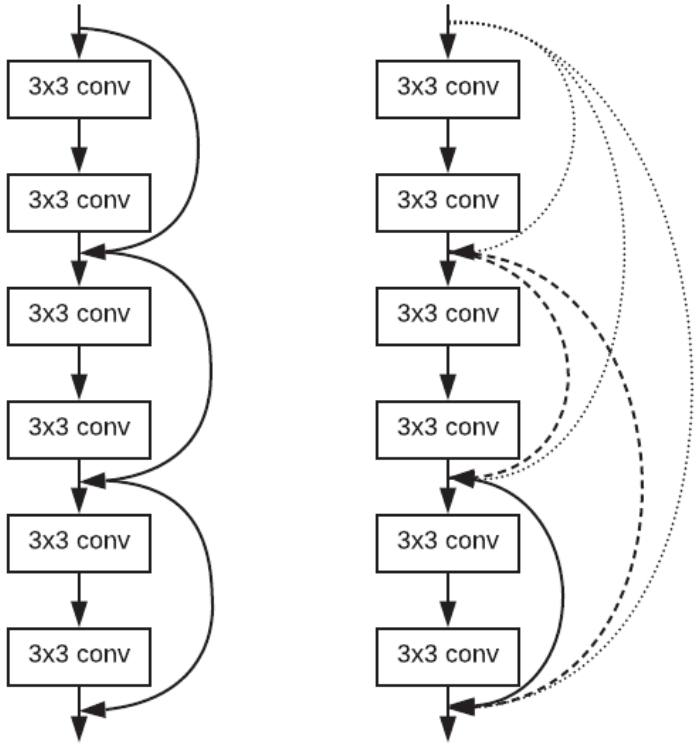


Figure 2–1: Illustration of dense connectivity structure. **Left**: Residual connections for a residual block, where the feature maps are added after feeding-forward a single time. **Right**: Dense connections, where the feature maps are concatenated and fed-forward to every subsequent block; note that these conv layers are usually thinner – have less filters – as constant concatenation substitutes for the reduction in number of feature maps.

2.4.2 Growth Rate

Due to the concatenation of feature maps at each layer, one can think of the network as having a global state which progressively grows as each layer adds its

contribution to it. Thus, a growth rate hyperparameter, k, is defined, and is used to set layer width – represented by the number of filters – and regulate the rate at which new information is added to the global state.

2.4.3 Dense Blocks

Dense blocks are separated groupings of layers, which are connected using the aforementioned dense connectivity scheme. Layers are defined to be of width k, and implement three consecutive operations resembling pre-activation [37]: BN, ReLU, and 3x3 convolution. Bottleneck layers are also used to improve computational efficiency, and are comprised of a 1x1 pre-activation convolution layer of width $4k$, followed by the regular 3x3 convolution layer.

2.4.4 Transition Layers

Transition layers are used to separate dense blocks, and perform down-sampling of feature maps. A compression factor, $0 < \theta \leq 1$, is introduced as a means of reducing the number of feature maps at the output of a dense block; if the dense block produces m features maps, the following transition layer reduces it to $[\theta m]$ feature maps. By reducing the depth – or number of feature maps – , the network addresses computational blow-up that arises from constant concatenation of feature maps, which would otherwise be infeasible in practice.

Transition layers are composed of a 1x1 convolution layer that produces $[\theta m]$ feature maps, where θ is the compression factor, and a 2x2 average pooling layer of stride 2; these operations are used to reduce the depth, and both the height and width, respectively, of the input.

2.4.5 Overall Structure

For this thesis, the DenseNet-121 architecture is explored for its balance between accuracy and parameter-efficiency, based on ImageNet performance. The overall structure is depicted in Figure 2–2. The network has a growth rate $k = 32$, compression factor $\theta = 0.5$, and uses zero-padding to match dimensions. It comprises an initial 7x7 convolution layer of stride 2 and width $2k$, followed by a pooling layer to further reduce feature map height and width. This is followed by four bottleneck dense blocks, with 6, 12, 24, and 16 layers respectively, and are all separated by transition layers. The final classification layer implements 7x7 global average pooling to reduce feature map height and width to 1, and a fully-connected softmax layer to produce a class prediction. The particular pre-trained weights that are used in this thesis achieve a 25.03% top-1 single-crop error on ImageNet with 7.0 million parameters, excluding the softmax layer [64].

Layers	Output Size	DenseNet-121
Convolution	112x112	7x7 conv, stride 2
Pooling	56x56	3x3 max pool, stride 2
Dense Block (1)	56x56	$\begin{bmatrix} 1 \times 1 \ conv \\ 3 \times 3 \ conv \end{bmatrix} \times 6$
Transition Layer (1)	56x56	1x1 conv
	28x28	2x2 average pool, stride 2
Dense Block (2)	28x28	$\begin{bmatrix} 1 \times 1 \ conv \\ 3 \times 3 \ conv \end{bmatrix} \times 12$
Transition Layer (2)	28x28	1x1 conv
	14x14	2x2 average pool, stride 2
Dense Block (3)	14x14	$\begin{bmatrix} 1 \times 1 \ conv \\ 3 \times 3 \ conv \end{bmatrix} \times 24$
Transition Layer (3)	14x14	1x1 conv
	7x7	2x2 average pool, stride 2
Dense Block (4)	7x7	$\begin{bmatrix} 1 \times 1 \ conv \\ 3 \times 3 \ conv \end{bmatrix} \times 16$
Classification Layer	1x1	7x7 global average pool
		1000D fully-connected, softmax

Figure 2–2: DenseNet-121 structure, with $k = 32$ and $\theta = 0.5$. Each conv layer is an implementation of BN-ReLU-convolution, and the initial 7x7 convolution has width $2k$. Adapted from [9].

2.5 MobileNetV2

MobileNetV2 [10] is one of the few architectures designed specifically with mobile applications in mind; with this thesis exploring mobile food recognition, the architecture is of immediate interest. This section highlights the critical design elements that enable the success of MobileNetV2, and discusses the rationale behind the final choice of architecture.

2.5.1 Depthwise Separable Convolutions

Similar to MobileNet [44], MobileNetV2 replaces standard convolution layers with depthwise separable convolution layers [65] – depicted in Figure 2–3 – to dramatically increase parameter and computational efficiency. Convolutions are first factorized into a depthwise component followed by a pointwise component, and they learn to capture same-channel and cross-channel features respectively. To illustrate how the factorization closely approximates a standard convolution, consider how standard convolutions sample patches along the height, width, and depth concurrently; instead of performing these in a single step, the factorization breaks it down into two – where the depthwise convolution first samples along the height and width, followed by the pointwise convolution that samples along the depth. As previously mentioned, depthwise convolution layers are used to improve computational efficiency; to highlight the performance gains, the computational cost of standard, depthwise, and pointwise convolutions are given, respectively, by [44]:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

$$M \cdot N \cdot D_F \cdot D_F$$

where stride one and padding is assumed, D_K is the kernel size, D_F is the feature map size, M is the number of input channels, and N is the number of output channels. The computational reduction, represented by the ratio between depthwise separable

and standard convolutions, is thus given by [44]:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

Considering a kernel size of 3, there is approximately 8 to 9 times less computation with depthwise separable convolutions, with only a small reduction in accuracy [44].

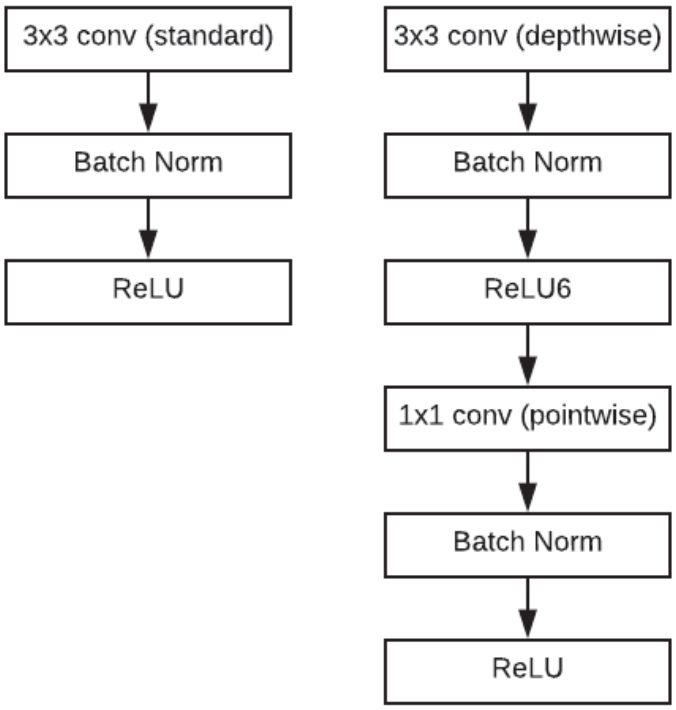


Figure 2–3: Comparison between standard and depthwise separable convolutional layers. **Left**: Standard 3x3 layer, with batch normalization (BN) and ReLU. **Right**: Depthwise separable 3x3 layer, which first performs 3x3 depthwise convolution followed by 1x1 pointwise. The former is followed by BN and ReLU6 [66]; the latter by BN and regular ReLU. Adapted from [8].

2.5.2 Linear Bottleneck

Bottleneck layers typically utilize 1x1 convolutions [32] to improve computational efficiency by reducing the depth of an input volume [31, 35]. Most approaches also use these layers to introduce more non-linearities in the network – including a ReLU either before or after the convolution – to increase its overall representational power. However, the authors [10] claim that the additional non-linearity is actually detrimental to performance, and instead opt for linear bottlenecks by removing the ReLU. In addition to empirical evidence [10, 67], they also provide a supporting explanation; generally, one can think of bottleneck layers as transforming the input information such that it is represented in a lower-dimensional subspace, where the subspace has a lower depth. When the difference in depth is sufficiently large, the use of ReLU – which sets all negative values to zero – can instead destroy a significant amount of information such that it leads to impaired performance. Interestingly, they note that the reduction is less noticeable when shortcut connections are present.

2.5.3 Inverted Residuals

The ground-breaking results of ResNet [35, 37] demonstrated the importance of identity shortcut connections in improving performance; thus, they are also used in MobileNetV2. However, instead of conventional residual blocks, the shortcut connections are made between bottleneck layer outputs instead, as depicted in Figure 2–4; these connections not only improve the memory efficiency of the network, but also the accuracy [10]. The primary motivation stems from the fact that the reduced,

linear output of the bottleneck layers contain all the necessary information with respect to their unreduced input, and so the inverted residual block can be used to greater success.

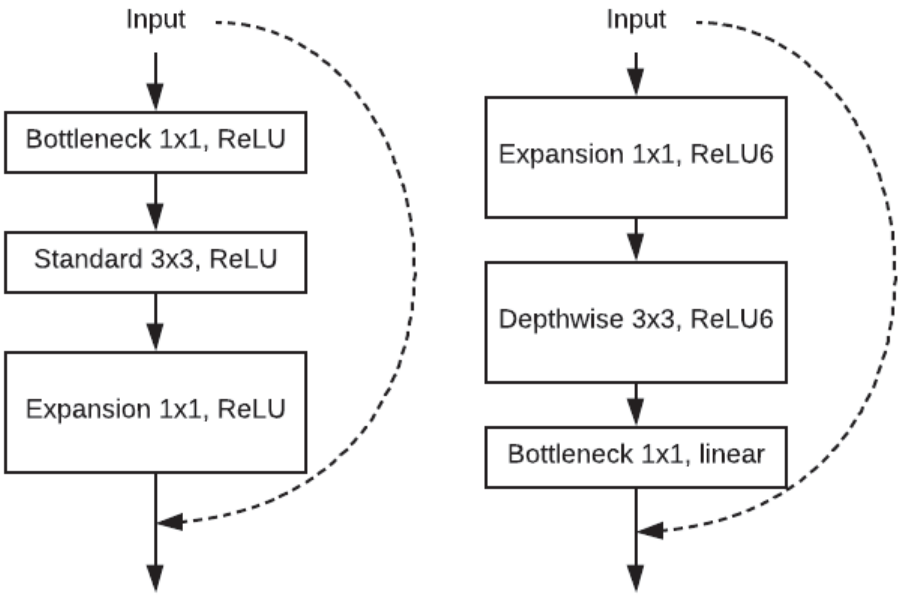


Figure 2–4: Comparison of standard and inverted residual blocks. **Left**: Standard block featuring identity shortcuts between expansion outputs. **Right**: Inverted block featuring identity shortcuts between bottleneck outputs.

2.5.4 Width Multiplier

A width multiplier, α, is introduced – originally used in MobileNet [44] – to modify the output depth of each layer to αN, where N is the original depth; another view is that the width multiplier adjusts the number of filters per layer. Reducing the width has the effect of increasing computational and parameter efficiency by approximately α^2 [10], but at the cost of accuracy; likewise, increasing the width

improves accuracy at the cost of efficiency. The original network is retained with $\alpha = 1$.

2.5.5 Structure

MobileNetV2 uses bottleneck residual blocks as its primary building block; it first performs 1x1 convolutions to expand the depth from k to tk – where t is the expansion factor, followed by 3x3 depthwise convolutions to capture same-channel features, and final 1x1 convolutions to both capture pointwise features and apply bottlenecks. The former two introduce non-linearity specifically with ReLU6 [66] instead of ReLU, for its robustness in low-precision calculations [44], while the latter remains linear as discussed in Section 2.5.2. The full structure is depicted in Figure 2–5; excluding the very first layer, the network is built using bottleneck residual layers, with all but the first one using an expansion factor $t = 6$. The first layer in each sequence of layers is responsible for downsampling via strided convolution.

For this thesis, the network is as discussed but with a width multiplier $\alpha = 1.4$ applied; the higher width allows for greater accuracy that makes it more comparable to DenseNet-121. The pre-trained weights that are used achieve a 24.77% top-1 single-crop error on ImageNet with 4.4 million parameters, excluding the softmax layer [64].

Input	Operator	t	c	n	s
224x224x3	3x3 conv	-	32	1	2
112x112x32	bottleneck	1	16	1	1
112x112x16	bottleneck	6	24	2	2
56x56x24	bottleneck	6	32	3	2
28x28x32	bottleneck	6	64	4	2
14x14x64	bottleneck	6	96	3	1
14x14x96	bottleneck	6	160	3	2
7x7x160	bottleneck	6	320	1	1
7x7x320	1x1 conv	-	1280	1	1
7x7x1280	avgpool 7x7	-	-	1	-
1x1x1280	fully-connected	-	num_classes	-	-

Figure 2–5: Full MobileNetV2 structure, adapted from [10]. t denotes the expansion factor, c the number of output channels, n the number of layers in the sequence, and s the stride of the first convolution in the sequence of layers. The depicted structure has width multiplier $\alpha = 1$.

2.6 Frameworks

Numerous frameworks are available for developing and training CNNs, including Caffe [68], TensorFlow [69], Keras [64], and PyTorch [70]. TensorFlow has been a notably popular framework, developed by Google, and it implements many performance optimizations through its focus on building static computational graphs; as such, it is also well-suited for distributed training paradigms. However, many have felt that the barrier to entry is quite high, and the static, low-level nature makes it relatively difficult to debug and iterate quickly; as a result, Keras – a higher-level API that often functions as a TensorFlow wrapper – has risen in popularity for its ability to greatly simplify the process of creating and training a network – to the extent that TensorFlow has further built on top of Keras. PyTorch, which is developed by Facebook, has also recently risen in popularity, and its focus on dynamic graphs eases the debugging process and facilitates faster iteration; that said, TensorFlow now also has an eager execution mode which runs more dynamically, and offers similar benefits to PyTorch.

While there are implementational differences between the various frameworks, the choice was widely thought to come down to personal preference, with many claiming that there were only small differences in either the training process or network performance at inference time. However, recent benchmarks [71] have revealed that there are notable performance differences between them, depending on the hardware and architecture used; while such differences may be relatively unimportant for research purposes, it can matter in real-world deployment – especially if the CNN is used for real-time classification. Additionally, various hardware may further restrict

choices due to lack of compatibility, such as with Google's tensor processing units (TPUs) [72] – although support is growing for other frameworks like PyTorch and Keras, and not just for TensorFlow. Finally, it is important to consider that not all frameworks officially support mobile deployment, making it prudent to choose a framework that would simplify the process. TensorFlow Lite [73] supports deployment and inference on both Android and iOS devices, while CoreML [74] supports it on iOS; as such, the model should be convertible into these formats when developed using the chosen framework.

CHAPTER 3
Food Recognition

3.1 Introduction

The World Health Organization (WHO) found that diet is a key factor in numerous complications including obesity, diabetes, and cardiovascular diseases [1]; as such, a significant effort has been made in studying how to encourage people to adopt healthy diets. In particular, interest has been shown in understanding the potentially-positive effect of technology on managing these conditions, and numerous studies have been conducted to analyze how to maximize user adoption of associated applications [3, 75, 76]. In the specific case of diet-tracking applications, there is a prevalent reliance on manual self-reporting to record the food intake of users [77, 78], which unfortunately introduces a significant source of human error; the result is large inconsistencies from potential misinputs of measurements, typos, or simply no inputs at all, all of which can drastically reduce the efficacy of the associated applications.

As a result, food recognition has long been of interest to the scientific community, largely for its potential in improving both the performance and overall quality-of-life of diet tracking applications. Two general types of approaches can be commonly seen: one focuses largely on object detection methods to facilitate fast, accurate classification of numerous food items in a single image [8]; the second – which this thesis explores – focuses on improving the accuracy of single-item classification, which in turn serves as a backbone for the former. While convolutional neural networks

(CNNs) have greatly improved food recognition accuracy, most approaches have not fully considered their employment with mobile devices in mind – particularly in terms of runtime performance and efficiency. Thus, the following sections have several aims: the first is to provide context and background by discussing various works that demonstrated state-of-the-art performance using CNNs; the second is to highlight the difficult nature of food classification given inherent class variation properties; the third is to analyze, at greater depth, a variety of research areas – their current and future approaches – that would facilitate a wide-scale improvement to dietary health. Regarding the second in particular, it will be demonstrated in Chapter 4 how the proposed user habit system – which incorporates contextual user information to improve classification accuracy – is highly effective at addressing the problem, and how it has the potential to yield substantial accuracy gains.

3.2 Related Work

Prior to the resurgence of CNNs, food recognition – and image classification in general – relied significantly on the use of hand-crafted methods for feature extraction; Scale Invariant Feature Transform (SIFT) [79], Fisher Vectors (FV) [80], and bag-of-features [81] could be commonly seen in conventionally-successful approaches [82, 83, 84, 85], and a large problem pervaded these efforts in that there exists a strong dependence on domain-specific knowledge to maximize the strength of these extracted features.

Following its success in 2012, many in the food recognition community began taking note of the efficacy of CNNs in not only learning to extract strong features

automatically, but in using them to outperform manual methods in classification accuracy. With transfer learning [28] – and the release of public food datasets including Food-101 [86], UEC Food-100 [87], and UEC Food-256 [88] – serving as enablers for further investigations in the food recognition domain, numerous approaches have since been proposed that demonstrated the efficacy of CNNs over these traditional, manual methods. In [89], the AlexNet [18] architecture was used to improve classification between ten food classes acquired from public images, while in [90], it served as a superior feature extractor, on both UEC datasets, for training one-vs-rest linear classifiers; its effectiveness was further improved following pre-training on an extended 2000-class subset of ImageNet [17].

With the efficacy established, more approaches shifted focus toward other CNN architectures for improving accuracy: GoogLeNet was used in [91] to improve accuracy on the UEC and Food-101 datasets; the Inception-v3 architecture [33] pushed accuracy even further in [92] for these same datasets; Nutrinet [93], a CNN architecture specific to food, was inspired by AlexNet, and showed competitive results with other architectures like ResNet [35] on a custom dataset; the wide ResNet [42] was adapted with a wide-slice branch in [94], comprised of a domain-specific design aimed at capturing layered food traits, and was coupled with ten-crop testing [18] to further improve accuracy on the UEC and Food-101 datasets.

In tandem to these efforts, increasing interest is being shown toward utilizing mobile devices for food recognition [82, 83, 91, 95, 96]; these devices are ideal for the task as they are typically equipped with a camera, have increasing computational capacity, and are both ubiquitous and portable. Although the aforementioned works

Figure 3–1: Wide-slice Residual Network, adapted from [94].

Figure 3–2: Examples of food items in the UEC Food-100 dataset [87] comprising a layered structure

successively utilized novel CNN architectures to improve accuracy, they do not emphasize the importance of computational efficiency – which is important to consider given the computational limitations of mobile devices. To illustrate, while the wide-slice residual network [94] substantially improves upon accuracy, it does so at a large increase in parameters and computational operations, as well as the employment of ten-crop testing. These are highly demanding options, and the limited capabilities of mobile devices render them infeasible in most practical scenarios. Nonetheless, great strides continue to be made in the domain, such as with the release of new datasets like ChineseFoodNet [97] and the consideration of food recognition systems at a larger scope [95, 98].

3.3 Challenges in Class Variation

Food items are inherently difficult to classify due to their class variation properties. First, to highlight the issue of low intra-class variation [94], oftentimes the only difference between *burger* and *cheeseburger* is a single slice of cheese; without proper training, representational power, or computational capacity, a CNN may be

unable to consistently differentiate between these classes. Even if the classifier is sufficiently trained to do so, the high similarity between these classes may lead to a top prediction that is marginally incorrect, leading to an overall guess that is incorrect. However, by far the most problematic consideration is that certain items are visually indistinguishable, such as *coffee* or *Coca Cola*, making it virtually impossible for the CNN classifier to differentiate these without an external form of accommodation.

Second, to highlight the issue of high inter-class variation [93], numerous meals and dishes – represented as classes in CNNs – can be cooked in a myriad of differing ways, and a single class can thus exhibit a large amount of variation in its representation. As such, there is a need to account for numerous variations of a single class, and having to account for all of them is exceedingly difficult; it is almost impossible to train on all the different home-cooked representations of numerous meal classes – such as *stew* or *sandwich* – and the difficulty is accentuated by how such representations are constantly evolving with time.

Thus, it is clear that a scheme is required to address these class variation properties if large advances are to be made in food recognition accuracy. Personalization constitutes one such scheme, and the nature of meal consumption makes it more-easily exploitable; by using contextual information – such as the user's dietary history – to limit the expected classes to those that are common for the particular user, it is less likely that the classifier incorrectly predicts the depicted food item. The user habit system proposed in Chapter ref4 demonstrates the potential efficacy of such a scheme, and presents a step toward a more ideal system that is generalizable to a myriad of different lifestyles.

3.4 Diet-Tracking Systems

While this thesis explores various avenues for improving mobile food recognition, it is important to reiterate that it is just one facet of facilitating dietary improvement; current approaches rely primarily on mostly-manual diet tracking processes, which are not very effective – as previously discussed [7, 75]. Thus, it is likely that several critical features – including automatic calorie estimation and mobile food recognition – will need to be consolidated in order to sufficiently-improve diet-tracking procedures, such that they facilitate larger-scale dietary improvement. In exploring current approaches, there is an evident trend that diet-tracking applications, such as MyFitnessPal [77] and MyFoodDiary [78], are seeing some widespread use and adoption; this observation reinforces the notion that automatic food recognition alone may be insufficient in creating a large impact. In addition, there are several considerations – such as the scalability of the proposed classifiers, or the emergence of potentially-enabling technologies like edge computing [99] – that are important to discuss when considering practical diet-tracking systems, due to a high likelihood of them being vital to practical implementations.

This section will demonstrate how the challenge of building an ideal diet-tracking system is ongoing, multi-faceted, and arduous; it will also alleviate some of the challenge by identifying significant works – across a variety of domains – that are critical to enabling future diet-tracking systems. Specifically, this section discusses how these advancements are excellent starting points for future work in each respective domain, and contributes its own practical considerations and recommendations that should be kept in mind when moving forward.

3.4.1 Calorie Estimation

Calorie tracking is necessary to all current diet tracking applications; as such, automatic calorie estimation is of interest as it pairs naturally-well with automatic food recognition, and alleviates the general tedium associated to current calorie-tracking processes. In addition, the automation of calorie estimation also eliminates sources of human error, similar to the case of food recognition. The feature is itself a composite of several others, which will be discussed below.

Volume Estimation

To estimate the calorie content of a depicted meal, a common method involves pairing its apparent volume with its dietary information to calculate the corresponding caloric value; however, the process of volume estimation remains very challenging. Most current methods require a 3D reconstruction of the food item to facilitate volume estimation, which typically requires additional steps than just a single image: DietCam [82] mandates the use of three images spaced about 120 degrees apart; [100] defines templates from pre-defined models of common shapes like cylinders, and specific irregular shapes like pears, to apply to the depicted food item; [101] utilizes a video recording to extract multiple views of the food at different angles; and [102] requires two images, in addition to the food on an elliptical plate and a credit card-sized reference next to it. A commonality among these methods is that the extra-step required is some form of external calibration, which adds a layer of inconvenience to them; in this regard, [96] takes a large step toward mitigating inconvenience by demonstrating how a mobile phone alone is sufficient for calibration. However, with that said, trade-offs still have to be made as very specific steps are required, and there

still exists inconvenience to the end-user; however, these nonetheless serve as a starting point toward an ideal implementation, and effectively highlights the continued efforts to push the efficacy of volume estimation techniques.

Nutritional Information

As previously discussed, automatic calorie estimation is widely considered a two-part process of pairing volume and nutrition information; as such, a food database is vital for retrieving the latter for a myriad of food items. Numerous entities including the Food and Agricultural Organization of the United Nations (FAO) [103], the United States Department of Agriculture (USDA) [104], and Health Canada [105], have published official nutritional information based on their own findings; additionally, commercial databases such as those by Nutritionix [106] and MyNetDiary [107] are also provided, and can be accessed via provided application programing interfaces (APIs).

Although nutritional information is widely available, there are several challenges to utilizing it properly; for one, the nutritional content of food can vary depending on the region and country from which they are sourced. Additionally, some classes such as *cheeseburger* can be composed of a differing variety of ingredients, which can complicate accurate-estimations due to the need to account for multiple valid representations of a single class. That said, these issues may not be significant for the purposes of a rougher estimate – or for loosely keeping track of calorie consumption – especially when considering the problematic nature of manual entry [7]; however, some users may still prefer manual correction for more fine-tuned control, and a consideration should be made in the implementation of post-estimate modifications.

Another effective option may include pre-defining numerous caloric values for different incarnations of a class of food, giving the user more control should they desire without introducing problems from manual entry.

Finally, there is consideration to be made in how database queries may lead to a large latency bottleneck, and a dependence on the availability of mobile data – which a mobile food recognition approach aims to avoid, or at least mitigate. A strategy then could be to implement an offline database of the classifiable food items directly on the mobile device – perhaps saving the data as text in an easy-to-parse format such as eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) – and periodically revise it through application updates; although a full nutritional database may be extremely large, a limited selection constituting classifiable foods is likely lightweight, and can feasibly be locally-stored on the device itself.

3.4.2 Classifier Scalability

An ideal classifier should be capable of recognizing any depicted food item, leading to an implication that it be capable of discriminating between thousands of classes comprising all food items and dishes across the globe – spanning a variety of different cultures and countries; for instance, when considering just Chinese, Indian, and Italian cuisines, each has hundreds if not thousands of different dishes, not taking into account basic items like fruits and vegetables, while further not accounting for the numerous valid representations of a single meal. As the number of classes increases, so too may the classification difficulty; after all, a CNN aims to learn an accurate representation of the underlying relationship in a given task, and it is reasonable to assume that this gets more difficult as the number, and variation, of

classes increases. In addition, from the results provided in Chapter 5, it is likely that there may be a limit in the complexity a parameter-efficient CNN may be capable of learning; as such, there is a trade-off to consider between increasing the capacity of the network to accommodate the increase in difficulty, and reducing computation to keep the network efficient.

As a result, it is prudent to investigate other avenues to extend the capabilities of mobile classifiers. First, there is the straightforward consideration of developing even more efficient architectures, aiming to improve upon current ones such as MobileNetV2 [10], and increasing the number of parameters to accommodate the increased difficulty in learning representations. Second, a post-classification scheme such as the proposed user-habit system (see section 4.5) can bolster accuracy without relying as heavily on learned-representation strength. Third, classes can be carefully curated and bundled such that they are sufficiently discriminable, to perhaps reduce the complexity of the learnable-representation; for example, all *burgers* and all *rice*-based dishes could be trained into their own single classes, reducing the need to discriminate between a large number of similar-looking classes. Lastly, several specialized classifiers could be trained for pre-defined subsets of food categories, such as one for *fast food* and another for *Chinese* dishes, to once again manage the underlying complexity; however, it is non-ideal in that a large amount of storage may be needed for numerous sets of specialized weights.

A final note is made to acknowledge the difficulty in actually acquiring a sufficient amount of training data, and curating them appropriately. A manual approach is very time-consuming and expensive, while a programmatic method – where images

are searched up for pre-chosen classes – may likely come with noisy images, and the requirement of legal procedures to ensure the usability of the chosen images. Perhaps an ideal method involves the union of manual and programmatic approaches, with varying amounts of each depending on the target use case.

3.4.3 Continuous Training

To briefly expand upon classifier scalability, the food classifier needs to be constantly trained to maintain, or even increase, its accuracy; as such, it is prudent to consider schemes that can continually train said classifier. A powerful server may be utilized, equipped with the appropriate hardware – such as a cluster of GPUs, for example – that would come with a large, one-time cost; however, the server might also need to be maintained, and may come with a longer-term monetary and time cost. A cloud-based approach like Google Cloud [108] – which also provides access to TPUs [72] for considerable training-time speedups – may also be taken, but may be more expensive in the long-term; there may also be associated privacy concerns when keeping data in the cloud. Regardless of the approach, weight updates can be periodically made at pre-determined training epochs, and deployed to users through application updates.

3.4.4 Object Detection

In an ideal scenario, a user should be able to take a single picture of their meal – comprising all its dishes and side-dishes – and have each component accurately recognized and classified. As such, it is prudent to investigate various object detection methods, which is critical to enabling such functionality [8]; the approach proposed in this thesis is highly relevant as it can be very-naturally extended to substantiate

the core, or backbone, of most object detection methods. Two-stage methods such as Faster R-CNN [109] and R-FCN [110] have typically exhibited high accuracy on various object detection datasets, and are also computationally-improved compared to older methods [111]. On the other hand, single-stage methods are usually much more computationally-efficient, and may thus be better suited to mobile deployment; that said, while YOLO [19, 112] and SSD [113] are significantly faster and more lightweight than two-stage methods, they do suffer a bit with reduced accuracy. More recently, SSDLite [10] has been proposed to further push computational efficiency at additional, marginal accuracy cost, while RetinaNet [114] sacrifices some of the computational efficiency of single-stage methods for accuracy comparable with two-stage methods.

Although object detection naturally extends the food recognition capabilities of the classifier, consideration must be made with respect to volume estimation, where most proposed approaches are specific to a single food item. The presence of multiple food items may complicate the procedure, and as such, it would be necessary to investigate schemes to extend volume estimation by adapting them to multiple items; a straightforward method involves performing volume estimation for each isolated item following their segmentation.

3.4.5 Dietary Analysis

While diet tracking constitutes the core of food diary applications [78], they usually provide a host of other attractive features, which users may find enticing, that also serve to facilitate increased usage and adoption of said application. As these features presently are, they typically perform some form of dietary analysis which

serves to provide metrics, encouragement, and goals to the user; it is readily-apparent that all such features would naturally extend the capabilities of the improved diet tracking system as previously discussed. To illustrate, caloric intake reduction could be calculated, and presented to the user on a week-by-week basis; the availability of such a metric, and consequently its improvement over time, may facilitate the encouragement of users who take proactive steps to improve their diet. Similarly, another example includes healthier food recommendations that may be provided to users based on their consumption habits, to facilitate an easier transition into a healthier diet; however, more influential recommendations like this should be done with care, and likely by a nutritionist or someone similar, as such recommendations should be made with caution, and dietary norms in mind.

3.4.6 Edge Computing

Mobile devices are inherently limited in computational capacity due to their portable nature, and so cloud computing is a natural consideration when deploying a CNN classifier; in this thesis, it was noted that both the increase in latency, and reliance on data-availability, are undesirable, while the accuracy and parameter-efficiency of the proposed CNNs makes them a superior option. That said, increasing interest in edge computing [99, 115] and mobile edge computing [116, 117] reveals a lower-latency alternative, as it aims to bring cloud-like computational capabilities closer to network edges, and the finding that parameter-constrained networks may be too limited in representational strength opens consideration toward the possibility more. When coupled with the need to ensure the scalability of the classifier, an edge computing-based paradigm may offer sufficient advantages over mobile deployments.

The benefits of edge infrastructures are largely the same as the cloud: there is increased computational budget that enables the use of not only larger CNNs, but also expensive methods like ensembles and multi-crop testing to boost accuracy; the potential availability of specialized hardware may lead to faster inference times by optimizing numerous CNN operations; benefits may be observed in battery life by offloading a good portion of computations; and the dietary history of a user may be simultaneously stored in the cloud when sending pictured meals to the server for recognition.

With that said, there remain problematic aspects to the approach. First, it is more expensive than the embedded approach, which may only have cost associated to training the CNN; in this scenario, there is additional cost both in terms of data usage for the user, and server upkeep for the service provider. Second, the reliance on network-availability is re-introduced, and it may be a burden on the user to maintain; the issue of increased latency may also remain if the inference time is not sufficiently reduced, or if the edge server is still too physically distant from the network edge. Third, security concerns are introduced as personal data needs to be sent to the server for classification. Finally, there is a burden on the service provider to ensure that numerous user requests may be simultaneously served as their devices will no longer be used for classification; the greater concern is when considering how meal consumption creates periods of high network traffic – like lunch or dinner – while remaining almost non-existent at other times. As such, it is likely that further research needs to be done before opting into an edge-based approach.

3.5 Summary

This section provided context and background for food recognition, and discussed various other CNN-based approaches; the inter- and intra-class variation properties of food items were also highlighted, along with how they lead to inherent classification difficulties. In addition, future orthogonal areas of focus were also discussed in how they may benefit diet-tracking as a whole, and fully leverage the advantages that automatic food recognition enables. There are several things to note before proceeding to the following chapters: the first is that the wide-slice structure [94] is further explored within the context of mobile food recognition, as it achieved the highest accuracy on the UEC Food datasets [87, 88] and is purposefully designed to learn layered features inherent to many food classes; the second is that the class variation properties serve, in part, as motivation in the development of the user habit system. To elaborate on the latter, it is likely that the classifier's accuracy may be misleadingly low due to incorrect classifications by a small margin; in other words, although the correct class may not be its most confident guess, it is likely present among the top ones. As such, with user context to facilitate, there is a very accessible option to not only simply, but effectively improve classifier accuracy; this observation is leveraged by the user habit system, and is discussed in greater detail in Chapter 5.

CHAPTER 4
Design and Methodology

4.1 Introduction and Overview

The convolutional neural network (CNN) is now core to almost all image classification tasks, due to its outperforming of conventional methods by a significant margin; some of these previous methods include Scale Invariant Feature Transform (SIFT) [79] and Fisher Vectors [80] – for feature extraction and representation [118] – and support vector machines (SVMs) – for classification [119]. Although CNN architectures, and training techniques, are studied extensively in literature, relatively little has been done in the context of food recognition. Current methods for diet regulation – such as manual-entry food diaries [78] – have been ineffective at combating dietary health concerns that only continue to grow [5]. As such, there is a clear need for novel improvements to current approaches, which food recognition facilitates with its potential to drastically reduce user error, and improve quality-of-life, for associated applications (see Chapter 3). In particular, there is much interest in exploring mobile food recognition [82, 83] due to the ubiquity of mobile devices – and their access to a camera, portability, and modest computational capabilities.

Before proceeding, a special note is made that for this chapter, the denotation of "I" will be used to indicate specific contributions made by the author; the purpose is to explicitly highlight author contributions, and to avoid ambiguity that may result from third-person phrasing.

That said, while numerous efforts have emerged to demonstrate how CNNs considerably improve food recognition accuracy [91, 92, 94, 93], there is scarce focus on mobile feasibility; specifically, CNN approaches are, often and inherently, computationally-expensive to implement, and there are considerations – such as runtime efficiency, or optimal performance on less-powerful hardware – that need to be appropriately made. Thus, I identify and focus on several of these considerations, and devise various methods to address them. First, I propose a user habit system to improve the accuracy of food classifiers; to briefly elaborate, I recognize and explore the potential of user context for addressing extreme difficulties – which others have noted [93, 94] – that may arise in food recognition, such as with visually-indistinguishable classes like *coffee* and *Coca Cola*. Second, I propose the use of DenseNet [9] and MobileNetV2 [10] architectures for mobile food recognition, to improve the parameter-efficiency of CNN-based classifiers without losing much accuracy – which the aforementioned user habit system further supplements; current approaches rarely consider the former, which is necessary to facilitating practical and immediate utilization on mobile devices. I also explore various approaches for common training techniques, such as image augmentation and fine-tuning, to maximize accuracy; to briefly elaborate, I was motivated to do so by the intuition that the smaller parametric capacity may lead to less overfitting tendencies, and thus a different set of best practices that diverge from conventional approaches. Lastly, I run the proposed architectures on a mobile device, and quantify and highlight their

runtime benefits. In terms of organization, this focuses primarily on the methodology and rationale, while Chapter 5 provides the corresponding results, evaluations, and discussions.

4.2 Dataset Choice

To evaluate the accuracy of the proposed networks, I train and validate them on the UEC Food-100 (UEC100) [87] and UEC Food-256 (UEC256) [88] datasets; I choose these for their abundance of labelled food images, and their use in similar evaluations across several related works [91, 93, 94, 82, 120] – which I utilize for direct comparisons to better highlight the advantageous of my proposed approach. Other benefits of the datasets include the provisioning of corresponding training-validation splits – which I organize the datasets by, in order to improve the consistency of reported results – and the inclusion of bounding box information, the importance of which is elaborated on in image pre-processing.

4.3 Architecture Choice

Diet-tracking applications are naturally moving toward mobile-based platforms due to the advantages of mobile devices in ubiquity, portability, camera access, and (modest) computational capacity; as such, there is a strong need to explore on-mobile CNN usage to facilitate local food classification. In addition, although computational capacity continues to improve at a rapid rate, mobile devices are inherently limited in this regard due to their portable nature; thus, CNNs need to respect the computational constraints of mobile devices while also remaining highly accurate.

Unfortunately, as CNN accuracy continues to increase, so too does the computational cost – typically via increased depth or width [42, 94] – as it is often required

to facilitate the improved performance. Additionally, numerous approaches predominantly focus on maximizing accuracy by whatever means; as such, it is common to observe the use of impractical techniques like ten-crop testing [94] – which lengthens inference by roughly a factor of 10 – or ensemble methods [31] – which greatly increases parameter usage from requiring multiple CNN instances. Therefore, in addition to pursuing accuracy, I explore the use of parameter-efficient architectures to help bridge the gap between current approaches, and practical implementation; I specifically investigate DenseNet [9] and MobileNetV2 [10], which are explicitly designed for both accuracy and efficiency, and briefly elaborate on these choices in the following sections. In addition, I note that detailed information on the architectures is provided in Chapter 2.

4.3.1 DenseNet

I choose to explore the DenseNet [9] for mobile food recognition as it is one of the most parameter-efficient architectures for which pre-trained weights are available – the importance of which I elaborate further in the training methodology; I specifically begin with the Keras 121-layer DenseNet, which has roughly 7 million parameters excluding the final softmax layer, and a top-1 single-crop error rate of 25.03% on ImageNet [64]. To briefly highlight its key design elements, a significant enabler for its parameter-efficiency is its densely-connected structure which, in addition to improving information flow – as discussed in section 2.4 –, also greatly reduces parameter usage by enabling the use of thinner layers – meaning to have less learnable filters – throughout the network. The reduction of learned feature maps is compensated by the dense connections, which feed-forward all intermediate

feature maps such that the input volume to every subsequent layer is richer and more-expressive; the result is a network that maintains its competitive accuracy despite being more parameter-efficient. Its strong performance is best encapsulated by how it outperforms ResNet [35], with superior accuracy and less parameters, across numerous datasets when trained identically [9].

4.3.2 MobileNetV2

I choose to explore the MobileNetV2 [10] architecture for similar reasons to the DenseNet; it is one of (if not) the most parameter-efficient architectures for which pre-trained weights are available. I specifically choose the MobileNetV2 with width multiplier $\alpha = 1.4$; it features a parameter count of about 4.4 million when excluding the final softmax layer, and a top-1 single-crop error rate of 24.77% on ImageNet [64]. While a lower width multiplier would be more runtime efficient, this particular incarnation is more accurate and closer in parameter-cost to the DenseNet-121; as such, this particular choice also allows for a fairer comparison between the two. It is important to note that the pre-trained MobileNetV2 actually strictly outperforms the DenseNet on ImageNet, and so its use is of particular interest. To briefly discuss its design, its efficiency stems from design choices that specifically target mobile applications; these elements include depthwise separable convolutions, inverted bottleneck residual connections, and linear bottlenecks – as discussed in section 2.5.

4.3.3 Mobile Wide-Slice Branch

While I primarily explore DenseNet and MobileNetV2, I also explore the wide-slice branch [94] for extracting features more relevant to food; the primary motivation

stems from its contribution to one of (if not) the highest reported classification accuracies on the UEC Food datasets [87, 88]. To elaborate, it employs the use of a wide-slice kernel design to explicitly learn layered features that are inherent and indicative of numerous food classes like *burger*; although its wide-slice design is highly atypical in literature, it is demonstrably-successful for food recognition, and so I decide to explore the wide-slice branch further. Additionally, the branch is utilized by affixing it in parallel to a main network – in the case of [94], the wide residual network (WRN) [42] – and so I attempt to similarly replicate and analyze its success. In implementing the branch, I discover that its parameter cost is, unfortunately, prohibitively expensive, and thus I propose an efficient re-design; to illustrate its expense, we first consider the number of parameters in the wide-slice convolutional layer that has 320 kernels of width 224, height 5, and 3 input channels:

$$320 \cdot (224 \cdot 5 \cdot 3) = 1,075,200$$

where when affixed to DenseNet and MobileNetV2, would increase network size by about 15% and 24% respectively. In addition, as the feature vector (FV) output of the wide-slice branch is concatenated with that of the main branch, it also increases the parameter usage of the following fully-connected (FC) layer. To determine this amount, we first consider the size of the FV following the max-pooling layer of width 1, height 5, and stride 3 – which further has an input volume of width 1, height 222, and depth 320. Each feature map outputted by the max-pooling layer has a width of 1, and a height given by:

$$1 + \frac{222 - 5}{3} = 73.33$$

which is either rounded to 73 – by ignoring the leftover pixels, or 74 – by padding for the missing values. As a height of 73 is stated, the total size of the (flattened) FV is given by:

$$320 \cdot 73 = 23,360$$

which is then subsequently concatenated with the FV of the main network, and fed into the following fully-connected layer. The original design uses a 2048-neuron fully-connected (FC) layer following the concatenation operation, and thus has a cost of:

$$23,360 \cdot 2048 = 47,841,280$$

which is several times larger than either the DenseNet or MobileNetV2, and infeasible for mobile deployment. Even if we assume the omission of the 2048-neuron FC layer, and a best-case of feeding directly into a 100-class FC layer, we still get a cost of:

$$23,360 \cdot 100 = 2,336,000$$

which remains a significant increase to either proposed networks. While I consider omitting the FC layer and investigating the branch as is, I do not find the benefits to be sufficiently-significant to warrant the cost; to elaborate, I find the benefits of the wide-slice branch to likely stem from a regularizing effect to the core network. [94] evaluates the wide-slice branch performance and reports a low standalone accuracy for it; however, they also demonstrate that it increases the top-1 accuracy of the WRN by about 3%. The results indicate that while the wide-slice branch may not necessarily learn strong and discriminative representations, they still facilitate an improvement to accuracy when coupled with a core network that does; these

results are more-easily explainable by a regularizing effect to the main network that improves generalizability, and overall accuracy on the validation set. As such, I am motivated to propose a computationally-efficient re-design that is more suited to mobile deployment.

I first halve the number of filters to 160 in the convolutional layer, which in turn halves its parameter cost; while I consider smaller sizes, reduction by a larger amount may degrade performance too significantly – especially given the very-large receptive field of the filters. Next, noting the presence of overlap in pooling to be much-less significant than computational feasibility, I increase the max pooling stride from 3 to 5, and further reduce the height of the pooling layer output to 44. Finally, I use two average pooling operations of height 22 and width 1, to condense the resulting FV of size 44 into an FV of size 2; I thus obtain a final (flattened) FV of size 320. I use global average pooling to condense each feature map into a single value due to its common employment for such tasks – as in the case of DenseNet and MobileNetV2; however, I use two due to concerns that averaging 44 values would dilute the information too significantly, and to avoid shrinking the FV size too drastically such that its representation is too insignificant a component in the final concatenated FV. I evaluate the efficacy of this proposed scheme by affixing it in parallel to both proposed networks to create wide-slice variants, and evaluate these in tandem with their vanilla variants.

4.4 Training

The accuracy of the CNN classifier is highly dependent on the training process; however, the procedure is often complicated as CNNs typically exhibit a strong

tendency to over-fit, commonly learning to predict their inputs perfectly while failing to be as accurate on the validation set. Thus, I explore the use of various training techniques - guided by best-practices as discussed in Chapter 2 – to maximize the accuracy of the proposed networks on the UEC datasets.

4.4.1 Image Pre-Processing

Following the common trend in numerous publications [9, 18, 35], I pre-process the images by removing the mean pixel activity and normalizing their values. To remove the mean – which serves to lessen the potential impact of factors like image brightness – I calculate per-pixel channel means across the training dataset, explicitly excluding the validation set to keep the data fully separated; to normalize – which may hasten convergence, though is not strictly required – I divide by 128 to acquire a range of 2. In addition, I note that I consider normalizing by dividing by 256, but opt to avoid any possible precision issues, however likely, that may arise from limitations in floating point representation.

I further pre-process each image by extracting ground-truth crops using the provided bounding box information, due to the presence of multiple food items in numerous images that may negatively impact training and evaluation. I also serialize the resulting data into *.tfrecords* format to speed up the training process – by not having to pre-process multiple times, and to input the data more efficiently; in serializing, I include both the pre-processed ground-truth crops, and their corresponding labels. I further segment these into five distinct files, corresponding to the provided dataset splits, and use the *val4* split for validation and the remainder for training.

4.4.2 Image Augmentation

Image augmentation has been empirically demonstrated to improve accuracy in virtually all cases, improving network generalization and supplementing training data [18, 35, 94]. I propose and evaluate three different approaches:

1. The image is first randomly (50% of the time) flipped horizontally to acquire reflection invariance, and the smaller side is resized to 256 pixels to facilitate random sampling. The brightness, contrast, and saturation are randomly adjusted with a delta of 0.15, 0.2, and 0.3, respectively, to apply desirable photometric distortions as per [18, 35]. Five central patches are then sampled at five different scales, evenly distributed from 40-88% of the image, and resized to the original image dimensions; this introduces scale and resolution invariance, and is motivated by previous success [31, 121]. Then, a random 224x224 crop is taken for each of the five samples, followed by three random 224x224 crops of the original image, to introduce translation invariance and to capture all edge data in non-square images [18, 94]. The post-flip, pre-distortion image is finally cropped and padded as necessary to 224x224 to actually train on undistorted data, and results in a total of nine training images.
2. The second approach is identical to the first, excluding scale augmentation. Specifically, the image is also randomly flipped, resized to have smaller side 256, and photometrically-distorted. However, instead of sampling at five different scales, eight total 224x224 random crops are taken – to mimic the successful scheme in [94]; as with the prior approach, the original image is resized via crops and pads, and also results in nine total training images.

3. The third approach is based on [122], which successfully trains using only a single augmented image comprising all transformations. Specifically, this approach extends the first by also randomly-flipping, resizing, and photometrically-distorting the original image; five samples are then also produced at multiple scales. However, only a single one of these is chosen at random, and added to the training dataset. This scheme is of particular interest as it manages to train the network on all desired transformations while only increasing dataset size by a factor of 2. Additionally, it is hypothesized that a milder augmentation scheme may be beneficial, as the proposed architectures have significantly-smaller parametric capacity with respect to counterparts [18, 30, 35], and may not struggle as much with overfitting.

To include a baseline result, I also train without augmentation. Finally, I also note that I performed preliminary experiments where *scheme 1* originally sampled eight multi-scale samples, instead of the proposed five-with-three-random-crops; I omit these results due to extremely-poor accuracy. To briefly discuss, since scale sampling was much more aggressive by starting at 20% instead of 40%, I may have introduced too much distortion due to significant noise when scaling-up these small samples to the original image size. The issue may have been further exacerbated by the fact that some original images were of very-small size, from 60 to 80 pixels per side, and are already very-distorted when scaling up. As such, the choice of starting at 40% scale sampling is largely motivated by this result. Furthermore, the result partly-motivates my proposal of *scheme 3* as well, since I felt that overfitting – in

this case – may not be as significant a concern as it typically is; thus, I was interested in exploring and quantifying the intuition further.

4.4.3 Transfer Learning and Fine-Tuning

I considered the availability of pre-trained ImageNet weights to be of paramount importance in the selection of architecture, as the benefits of transfer learning are ubiquitously documented in literature [28, 91, 94]; it is a critical component to not only converging to a good set of weights, but also to do so significantly quicker.

I begin with pre-trained weights using the Keras [64] applications package, and use TensorFlow [69] for implementation as its flexibility facilitates investigation of my data augmentation schemes. I replace the ImageNet fully-connected (FC) softmax classification layer with new 100- and 256-output ones, corresponding to UEC100 and UEC256 respectively, and initialize them using the He normal initializer [54]; I note, however, that initialization is less important here as it is only for a single layer, since the rest of the network is already pre-trained. I then train solely the new layers to complement the pre-trained weights in acquiring good, discriminative features. I report the corresponding training and validation accuracies, with respect to a randomly-initialized network, in Chapter 5 – over a predetermined number of epochs.

Once the networks are trained, I further explore fine-tuning an increasing number of weights by unfreezing their values – starting with the deepest 25% of weights, up to 100% of them; the motivation stems from the observation that shallower layers typically learn more generic features, while deeper ones specialize [29, 57]. I am further motivated by noting that the small parametric capacity of the proposed networks

may indicate less difficulty in overfitting, and that their parameter-efficiency may perhaps be due to more inherently-generalized representations that the corresponding architectures enable; as such, full fine-tuning – which may typically lead to overfitting – remains of interest to explore. I also note that I only explore fine-tuning after training the new classification layer, as gradient updates from a randomly-initialized layer may completely undo the well-learned representations of transfer learning; that said, full fine-tuning can technically be done as soon as the accuracy begins converging well, and without waiting for it to plateau as in this case. I share the resulting accuracy of a variety of fine-tuning prescriptions in Chapter 5.

4.4.4 Optimization

To make my methodology more consistent with related works, I use a similar optimization scheme as [94]. Specifically, weight updates are performed using batch gradient descent with momentum 0.9, initial learning rate 0.1, and learning rate decay 0.0005. Furthermore, the learning rate was step-decayed to 0.002 and 0.0004 at epochs 25 and 45 respectively, with training occurring over either 50 or 60 epochs. The network is trained using a tensor processing unit (TPU) [72] via Google Colaboratory [123], to facilitate less time-consuming exploration of the various proposed training schemes. Regarding batch size, although a setting of 64 is chosen, its resulting size is 512; to explain, TPUs are optimized for larger ones by design, and it further multiplies the chosen size by eight – one for each core – to perform training more efficiently.

4.5 User Habit System

Food items are inherently difficult to classify due to challenging class variation properties as mentioned in section 3.3. To demonstrate, examples from the UEC256 dataset include the presence of numerous classes of *rice* dishes that vary slightly from one another, such as *chicken rice*, *fried rice*, and *pilaf*, while classes like *sashimi* can include numerous different cuts of fish including *salmon*, *tuna*, and *butterfish*, which can increase the difficulty of learning good, discriminative features. The class variation issue is further highlighted in considering that there are an innumerable number of ways to cook a dish, making it almost impossible to comprehensively train a network to detect every possibility. To highlight the extent to which this issue may be problematic, certain items in real-life may look virtually identical, such as *Coca Cola* and *coffee*, and it is impossible to consistently classify such items from visual information alone.

To address the issue, I propose a novel user habit system that utilizes user context – specifically, their dietary history – to improve accuracy, by biasing and personalizing classifier predictions based on the specific user. The method is primarily inspired by two observations: first, that top-5 predictions for most CNNs are often very accurate – DenseNet121 and MobileNetV2 ($\alpha = 1.4$) exceed 92% on ImageNet [64], while [94] illustrates how most CNNs exceed 90% on the UEC datasets; and second, that meal consumption is uniquely habitual in nature, with people often exhibiting preferences to certain meals at different times of day – oft-repeated on a week-to-week basis. To further clarify, there is usually an observable pattern in the

timing, and the contents, of a user's meal consumption. Specifically, regarding timing, most people eat their meals at set times of day that correspond to the popular denotations of breakfast, lunch, or dinner; regarding contents, people often favour certain dishes over another, and may not consume some foods at all. Additionally, there are further habitual considerations in the restaurants a person may like to frequent, the dishes they typically cook at home, and the consumption of previous-day leftovers, for example. Considering these factors as an aggregate, especially over the course of a period of time like a month, it is very likely that there exists an underlying pattern for every individual that is both observable and exploitable, such that it can be leveraged for increasing food recognition accuracy.

4.5.1 System Description

Based primarily on the above, the proposed approach comprises a system – depicted in Figure 4–1 – that first builds a dietary history for the user; it does so by recording the label, mealtime (such as breakfast or lunch), and day-of-week, whenever food recognition is performed. As the system is increasingly utilized, this history becomes progressively rich and detailed such that it more accurately encapsulates the user's habits, and provides sufficiently-meaningful context for personalizing food predictions. The system then utilizes this information by modifying the prediction scores of the classifier such that the contextual information is accounted for, and chooses a final prediction corresponding to the best resulting score. Specifically, context multipliers are computed for each of the top-5 classes following a classification request; each context multiplier is then applied to each corresponding softmax value to obtain the new set of scores, and the system chooses the class corresponding to

the highest score. The following paragraphs will further elaborate on the context multiplier, and discuss several implications of the proposed system.

The context multiplier, c, is applied to softmax scores via multiplication in order to modify their values, and is computed as follows:

$$c = 1 + s(i_1 \cdot f_{day,mealtime} + i_2 \cdot f_{mealtime} + i_3 \cdot f_{others}), i_1 + i_2 + i_3 = 1$$

where 1 is included to account for the original softmax score, s is the strength coefficient which modulates the multiplier's impact, the various i are importance coefficients that sum to 1 and determine how indicative their associated frequencies are to the user's habits, and the various f are frequencies that represent how often a predicted food class appears in the user's diet on the same day and mealtime, the same mealtime, and the other mealtimes, respectively. Generally, the interplay of i and f are used to facilitate profiling different types of habitual users, with values for i chosen such that they are representative of users that may consistently eat *fried rice* on Wednesdays, or consume leftovers for lunch the following day, for examples.

To elaborate, the proof-of-concept experiment features a hypothetical user that follows a repeated weekly pattern in meal consumption; specifically, they consume the same food items repeated every seven days. A full depiction of their diet can be found in Table 4–1. For this profile of user, the frequency of a certain food class previously appearing on the same day and mealtime is an important indicator to their eating habits; as such, it follows that a higher importance coefficient should be assigned to this frequency. Similarly, the hypothetical user also exhibits a regularity to food items consumed for each meal; for example, food items consumed at breakfast (meal

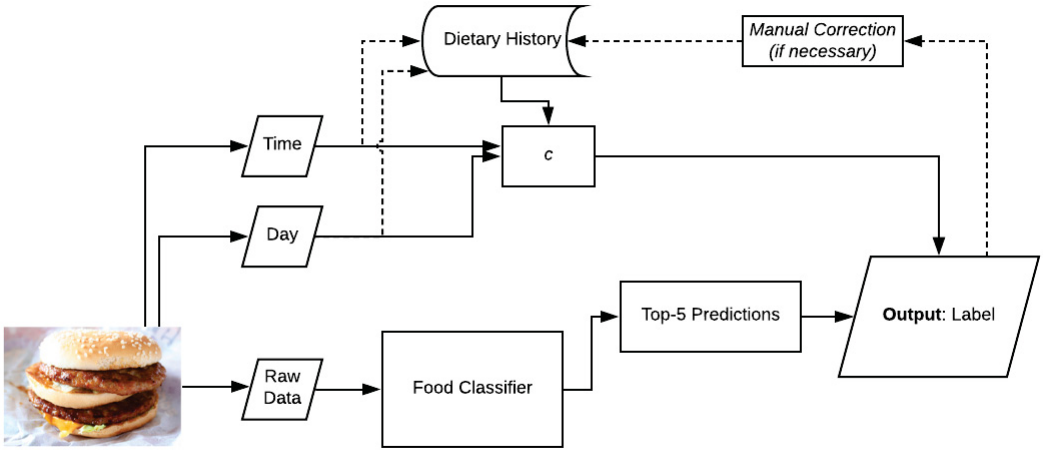


Figure 4–1: Overview of the proposed food recognition system incorporating user habits, where c is the context multiplier. Top-5 predictions are utilized by modifying their softmax scores as per c, and dietary history is progressively built with each food classification request – where the mealtime (time), day-of-week (day), and label are recorded.

1) are not seen being eaten for dinner (meal 2), and vice versa. It follows that the importance coefficient for the same-mealtime frequency should thus be high as well, but not as much as the previous frequency. In addition, the frequency of the food class appearing at different mealtimes should still offer an impact, albeit minimal, as there remains a chance that they may break from routine for a day or two moving forward. This hierarchy of importance is the primary motivation for ensuring they sum to 1, as it is the ratios between them that are important – as opposed to the raw magnitude. Thus, the final derived form is as follows:

$$c = 1 + s(0.8 \cdot f_{day,mealtime} + 0.15 \cdot f_{mealtime} + 0.05 \cdot f_{others})$$

Although the above accurately captures the hypothetical user, it is reasonable to assume that most people would not follow a similar, strict routine, and it is important to be able to finetune the various importance coefficients to fit different profiles of users. While non-ideal, it may be prudent to allow users to adjust these settings in a convenient way; for example, they may pick from pre-defined profiles for which corresponding importance coefficients are pre-determined, or can be asked to select slider values for certain questions like, "how often do you eat the same dish, the same day, on a weekly basis?". Alternatively, the use of an algorithm to retroactively analyze various properties of the dietary history may be effective in determining optimal settings in an automated, programmatic way, that avoids having to rely on manual user entry. These are discussed further in the results and conclusions.

Meal	Day1	Day2	Day3	Day4	Day5	Day6	Day7
1	sushi	miso soup	miso soup	teriyaki grilled fish	teriyaki grilled fish	egg roll	sushi bowl
	udon noodle	sashimi	sashimi	chicken rice	chicken rice	green salad	mixed rice
2	tempura udon	beef noodle	mixed rice	pilaf	fried rice	pilaf	fried rice
	sushi bowl	chicken cutlet	sirloin cutlet	sirloin cutlet	sashimi bowl	beef noodle soup	chicken cutlet

Table 4–1: Sample proof-of-concept diet

Proceeding to the proof-of-concept, the considered week-long diet is as depicted in Table 4–1, as previously mentioned; it features the hypothetical user consuming two meals a day, each consisting of two different food dishes, for a total of 28 different food items. This week-long diet is repeated 4 times to extend it to a month-long sample, and includes a large degree of repetition to properly simulate a habitual user. For pre-processing each image, the bounding box is not considered in the evaluation despite being trained with them, in order to increase the difficulty and challenge in classifying the images; however, the per-pixel channel means are subtracted from each image to be consistent with how the CNNs were trained. To input the images to the CNN, they are resized such that the smaller side is scaled, and the larger centrally-cropped to 224 pixels; in addition to compatibility, it also incorporates a degree of practical consideration as this is a likely implementation method for feeding inputs into the classifier. For image selection, only images in the validation set are chosen to ensure the classifier was not trained on them, and high-similarity groups of classes, such as *rice* and *cutlet* classes, are intentionally selected to increase the classification difficulty via low inter-class variation. In addition, several of the chosen images feature several food classes per single image, and are purposefully chosen to make the recognition task more difficult.

Regarding the classifier, the top-N results of the CNN will be considered, for both N=5 and N=10. Regarding the choice of strength coefficient, s, various values are considered to determine how much biasing is required to improve the top-1 accuracy; while it is clear that larger s would increase the amount of bias, it is prudent to consider the minimum required amount for high accuracy to ensure the classifier

output does not become largely-negligible, and unable to account for deviations in eating habits. The results are reported and discussed in Chapter 5.

4.5.2 Similar Approach

A special section is dedicated to discussing a comparable finding made in parallel to devising this system, and to highlight differences and advantages in my proposed approach. [98] also demonstrates the importance of context for boosting classification accuracy, specifically using factors like time-of-day; it reports results that indicate how context and, to a degree, personalization of the classification results is highly effective for increasing accuracy. It shares similar motivations – such as addressing the challenge of visually-indistinguishable items like *regular* and *diet Coca Cola* – to further emphasize the necessity of context, and even notes the importance of similar factors like food consumption frequencies, as I do in this work.

That said, [98] deviates in its methodology with respect to what I propose. For example, they emphasize meals with multiple food items, and rely on co-occurrence statistics between food items to increase confidence in certain classes; interestingly, as my approach is specific to single-item classification while theirs seems more suited to object detection methods, both can be used in conjunction with each other for further gains. In addition, they relieve the user of any required inputs, which is ideal and may not necessarily be true for my proposition – as discussed in the prior section. However, my system is advantageous in that it is more lightweight, simpler to implement, more flexible, and more modular. Specifically:

- Their system requires an explicit learning model to learn the user preference over time. This imposes larger restrictions on the mobile device.

- Their system is not fully offline on the mobile device, requiring network connectivity. As such, it is also less flexible and portable.
- Their system is tailored to a controlled environment, further reducing flexibility.
- Their system is much more complex and computationally intensive, which may be non-ideal for on-device mobile classification.

4.6 Mobile Benchmarks

As the goal is to facilitate mobile deployment of CNNs, I specifically evaluate the runtime performance on a mobile device; in this regard, the runtime memory and inference time of the classifier is of notable importance, and I thusly evaluate both the DenseNet and MobileNetV2. It is important to note that since they are typically not as accurate as larger architectures, they need to be sufficiently advantageous in these regards. As such, I also similarly evaluate the runtime performance of the Inception-v3, which serves as an apt direct comparison – for reasons I make clear throughout Chapter 5 – to provide a baseline comparison to alternatives.

In the choice of device, there are numerous developments that are significant to the runtime performance of the classifier; namely, of current significance is the availability of the Neural Networks API (NNAPI) [124], and a GPU – with the former being specific to Android devices. Like non-mobile systems, a GPU is critical to computational speed-ups from hardware that facilitates optimization of vector operations that are often performed in CNNs, while the NNAPI implements many optimized operations that drastically speed-up computations. As such, I specifically consider the use of a CPU-only device, as it is ideal for inferring more-generalized conclusions that are relevant to present times; it is likely that most devices today

would not feature the availability of the NNAPI or GPU, as they require newer smartphones as a pre-requisite. I fully detail the experimental setup, including the device details, in Chapter 5, where I also discuss the results of the benchmark.

CHAPTER 5
Experiments, Results and Discussion

5.1 Experimental Setup

This chapter discusses the various experiments and results that quantify the methodology as outlined in Chapter 4. All convolutional neural network (CNN) experiments were run using Google Colaboratory [123] on the tensor processing unit (TPU) backend to have access to the hardware for dramatic speed-ups in iteration time. All code pertaining to the CNNs were written in Python 3.6.2 using both graph-based and eager-execution TensorFlow [69], version 1.13.1, and TensorFlow-wrapped Keras [64], version 2.2.4-tf, respectively. Numpy [125], version 1.16.2, was also used to facilitate the image pre-processing and augmentation. For the mobile evaluation, experiments were conducted on a Samsung Galaxy S7 device, model SM-G930W8, running on Android Version 7.0 and at API level 24; in addition, it is explicitly noted that this device did not have access to the Neural Networks API (NNAPI) [124], nor a GPU, for computational speed-ups.

5.2 Transfer Learning

To empirically, and briefly, verify the positive affects of transfer learning, the DenseNet was randomly initialized using He initialization [54] and trained for the same number of epochs as a pre-trained network using ImageNet weights [64] on the UEC Food-100 (UEC100) dataset. For both approaches, *scheme 1* was used for

image augmentation (see section 4.4.2), and the optimizer settings were identical for both (see section 4.4.4). The results are shown in Figure 5–1.

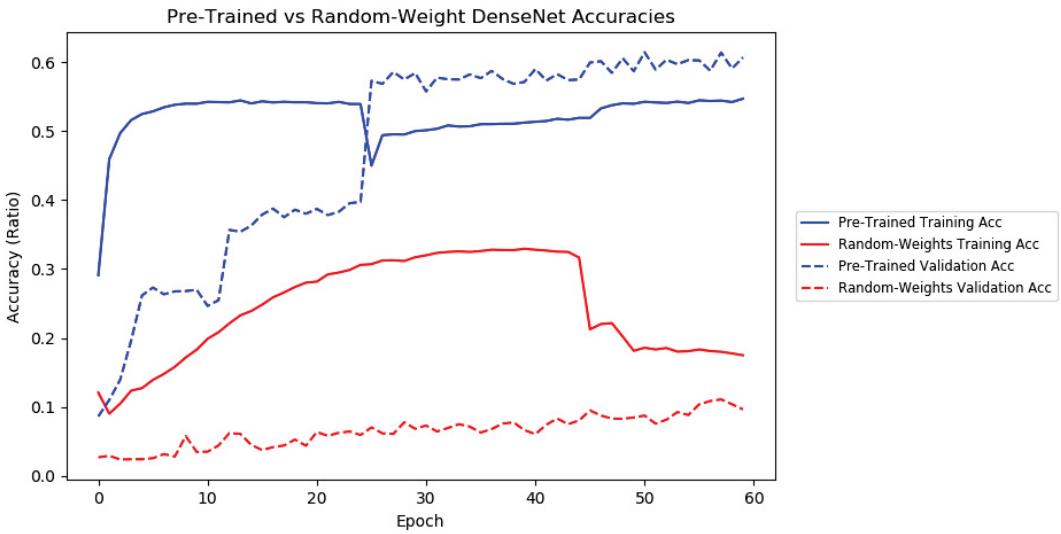


Figure 5–1: Accuracies for pre-trained and random-weight DenseNet on UEC100, using augmentation *scheme 1*

As can be easily observed, the accuracy converges to a much better value with transfer learning than with random initialization of weights, in the same amount of time. The observation is consistent with the well-documented benefits of transfer learning, and is not a surprising outcome. The rate-of-increase for random initialization is significantly slower than in the case of the pre-trained weights, and may be exacerbated by *scheme 1* – which will be later shown to be non-ideal for either proposed network. The training accuracy curves are a bit strange in shape as well, in particular for transfer learning which has a rapid plateau that dips a bit when the learning rate drops; the augmentation scheme may again be a factor, as a similar shape is seen in the other results to follow.

Summary of training approach	
Transfer learning	*ImageNet weights*
Image augmentation	?
Fine-tuning	?

Table 5–1: Summary of final training approach up to this point, given transfer learning results

5.3 Image Augmentation

To evaluate the image augmentation schemes as discussed in Section 4.4.2, each scheme was used to train the FC softmax layer of the pre-trained DenseNet and MobileNetV2. The wide-slice versions of each of these were omitted due to time concerns with the training process, and so just two networks were evaluated for each scheme. The networks were trained on both UEC datasets, and the results are depicted in figs. 5–2 to 5–5. Additionally, it should be noted that several experiments are not depicted that investigated these schemes in tandem with the various fine-tuning approaches proposed in the following section; these undepicted experiments are consistent with, and thus also somewhat encourage, the discussion to follow.

The most striking observation is that both networks trained well with *no image augmentation*, facilitating second-best and best validation accuracies on UEC100 and UEC256, respectively. The high training accuracy indicates that the network did indeed begin overfitting on the training dataset, as expected, but still achieved top-two validation accuracies despite this. In terms of image augmentation, *scheme 3* performed the best by achieving the highest validation accuracy on the UEC100 dataset, albeit by a small margin; however, it allowed the networks to generalize

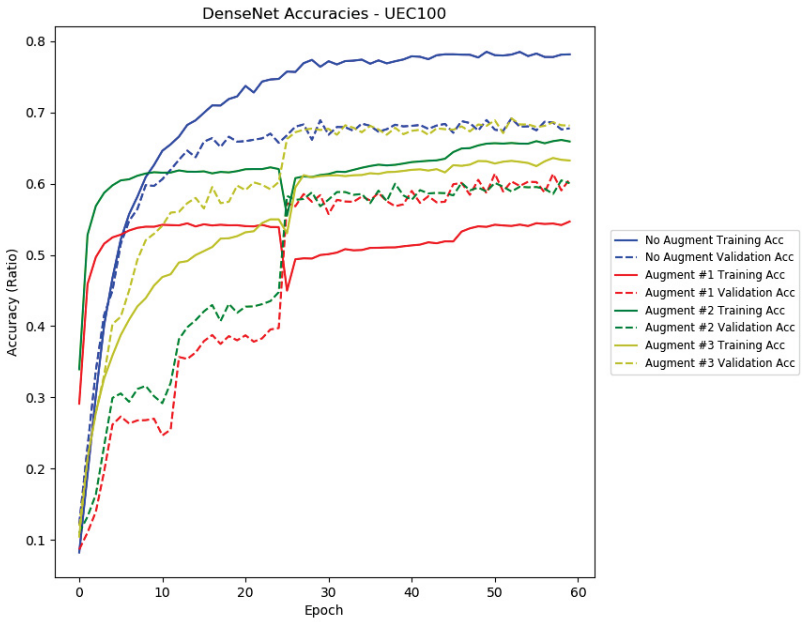


Figure 5–2: Training and validation accuracies for DenseNet121 on UEC100 for each image augmentation scheme.

better, and can be seen by how the training accuracies more closely followed the validation accuracy curves.

Surprisingly, the networks performed much more poorly with more image augmentation, and can be seen by how *scheme 1* and *scheme 2* – both with a total of 8 augmentations – achieved the lowest validation accuracies by about 5 to 10 points. In addition, *scheme 2* exhibited a large amount of overfitting with its high training accuracy across all configurations, though *scheme 1* generalized better with its lower training accuracy and even dropped below its validation accuracy on UEC100.

There are several ways to interpret these results; for one, the architectures may have less overfitting tendencies to begin with, as evidenced by their competitive

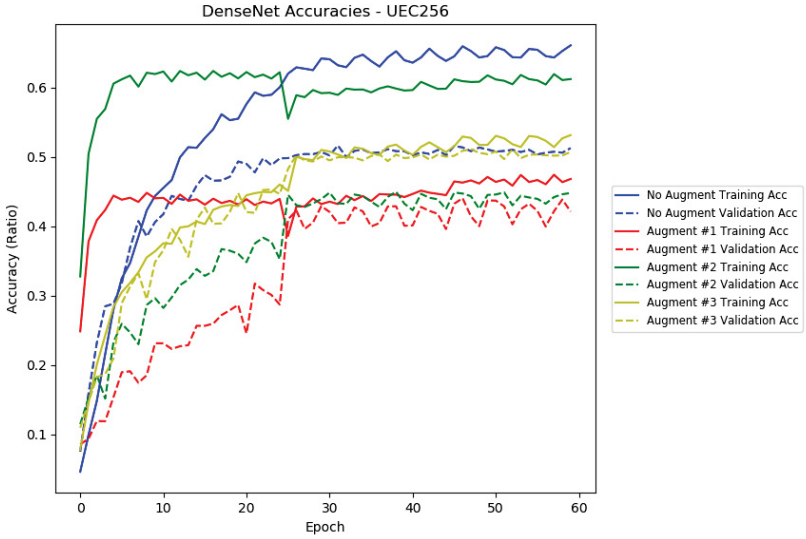


Figure 5–3: Training and validation accuracies for DenseNet121 on UEC256 for each image augmentation scheme.

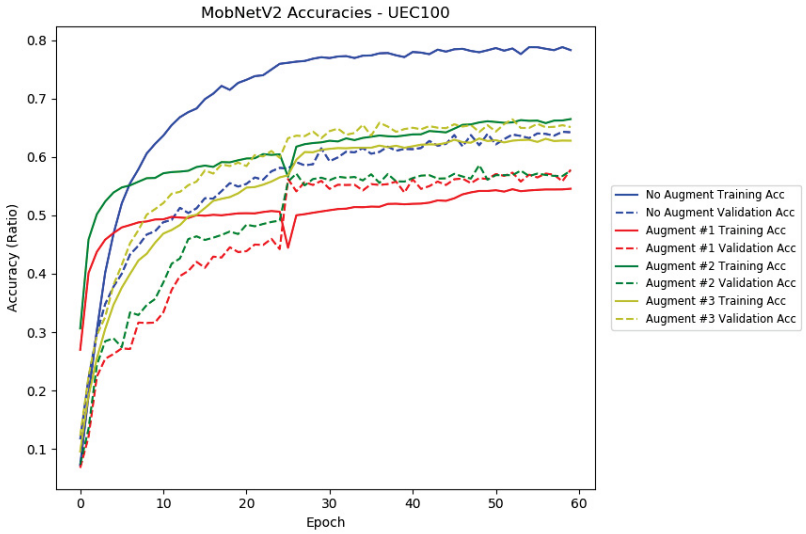


Figure 5–4: Training and validation accuracies for MobileNetV2 on UEC100 for each image augmentation scheme.

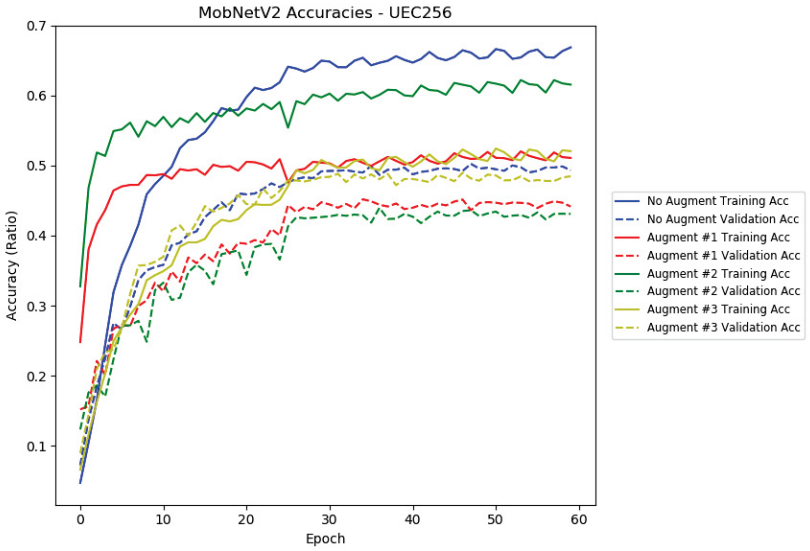


Figure 5–5: Training and validation accuracies for MobileNetV2 on UEC256 for each image augmentation scheme.

accuracy on ImageNet despite having less parameters. In fact, the findings here were later found to be consistent with those in [44], where less data augmentation was used to successfully train MobileNet; in addition, the authors claim that this was due to smaller models having less trouble, in general, with overfitting. Interestingly, they corroborate their observations further by noting how they found distortions from small crops to be troublesome as well, similar to the preliminary iterations of *scheme 1* (see section 4.4.2). It is also important to consider how the use of pre-trained weights may have set the network at a highly generalized baseline to begin with, meaning less need to combat overfitting to start with; this may also aid in explaining the degraded performance of the network when employing the heavier augmentation

schemes, as further generalizing the already-generalized weights may not allow the smaller network to fit on the target task sufficiently-well.

A second consideration should be made in how the training procedure was not fully explored. For instance, training may have benefitted from different augmentation schemes, where perhaps a single image encompassed a single augmentation, such as solely positional-translation or brightness-shift. In addition, other optimizers such as RMSProp [59] may have been more effective at increasing convergence, as it was in [10, 44] for training MobileNet and MobileNetV2; thus, the exploration of various hyperparameter settings for each of these optimizers is important as well. Finally, it is possible that different combinations of optimizer-augmentation pairings may yield greater gains, though it would be fairly time-consuming to fully explore. It is prudent to consider all these factors, but it is left to future work due to time constraints.

Nonetheless, the proposed networks seem to be capable of being trained effectively with minimal augmentation following the use of pre-trained weights, which is highly desirable due to the implications in reduced training time and faster experimental iteration; there is reduced burden on expensive hardware for training purposes, and more time is granted toward exploring optimizations for the target task.

Summary of training approach	
Transfer learning	*ImageNet weights*
Image augmentation	*Scheme 3*
Fine-tuning	?

Table 5–2: Summary of final training approach up to this point, given image augmentation results

5.4 Fine-Tuning

To further bolster the benefits of transfer learning, some amount of fine-tuning is usually required on the target task. Conventionally, the shallowest layers are usually kept frozen as the learned representations are typically more generic in nature [29], and this greatly assists in preventing the network from overfitting and experiencing accuracy degradation. As a result, these weights are almost always left untouched, while the deeper layers are sometimes fine-tuned to better capture the features in the datasets of interest. However, due to the possible implications of the smaller parameter usage in the chosen networks, they may atypically benefit from fully unfreezing all layers and allowing the weight updates to propagate throughout the network.

To evaluate these different fine-tuning prescriptions, the DenseNet model was fine-tuned on both UEC100 and UEC256; however, it should be noted that the fully-connected softmax layer was already partially trained to avoid gradients from its random initialization from destroying good pre-trained weights. The results can be found in Figure 5–6 and Figure 5–7 for both UEC100 and UEC256 respectively. It should be noted that although MobileNetV2 was not fully evaluated due to time concerns, several unmentioned experiments were conducted on them to verify the consistency in observed behaviour across all networks.

From the results, there is a clear trend of increasing validation accuracy as more weights are unfrozen in the network, up to 100% of them, on both datasets. The observation here is consistent with prior discussions and observations, where the smaller parametric capacity may explain the observed tendency to not struggle as much with overfitting, and has implications in the choice of training techniques that

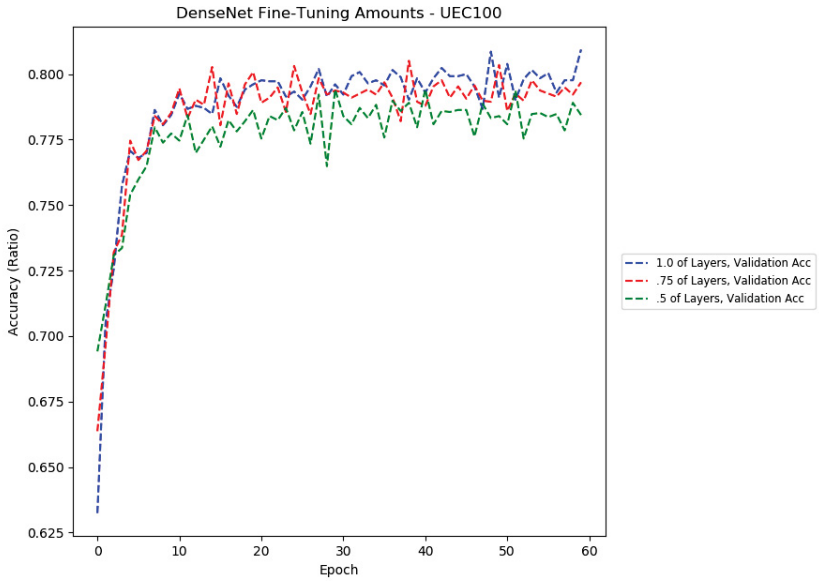


Figure 5–6: Ratio of layers fine-tuned for DenseNet on UEC100 with no augmentation

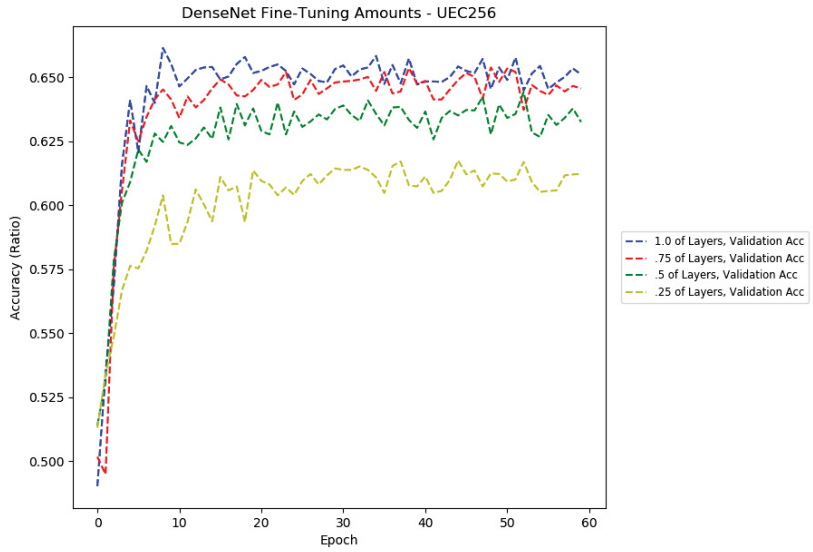


Figure 5–7: Ratio of layers fine-tuned for DenseNet on UEC256 with no augmentation

would typically be problematic for larger networks. The additional fact that transfer learning initializes the weights to good, more-generalized representations may also play a factor in why full fine-tuning is beneficial for DenseNet.

Summary of training approach	
Transfer learning	*ImageNet weights*
Image augmentation	*Scheme 3*
Fine-tuning	*All weights*

Table 5–3: Summary of final training approach up to this point, given fine-tuning results

5.5 Final Results

The networks were trained to acquire their accuracies for comparisons to other approaches; as such, the prescription used was based on the best results from the preceding sections. All models were trained using transfer learning, starting from the ImageNet pre-trained weights; the final FC-softmax layer, and wide-slice branch convolutional layer, had weights initialized using He initialization [54]. The networks were first trained using image augmentation *scheme 3* with the pre-trained weights frozen – to train for non-random values in the initialized parameters –, and the optimizer as defined in section 4.4.4. Following this, the networks were fully fine-tuned, where all the weights were unfrozen, using the same procedure. It should be noted that a more efficient training scheme could technically be used, but due to the process of acquiring results for the various section, some redundancies in the training process resulted; as such, there were intermediate saved weights that allowed for training to proceed more quickly, which was a large concern due to time constraints. To elaborate, training with the weights frozen could have been done in

less epochs, and unfreezing the weights could have been incorporated into a single training procedure instead of starting-and-stopping twice. Regardless, the single-crop accuracy plots are reported for each network on both UEC100 and UEC256 in Figure 5–8 and Figure 5–9 respectively, with final values reported in Table 5–4.

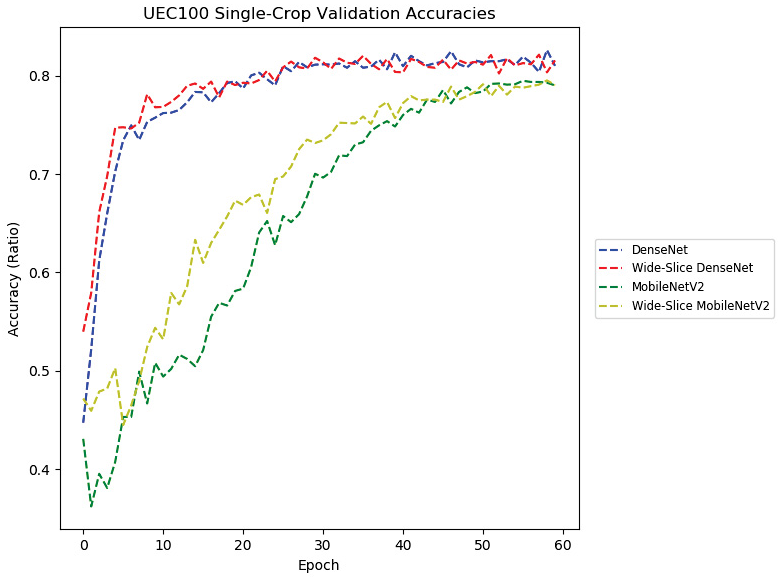


Figure 5–8: UEC100 single-crop validation accuracies for proposed networks

Regarding accuracy curves, the DenseNets consistently converged faster than both MobileNetV2s, with the wide-slice variant converging notably sooner in the case of the latter on the UEC100 dataset; other than this, there is not much difference between the vanilla and wide-slice flavors of the networks. The DenseNets appear to have converged quickly on UEC100, while the MobileNetV2s seem to require additional epochs as the curves only just begin to plateau, indicating that there may be more performance gains given longer training. Additionally, the learning

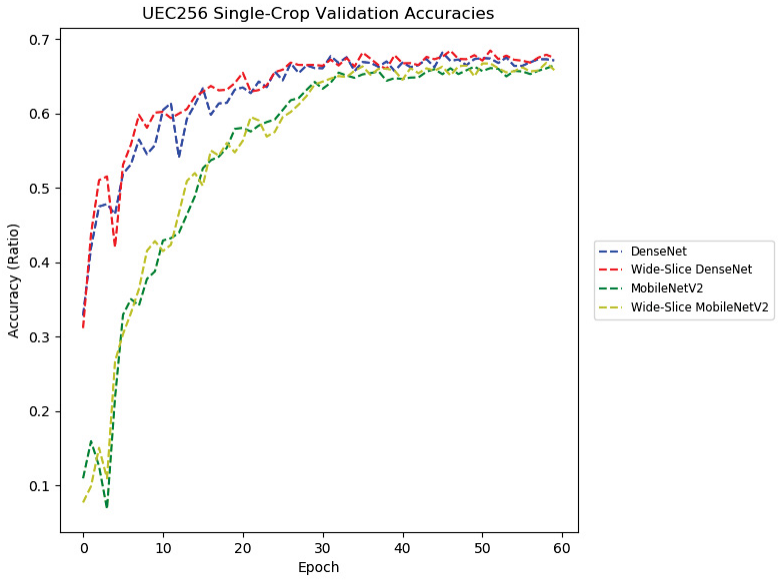


Figure 5–9: UEC256 single-crop validation accuracies for proposed networks

curve was much smoother for the DenseNets, especially in the first 10 epochs, while the MobileNetV2s possess more sporadic curves that are not as monotonic. On UEC256, the DenseNets once again converged fairly quickly, but interestingly, so do the MobileNetV2s – as indicated by the larger plateau. Additionally, although the DenseNet curves are a bit more sporadic initially, the validation curves are generally quite smooth. In general, it seems that the MobileNetV2s may have had a harder time learning than the DenseNets; a possible implication is that the smaller parametric capacity mandated learning more specific representations than the ImageNet weights provided, and with greater specificity than DenseNet required, which would further imply the need for MobileNetV2 to have more-strongly fine-tuned weights.

Dataset	DenseNet	MobileNetV2	Wide-slice DenseNet	Wide-Slice MobileNet
UEC100	82.6%	79.5%	82.1%	79.5%
UEC256	68.2%	66.5%	68.5%	66.8%

Table 5–4: Final top-1 single-crop accuracies for the proposed networks

Regarding final accuracies, the DenseNet, wide-slice DenseNet, MobileNetV2, and wide-slice MobileNetV2, achieved 82.6%, 82.1%, 79.5%, and 79.5% on UEC100 respectively, and 68.2%, 68.5%, 66.5%, and 66.8% on UEC256 respectively. In general, the DenseNets outperformed the MobileNetV2s, although the accuracies are only 2-3% less; this result can be explained by the higher parametric capacity of the DenseNets, where larger networks typically converge to better accuracies [94]. On UEC100 however, the MobileNetV2s only just began converging as described above, and the discrepancy may be reduced given longer training time. The models performed notably worse on UEC256, being 12% lower in final accuracy; this may be explained by a need for increased parameters to capture a more complex underlying relationship from the introduction of challenging classes to differentiate between, such as *pork loin cutlet* and *pork fillet cutlet*, and the presence of more classes in general that may further complicate the underlying relationship.

Regarding the wide-slice branch, it is important to reiterate that it is thought to improve classification accuracy of food items due to its purposeful design in capturing wide-layered features that are indicative of food classes like *burger*. Despite the strong results in [94], it did not seem to yield much benefit here; on UEC100, it actually reduced DenseNet accuracy by 0.5% and had no effect on MobileNetV2, while on

UEC256, it only increased accuracy by 0.3% on both architectures. As a result, there are several implications that are important to consider.

First, the efficient re-design proposed in this thesis – which is necessary as the original is too computationally-demanding – may have rendered the branch ineffective due to reducing the convolutional layer width and output feature vector size too drastically; thus, either it is necessary to craft the re-design more carefully, or simply consider it an option too demanding for mobile deployment. Second, it should be reiterated that although [94] reported strong results, the accuracy for the standalone branch was very low; in addition, it is well-established that most large CNNs exhibit overfitting concerns, and the efficacy of various generalization methods including batch normalization, dropout, and image augmentation, help further cement this claim. With these observations, and the reasonable suggestion that the success of the wide-slice branch may come primarily from a regularizing effect as opposed to the strength of its extracted features (as per section 4.3.3), it is likely that the benefits were marginal due further to the findings that smaller networks in general do not struggle as much with overfitting; to reiterate, the claim is corroborated in [44]. As for the performance increases on UEC256, it may have come simply from a general increase in the number of parameters available to the networks; the larger number of classes may require a more complicated representation to be learned, which the increased parametric capacity may have accommodated to show a beneficial increase. The results in [94] help support the observation; for example, the reported accuracy of Inception-v3 on UEC100 is very similar to the results in this thesis, but the disparity on UEC256 is notably larger. These observations may indicate that it is likely the

size of the network should be chosen based on the modelling difficulty of the target task, and it has important general implications for the mobile deployment of CNNs:

1. There is a need for even more parameter-efficient architectures that are capable of learning stronger, more discriminative features.
2. There is a need for greater computational capacity on mobile devices. Larger memory capacity would enable the use of larger CNNs to model more complex tasks, and although processing was not discussed, it would enable faster inference.
3. There is a need for schemes that can leverage the superior top-5 accuracies of CNNs.

5.6 Efficiency and Performance Comparisons

While larger networks may perform better on food recognition [94], their significantly larger memory cost makes them non-ideal for mobile deployment, which has limited computational capacity. Thus, a large consideration in the use of DenseNets and MobileNetV2s comes from the advantages they offer in parameter reduction, and increased parameter-efficiency; that said, the classifier still needs to be accurate, and so these not coming at a large accuracy cost is important to consider as well. To facilitate discussion of these factors, the parameter count of similar approaches are obtained, and listed alongside the published accuracy and architecture of the respective works in Table 5–5. The following paragraphs will first detail the procedure employed in acquiring these values – as they were not always provided –, discuss how they are used to quantify the parameter-efficiency of the various networks, and discuss the final results altogether. A note should be made that the wide-slice variants

are not evaluated in this section, primarily due to the negligible impact they had and to streamline the comparisons to follow.

Network	Top-1 (UEC100)	Params (UEC100)	Top-1 (UEC256)	Params (UEC256)
AlexNet [120]	57.9%	58.7M	N/A	N/A
GoogLeNet [91]	77.2%	6.1M	63.8%	6.2M
AlexNet+OVA [90]	78.8%	58.7M+	67.6%	59.3M+
Inception-v3 [92]	81.5%	22.0M	76.2%	22.3M
ResNet-200 [94]	86.3%	67.1M	79.1%	67.4M
WRN [94]	86.7%	40.0M	79.8%	40.1M
WISeR [94]	89.6%	90.3M	83.2%	90.7M
DenseNet-121 (proposed)	82.6%	7.1M	68.2%	7.3M
MobileNetV2 (proposed)	79.5%	4.5M	66.5%	4.8M

Table 5–5: Comparison of various CNN-based results with respect to the proposed approaches

To calculate the parameter cost of a network, the most common process involves computing parameters primarily for three layer types: convolutional (conv), batch normalization (BN), and fully-connected (FC). BN is typically ignored in estimates as their cost is relatively negligible, and optimization techniques such as "folding" can be used to incorporate them into preceding or following conv layers such that they have no associated cost at inference time. The cost of a conv layer is given by:

$$(k^2 \cdot d_i + 1) \cdot N$$

where k is the conv filter size assumed to be symmetric, d_i is the depth of the input feature maps, N is the number of filters, and 1 is for the bias term; however, it should be noted that bias can usually be omitted for most modern networks due to the use of BN layers. In addition, the parameters in an FC layer is given by:

$$n_i \cdot (n_o + 1)$$

where n_i is the size of the input feature vector, n_o is the number of neurons corresponding to the number of output classes, and 1 is for the bias term in each neuron. Although most modern networks have dropped typical FC layers in favor of global average pooling, the final softmax layer is typically comprised of an FC layer, with a number of neurons equal to the number of output classes; thus, in this case, n_o would usually be equal to 100 and 256 for UEC100 and UEC256 respectively for this final layer.

For convenience and time-saving purposes, the parameter values are obtained from the following sources, in chronological order, when available: the original publication, Keras [64] pre-trained models, and manual calculations as per aforementioned equations. Furthermore, the use of bias is assumed for all conv layers unless BN is used, and for all FC layers in general.

There are several factors to note when obtaining these parameter values. First, parameter costs for less-common layer types such as inception [31] or depthwise-separable conv [65] are not explicitly included in this text, but were inferred in a similar fashion. Second, although the parameters for GoogLeNet [31] are listed in the original publication, there were discrepancies – albeit slight – when verifying the

reported values via manual calculations; as such, the ones reported here are based on manual calculations. Additionally, the initial 7x7 conv layer is assumed to be factorized into 7x1 and 1x7 as their provided value implies as such – with a large discrepancy in computed value otherwise. Third, for WISeR [94], it is assumed that the wide-slice branch produces a flattened feature vector of size $320 \cdot 73 = 23360$ prior to concatenation, as nothing specific is mentioned in terms of handling it. It is also assumed that the last 2048-FC layer in the depicted architecture actually corresponds to the softmax output layer as it is otherwise not explicitly mentioned or depicted, and such an assumption would produce a more optimistic estimate that is not as excessive in terms of number of parameters.

While the raw number of parameters provides a good overview comparison of the various architectures, it is beneficial to directly quantify and visualize the parameter-efficiency to facilitate further discussion on this point; thus, the accuracy-per-parameter is computed for each network. Since it provides a measure of each parameters' contribution to the accuracy, it is important when considering parameter-efficiency. This value is computed as follows:

$$p = \frac{acc}{n_{params}}$$

where p is the accuracy per parameter of the network, n_{params} is its number of parameters, and acc is its accuracy. However, this raw value is somewhat meaningless in that the relative performance of each network is what is truly of interest. As such, $p_{MobNetV2}$ is arbitrarily chosen as a reference point, and all other networks are

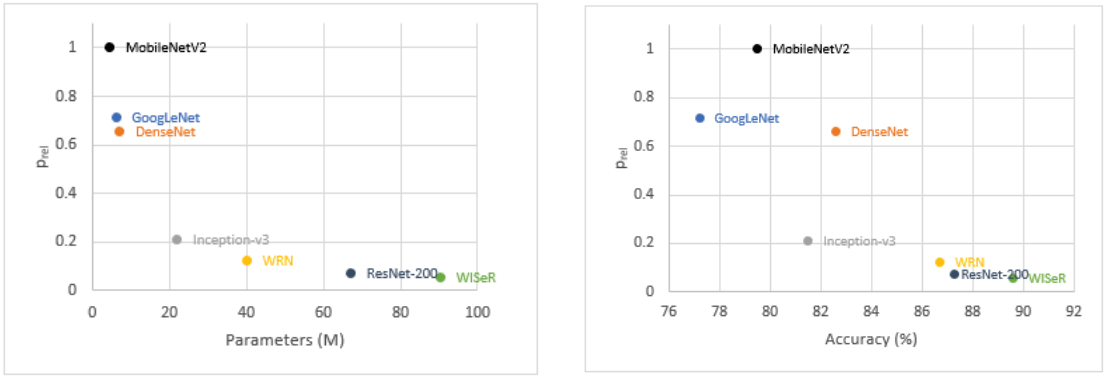
compared against it to acquire a relative performance metric; in other words:

$$p_{relative} = \frac{p_{network}}{p_{MobNetV2}}$$

where $p_{relative}$ is the parameter-efficiency of the network with respect to MobileNetV2, $p_{network}$ is its per-parameter accuracy, and $p_{MobNetV2}$ is the per-parameter accuracy of MobileNetV2. In addition, another benefit is that the metric is a lot cleaner in this form, as the non-relative version would be comprised of numerous, extremely-small values from dividing by the number of parameters, which is several orders of magnitude larger than the accuracy.

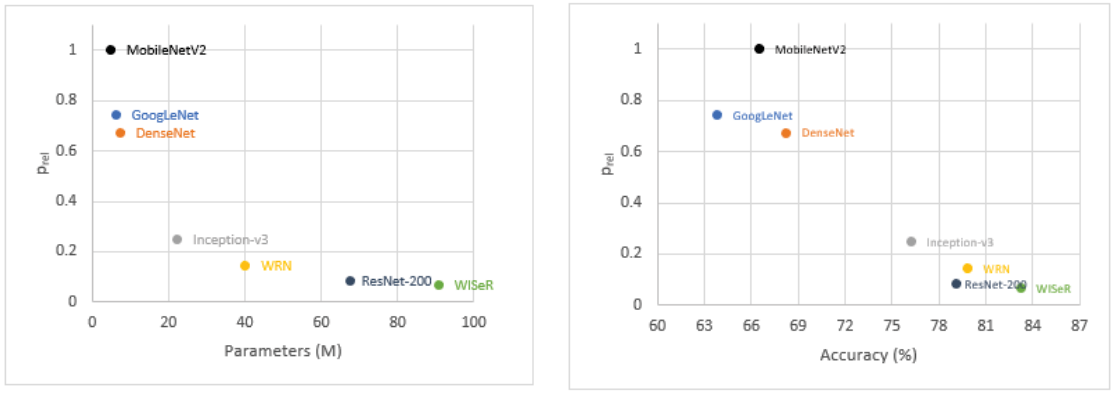
To provide a more comprehensive visualization, $p_{relative}$ is plotted against the parameter count and accuracy for both datasets in Figure 5–10 and Figure 5–11. The graphs facilitate an understanding of how the parameter-efficiency measures up to these factors, and more clearly highlights the trade-offs between the two. For mobile deployment, the chosen network should ideally have high parameter-efficiency, low parameter count, and high accuracy; when discussing which network is better, it is primarily with this context in mind. Finally, it should be noted that both of the AlexNet approaches are omitted as there are readily-apparent strictly-better options in both parameter-usage and accuracy, such as Inception-v3 and DenseNet.

From the results, it is evident that the proposed architectures are indeed more parameter-efficient than the alternatives; however, this excludes GoogLeNet, which is slightly more efficient than DenseNet. With that said, on the two datasets, MobileNetV2 is strictly-better than GoogLeNet; it is consistently more efficient, has less parameters, and has higher accuracy. In addition, although GoogLeNet possesses less



(a) (b)

Figure 5–10: Relative performance-efficiency with respect to (a) parameters and (b) accuracy on UEC100



(a) (b)

Figure 5–11: Relative performance-efficiency with respect to (a) parameters and (b) accuracy on UEC256

parameters and is slightly more efficient, it is about 5% less accuracy than DenseNet across both datasets – which is non-negligible. Additionally, in a more general vein, there seems to exist a trend where the less a network's parameters, the more efficient it tends to be.

On UEC100, the proposed architectures perform very well; DenseNet is 1.1% more accurate than Inception-v3 despite having 3-times less parameters, while MobileNetV2 only lags by 2.0% despite having almost 5-times less parameters. In addition, although the various ResNet-based architectures from [94] outperform the proposed architectures by a maximum of 10.1%, they also have up to about 20-times more parameters; in addition, WISeR explicitly utilizes ten-crop testing [18] to improve its reported results, and it is likely that both the ResNet-200 and WRN do as well – although it is not explicitly clarified, these are likely contributions by the same authors as there are no acknowledgments to their results elsewhere in literature (only the original publications), and so it is likely that they used the same experimental set-up for these evaluations. Keeping this in mind, there would likely be less disparity in single-crop accuracies between them and what is proposed, and helps further cement the claim that both MobileNetV2 and DenseNet perform well on this smaller dataset despite their relative limitation in parameters.

On UEC256, however, the proposed architectures are noticeably less accurate; for example, in this case, Inception-v3 actually achieves 8.0% and 9.7% higher accuracy than DenseNet and MobileNetV2 respectively. In addition, there is a peak accuracy gap of 16.7% with respect to the top-performer, 6.6% larger than on UEC100. Interestingly, this may indicate potential limitations in representational power in these smaller networks, likely coming from the lack of parameters leading to reduced representational capacity; although it seems sufficient for UEC100, it may not be for UEC256 – which may have a more complex underlying representation from both the increase in number of classes, and the properties of said classes. However, it must

be noted that training optimization was not fully explored, and it may be prudent to do so to improve results on UEC256; methods to consider may include alternative image augmentation schemes or training optimizer like RMSProp [59], which was successfully used in training MobileNetV2 on ImageNet [10].

5.7 User Habit Integration

To evaluate the efficacy of the proposed user habit system, the vanilla DenseNet and MobileNetV2 are considered as classifiers; once again, the wide-slice variants are not investigated due to the marginal differences they provide. To facilitate discussions, the top-1, top-5, and top-10 accuracies are first reported in Table 5–6 for both networks; since the proposed system relies on the top-N prediction scores to bolster their accuracies, it is important to quantify and list these target values.

Model	Top-1	Top-5	Top-10
DenseNet	64.286%	78.571%	82.143%
MobileNetV2	71.429%	82.143%	85.714%

Table 5–6: Single-crop accuracies on sample diet

As expected, the top-5 accuracy is much higher than the top-1, with 14% and 11% improvements for the DenseNet and MobileNetV2 respectively. The accuracy only increases by about 3.6% for top-10, which accounts for a single additional correct guess due to the low sample size of 28 images; as such, this makes its performance gain marginal in comparison. Thus, to also maximize efficiency, only the top-5 predictions are utilized to bolster performance.

Interestingly, the MobileNetV2 performs better than the DenseNet in classifying the chosen samples in the sample diet; however, the top-5 and top-10 accuracy

discrepancy can be explained by a mis-classification of a single image – which the MobileNetV2 classified properly on its 9th-best guess – and the top-1 discrepancy explainable by the presence of several multi-class images where its weights favor the target class more than the others. As a result, the top-5 difference between the two CNNs is accounted for by the single aforementioned mis-classified example.

Incorporating the user-habit system, the resulting top-1 accuracy is reported in Table 5–7. For both CNNs, the accuracy eventually reaches the ceiling of the top-5 accuracy, resulting in an increase in top-1 accuracy of about 14% and 11% for DenseNet and MobileNetV2 respectively. For the former, a strength coefficient value of 0.7 is required, while the latter required 0.4. However, the accuracy was only 3.6% off, or a single example, with coefficients of 0.2 and 0.1 respectively, showing a significant improvement even if the scheme is only slightly used. Interestingly, the MobileNetV2 did not require as much biasing as the DenseNet, indicating that it may be relatively better at discriminating between the represented similar-looking classes.

Model	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
DenseNet	67.86	75.00	75.00	75.00	75.00	75.00	78.57	78.57	78.57	78.57
MobV2	78.57	78.57	78.57	82.14	82.14	82.14	82.14	82.14	82.14	82.14

Table 5–7: Top-1 accuracies (%) given differing strength coefficients

The proof-of-concept overall demonstrates the importance of context for boosting food recognition accuracy, as well as its efficacy in doing so. In addition, the method is completely modular, and can be independently applied to the classification results of not just CNN-based approaches, but in general to any approach that

produces a vector of predictions and their corresponding scores. In addition, it is important to highlight that the top-1 accuracy rises to a maximum of 82.143% despite several difficulties that were preserved in this evaluation. To reiterate, these constitute the use of images that are not ground-truth cropped, and that may contain several different food items in a single image. This adds a measure of encouragement to these results, and indicates that the proposed scheme remains effective despite difficulties that may arise from a user capturing difficult and highly-cluttered images that may conventionally reduce the classifier's accuracy by a noticeable degree.

However, to reiterate some discussions in section 4.5, there are numerous parameters that were tuned according to the sample diet; as such, it is difficult to claim that these results will be generalizable to an unknown or generic user diet. Thus, benefits may be found in allowing users to tune the various parameters according to their lifestyle. An easy, user-friendly method of doing so may involve the use of simple sliders, where a user is asked to rank how indicative a certain factor is to their diet in a questionnaire format. For example, if the user consistently sticks to a weekly schedule similar to this proof-of-concept, they may slide to a selection of *9* or *10* out of *10* when asked, "how often do you consume the same food on a weekly basis?", where higher values indicate higher frequencies. These answers can be used to adjust the importance coefficients to the context multiplier, aiming to maintain a ratio of 1 between these to preserve the hierarchical relationship between the various food frequencies. In addition, a user could be asked to indicate how indicative such habitual contexts are to their diet, and the strength coefficient can be adjusted correspondingly. It is also prudent to consider automated alternatives to

reduce user input, in an effort to eliminate potential avenues for user error; such an alternative may involve an algorithm that occasionally examines a users dietary history retroactively, to profile the user based on their eating habits and utilize a good set of hyperparameters for their particular profile. These settings could be manually pre-determined such that they are simply applied as-is for the user, or adaptive and tuned such that they match certain criteria based on the algorithm employed.

5.8 Mobile Performance

While the proposed networks have been discussed extensively in regards to efficiency and benefits to mobile deployment, it remains useful to quantify these claims by performing actual evaluation on a mobile platform, specifically for runtime memory and inference time. A Samsung Galaxy S7 SM-G930W8 was chosen as the target device, as discussed in section 4.6. The code at [126] was then used as a base, and modified to support inference using the proposed models; these were included in the application after converting them into TensorFlow Lite FlatBuffer *.tflite* files using the TensorFlow Lite converter [127], and the modified application was finally built using Android Studio. The runtime memory was captured using the Android Profiler native to Android Studio, while the inference time was obtained using the modified application. In addition to the proposed networks, the Inception-v3 [33] CNN was also evaluated to provide a baseline comparison; it was chosen largely due to the closeness in performance to the proposed networks, as discussed in section 5.6. These results can be found in Table 5–8.

To reiterate, it is important to note that this evaluation does not consider the use of the Neural Networks API (NNAPI) for Android, nor the use of GPUs for

Architecture	Runtime Memory (MB)	Inference Time (ms)
MobileNetV2	107	225
DenseNet121	121	720
Inception-v3	200	1080

Table 5–8: Benchmark measurements for the various CNNs on the given mobile device

computational speed-ups; this choice is motivated primarily by the observation that users may not have access to newer phones with such improvements, making it prudent to perform evaluations using solely the CPU of the chosen mobile device. With that said, the MobileNetV2 and DenseNet121 have runtime memories of 107MB and 121MB, and inference times of 225ms and 720ms, respectively. As expected, the MobileNetV2 runs more efficiently than DenseNet121; its runtime memory is about 20MB lower, but its inference time is about 500ms faster – a significant difference. While the difference in runtime memory is relatively small, the large gap in inference indicates that the MobileNetV2 architecture may be much better suited for inference when considering solely a CPU, and not parallel-operation optimizations which NNAPI or GPUs may introduce. Although the DenseNet does not measure up to the MobileNetV2, it is still better than Inception-v3 by about 80MB in runtime memory, and 300ms in inference time. Coupling these findings with the fact that DenseNet is more accurate than MobileNetV2, there is a trade-off to consider primarily between inference times and accuracy, due to the smaller differences in runtime memory.

However, there are several caveats to these findings that are important to emphasize. First, it should be noted that the accuracy of the two proposed networks

suffer on the more-difficult UEC256 dataset than Inception-v3; as such, it is reasonable to claim that they may also suffer on more-difficult datasets in general. While their advantages in runtime efficiency are desirable, the lack of accuracy may make them hard to effectively utilize when the target task is sufficiently difficult to model. In this regard, the use of the user habit system in section 5.7 would be key to bolstering their accuracies, since the incorporation of context has been shown to be critical in improving accuracy. This scheme would serve to not only reduce the accuracy-gap significantly, but also enable the advantages in runtime efficiency at the same time.

Second, these results are also only true when using solely the CPU of the mobile device. As previously mentioned, it is important to consider performance on devices that are more ubiquitous by present-day standards. Moving forward, the ubiquity of newer Android devices, which are capable of supporting NNAPI or GPUs for computational speed-ups, will increase; at that time, it is important to re-evaluate the performance of various networks on mobile platforms. These measurements are likely to change since certain operations can be optimized and implemented more efficiently depending on both the API and hardware. Nonetheless, in general, these measurements remain good guidelines to consider for effectively deploying trained CNN classifiers to mobile devices in present-times.

5.9 Summary

In this chapter, the efficacy of DenseNet and MobileNetV2 for mobile food recognition is evaluated and quantified. The effort is split largely into three parts: exploring different training approaches to maximize accuracy, evaluating the proposed

user-habit system for further bolstering accuracy, and evaluating their runtime performance on mobile devices.

The networks are first trained on the UEC Food-100 and -256 datasets, including the wide-slice variants as per section 4.3.3, and consider evaluations of various training methodologies to achieve highest accuracy. Transfer learning was found to be a highly effective prescription for all proposed networks, and was instrumental in achieving good performance. Thus, weights were initialized with ImageNet pre-trained weights when possible, and He initialization [54] otherwise.

Then, the fully-connected softmax layer was trained by exploring different image augmentations schemes according to section 4.4.2. The major finding here was that strong results are achievable using minimal data augmentation; specifically, the augmentation scheme of using a single image comprising all desired augmentations and distortions not only facilitated the acquisition of good weights, but also performed the best of all the schemes evaluated. The finding is similar to that in [44], where it was found that smaller networks in general do not struggle as much with overfitting as larger counterparts. Thus, an additional benefit and general takeaway is that the proposed networks seem to be capable of being trained effectively with minimal augmentation following the use of pre-trained weights, which is highly desirable due to the implications in reduced training time and faster experimental iteration.

Following the evaluation of various data augmentation schemes, various fine-tuning procedures were evaluated, from unfreezing the weights of the deepest 25% of layers up to 100% of them. It was shown that fine-tuning across all weights was the most effective method, and as such, all of the networks' weights were further

fine-tuned to improve accuracy, and the best achieved accuracies were reported and discussed.

The DenseNet, wide-slice DenseNet, MobileNetV2, and wide-slice MobileNetV2, achieved 82.6%, 82.1%, 79.5%, and 79.5% on UEC100 respectively, and 68.2%, 68.5%, 66.5%, and 66.8% on UEC256 respectively. The DenseNets outperformed the MobileNets by 2-3%, but can be partially explained by the larger amount of parameters instead of just an inherent ability to learn better representations from architectural differences. They achieve 12% less accuracy on the larger dataset, which may be explained by the increase in classes leading to an increased need for stronger learned representations. The re-designed wide-slice variants were found to bring negligible benefits, and is possible that the benefits it demonstrated in [94] were due to how computationally-intensive the branch is; the efficient variant was incapable of bolstering performance in a similarly-observed way. It was then discussed that the branch may have needed to retain more of its original size and computational complexity, and was minimized too drastically in this work. An alternative view was also discussed that the branch may remain too computationally-intensive for the purposes of mobile deployment; if the excellent performance is contingent on the use of a large number of parameters, then it may not be suited for mobile deployment due to the computationally-constrained nature of mobile devices. Nonetheless, a larger note is made that its published results indicate that its observed benefit may come from a regularizing effect, which is not compatible with smaller networks in general given that they do not require much in terms of regularization, especially at an excessive computational cost; to clarify, the published accuracy of the standalone branch was

very low, while it facilitated excellent results only when coupled with a wide ResNet (WRN) – a larger network that is known to exhibit overfitting tendencies similar to most CNNs [42].

Noting that the benefits of the wide-slice branch were negligible, the results of the DenseNet and MobileNetV2 are then compared to other published results on the same datasets in section 5.6. It was demonstrated that the proposed networks are indeed more parameter-efficient, excluding the finding that GoogLeNet is slightly better than DenseNet in this regard, albeit less accurate. In addition, MobileNetV2 was found to be strictly better than GoogLeNet given its higher accuracy, higher parameter-efficiency, and lower parameter-cost, making it the superior option in almost all circumstances. In terms of accuracy, the proposed networks performed very well on UEC100, with DenseNet exceeding the accuracy of Inception-v3 despite it having 3-times the parameters at 22 million. MobileNetV2 only lagged by about 3%, but has about 5-times less parameters. The other networks performed better, but are significantly larger with the next closest having 40 million parameters, making Inception-v3 the best direct comparison. On UEC256, the proposed networks perform noticeably worse, highlighted by the observation that the DenseNet is 8% less accurate than Inception-v3. An explanation was provided that the reduced parametric capacity may introduce limitations when the target task is sufficiently-complex to model and represent. This finding makes it prudent to explore other avenues in terms of training methodologies and optimizations; for example, it may be beneficial to explore more image augmentation schemes, and other optimizers – and corresponding

hyperparameter settings – like RMSProp [59] may yield increased benefits, as it was successfully used to train MobileNetV2 on ImageNet [10].

Following the evaluation of training procedures, the proposed user-habit system was evaluated and found to be effective; on the sample diet, several challenging classes – in that they are visually similar to other classes on the same dataset, like *fried rice* and *pilaf* – were classified both ignoring and using this contextual information. Performance gains were seen up to 14% and 11% for DenseNet and MobileNetV2 respectively, and effectively highlights the importance of context when performing food recognition. An additional measure of encouragement was found in how these results were achieved despite purposeful increases in difficulty to the task, where ground-truth crops were not extracted, and several pictured food items contained the presence of multiple food classes for which the classifiers were trained. However, it was noted that for the proof-of-concept experiment, the hyperparameters were tuned to fit the sample diet, which was created to be highly consistent and exhibit a good degree of regularity and routine; this may however not be true for all users. Thus, several options to improve the flexibility and modifiability of the system are discussed, although their evaluations are left largely for future work; that said, these are very important considerations to keep in mind when it comes to inferring general conclusions, or proceeding with practical deployment.

Finally, the runtime performance on mobile devices were evaluated for the DenseNet, MobileNetV2, and Inception-v3 – which served largely to establish a baseline since it was the closest competitor across all similar work in section 5.6. They had runtime memories of 121MB, 107MB, and 200MB, and inference times of 720ms,

225ms, and 1080ms, respectively. It was found that the MobileNetV2 architecture may be much better suited for inference when considering the use of solely a CPU. In addition, although the DenseNet is notably inferior in inference time, it is still more accurate than MobileNetV2, and improves on Inception-v3 by about 80MB and 300ms in runtime memory and inference time, respectively. As such, there is a serious trade-off to consider between significant improvements to inference time, and higher accuracy. It was further noted that DenseNet and MobileNetV2 suffered in terms of accuracy on the UEC256 dataset, and although their advantages in runtime performance are desirable, the lack of accuracy may make it difficult to use them when the target task is difficult to model. As such, the importance of NNAPI or GPU availability is highlighted, as is the importance of the user habit system proposed in this work.

CHAPTER 6
Conclusion

This thesis proposed, and investigated, various approaches that improve the runtime efficiency and accuracy of current food recognition approaches, that in turn facilitate their real-world deployment on mobile devices. The first main contribution is training the DenseNet [9] and MobileNetV2 [10] convolutional neural networks (CNNs) to achieve state-of-the-art performance, in parameter-efficiency, on the UEC Food-100 (UEC100) [87] and UEC Food-256 [88] (UEC256) datasets; they are also especially accurate in the case of UEC100 – achieving 82.6% and 79.5% respectively – where the DenseNet notably outperforms Inception-v3 by 1.1% despite having 3.1 times less parameters. In addition, their runtime performance on mobile devices is quantified and shown to be highly efficient, where they use 39.5% and 46.5% less memory, and perform 33.3% and 79.2% faster inference, respectively, compared to the baseline of Inception-v3.

There are several important considerations to reiterate regarding these results. First, while MobileNetV2 is most parameter-efficient by a larger margin, GoogLeNet [31] slightly outdoes DenseNet on both datasets; however, it is strictly outperformed by MobileNet, and is 5.4% and 4.4% less accurate than DenseNet on UEC100 and UEC256 respectively. Second, the networks are noticeably-less accurate on UEC256, despite still being state-of-the-art in parameter efficiency; DenseNet and MobileNetV2 achieve 68.2% and 66.5% respectively, with the highlight comparison

of Inception-v3 now outperforming DenseNet by 8.0% with a 76.2% accuracy. It was discussed that the poorer accuracy may be due to limitations in representational capabilities as a consequence of less parameter usage, and that it may be necessary to explore other avenues, like the RMSProp [59] optimizer – which was used in the successful training of MobileNetV2 on ImageNet [10] –, for accuracy gains. Third, the wide-slice branch [94] was found to be of marginal benefit – increasing accuracy by a maximum of 0.3% – following a proposed efficient re-design that enabled mobile feasibility. It was discussed that its benefit was likely through a regularizing effect on the core network, which the proposed networks – by virtue of being highly parameter-efficient as is – do not seem to struggle as much with, and thus benefit from either. Fourth, numerous training approaches are explored for the proposed networks, and an atypical set of best practices – involving the use of single-image augmentation and full fine-tuning of pre-trained weights – is recommended by this thesis; this atypical set may lend credence to the observation that the smaller networks have less inherent difficulties with overfitting – a claim corroborated in [44] – which may have implications in easier and faster training. Lastly, the mobile benchmarks pertain to CPU-only devices, and does not account for developments such as the NNAPI [124] and on-device GPU – which are highly contingent to runtime performance due to optimizations they provide; thus, as these enabled-devices become more ubiquitous, it is important to re-evaluate the provided benchmarks to take them into account.

The second main contribution is the proposal of a novel user habit system that utilized user context to increase the accuracy of DenseNet and MobileNetV2, by 14.3% and 10.7% respectively, on the proof-of-concept evaluation; in addition to

improving accuracy, it was explicitly designed to be suited for mobile applications by virtue of being relatively simple and extensible. That said, the results do not necessarily generalize to all users as the evaluation was conducted on a proof-of-concept user – whose diet was highly regimented, strict, and habitual – with several of the parameters fine-tuned for this particular profile. To generalize, it is important to adjust these parameters to different profiles; the first of two primary approaches involves allowing the user to tune them at their own discretion. A user-friendly format for doing so may involve a questionnaire, which can include setting a slider value for a series of statements such as, "I often eat the same dishes for breakfast"; alternatively, the user may select from a pre-defined list of profile pre-sets that best correspond to their particular habits. While simple, these methods rely on user input, which is a source for human error; as such, the second approach involves algorithmically, and retroactively, analyzing the user's diet to determine optimal settings. A simple option may involve profiling the user into several pre-sets, as before; alternatively, a more complex one may target the user's habits with more specificity, using a more sophisticated algorithm. Nonetheless, these are important factors to explore further in future works.

The third main contribution is the consideration of practical diet-tracking systems to guide the development of future work. As dietary improvement is, presently and largely, being addressed with manual diet-tracking processes, food recognition likely only constitutes a single facet to an improved process; more realistically, a fully realized diet-tracking system is necessary for seeing larger-scale dietary improvement. This thesis identifies several key features, including automatic calorie

estimation, and explores many such relevant topics to highlight current-and-future efforts to their development; the thesis then formulates its own discussions that are critical to it providing guided recommendations for launching future efforts into each respective domain.

In addition to these contributions, the thesis also provides a brief background on both CNNs and food recognition. The former focuses on elaborating the proposed architectures, and providing context for their use – and for the use of certain techniques in this thesis; the latter highlights challenges in food class variation, and provides a brief overview of related work.

Ultimately, there is a growing concern of diet-based health complications, such as obesity and diabetes, in the world at large; as such, it is crucial to continue developing novel methods to mitigate these. To reiterate, it is a firm belief of the author that health is a very precious commodity that people often neglect; with the unrelenting busyness of life – which only seems to grow day-by-day – it is all the more imperative that new and convenient ways are developed to help maintain a healthy lifestyle. It is hoped that the contributions of this thesis toward mobile food recognition facilitate such convenience; its continued development, and subsequent practical employment, would undoubtedly improve the diet tracking processes that in turn enhance the wide-scale self-management and regulation of dietary intake. It is further hoped that the findings presented here can pave the way to the development and deployment of a readily-available and highly-useful application, and it is the author's sincere wish that the efforts presented here can facilitate a reduction, if not elimination, to the growing health concerns that stem from diet.