Accessible Retail Shopping For The Visually Impaired Using Deep Learning

by

Akshar Patel

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Sethuraman Panchanathan, Co-Chair
Hemanth Kumar Demakethepalli Venkateswara, Co-Chair
Troy McDaniel

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Over the past decade, advancements in neural networks have been instrumental in achieving remarkable breakthroughs in the field of computer vision. One of the applications is in creating assistive technology to improve the lives of visually impaired people by making the world around them more accessible. A lot of research in convolutional neural networks has led to human-level performance in different vision tasks including image classification, object detection, instance segmentation, semantic segmentation, panoptic segmentation and scene text recognition. All the before mentioned tasks, individually or in combination, have been used to create assistive technologies to improve accessibility for the blind.

This dissertation outlines various applications to improve accessibility and independence for visually impaired people during shopping by helping them identify products in retail stores. The dissertation includes the following contributions; (i) A dataset containing images of breakfast-cereal products and a classifier using a deep neural (ResNet) network; (ii) A dataset for training a text detection and scene-text recognition model; (iii) A model for text detection and scene-text recognition to identify product images using a user-controlled camera; (iv) A dataset of twenty thousand products with product information and related images that can be used to train and test a system designed to identify products.

DEDICATION

Dedicated to

My God, My Guru, My Professors, My Mentors, My Family, and My Wife.

ACKNOWLEDGMENTS

Special Thanks To

Dr. Sethuraman Panchanathan

Dr. Hemanth Kumar Demakethepalli Venkateswara

Dr. Troy McDaniel

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.3 billion people live with some form of visual impairment, with around 250 million people who have visual impairment ranging from moderate to complete blindness and close to 826 million people who have near vision impairment.

People who have such visual impairment, find it incredibly arduous to shop physically in stores due to their disability resorting to shopping with a friend, relative, shopping store assistant, creating shopping lists, and giving it to others who would buy it for them. None of these alternatives improve the accessibility of shopping for the visually impaired. Hence, such people can resort to not shopping from retail stores and online stores to do their shopping.

Considering close to one billion people suffer from a varied range of visual impairments that hampers their experience of shopping physically in stores, retails are losing access to close to 15% of the world's population. These people constitute a vast untapped market for retail spaces.

On the other side, online retailers have made huge strides in making their platforms accessible to virtually anyone who wants to shop. These online retailers have used the web and the mobile platform to their advantage. Both the web and mobile, provide accessibility features natively, which makes the online retailers' website and mobile apps very easy to navigate for people with different impairments. There also has been much research into making websites and mobile apps accessible.

This improvement in accessibility, however, has not extended for accessibility in retail spaces. While they have done lots to improve the experience of a usual customer, improvements in terms of making their spaces truly accessible and making shopping a seamless operation for people with impairments is a far-off dream.

With retail stores facing massive competition from online stores and with increasingly drying footfall and subsequent closure of a lot of retail spaces, making their stores more accessible to people with impairments may be the strategy that saves them from fading out of existence.

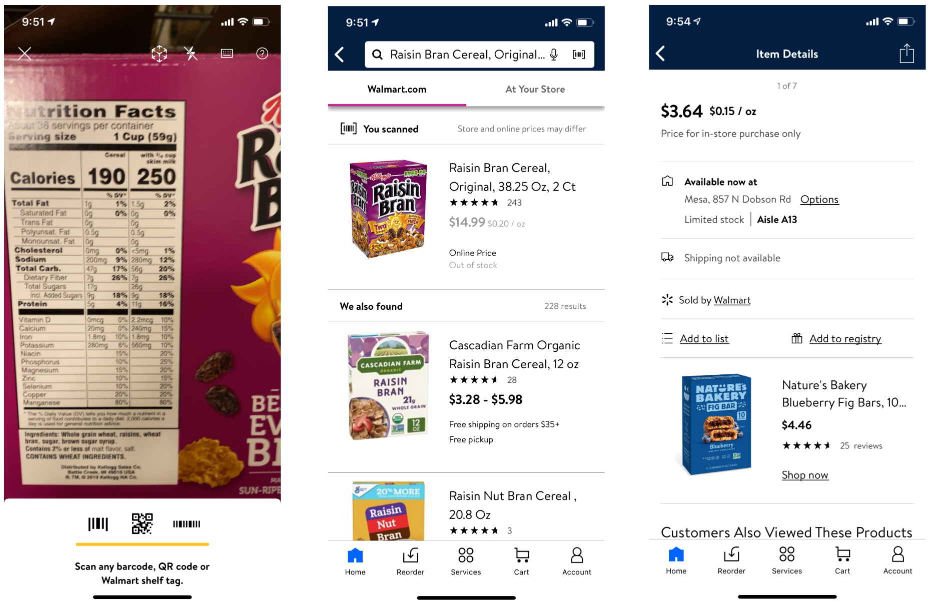
Recent developments around the world surrounding the pandemic of Covid-19 has caused people to stock up on daily necessities, food, and emergency utilities. Online stores have been out of stock of a lot of necessary items. This situation and other "Acts of God" in history have shown that in certain situations, retail stores are the only avenue to buy the necessities even if it means standing in a line and waiting to get our hands on the required products. These situations increase the importance of making retail stores accessible to everyone involved.

One of the ways the mega retail stores are making their stores more accessible is with an accompanying mobile app. Retail giants such as Walmart, Target, Costco, and more have created mobile apps that not only help a customer to shop through the mobile app and get the goods delivered at home, the app also provides information that helps the people to shop while visiting one of their retail stores. A few ways these apps help the users are as follows:

- Ability to scan the product barcode on either the product itself or on the shelf to get information about the product.

- Search the product inside the app and getting to know the location of the product inside the store.

- If a product is unavailable in a store, find a store around the location of the user, in which it is available.



Raisin Bran Cereal, Original...

Walmart.com | At Your Store

You scanned — Store and online prices may differ

Raisin Bran Cereal, Original, 38.25 Oz, 2 Ct
★★★★★ 243
$14.99 $0.20 / oz
Online Price
Out of stock

We also found — 228 results

Cascadian Farm Organic Raisin Bran Cereal, 12 oz
★★★★★ 28
$3.28 - $5.98
Free shipping on orders $35+
Free pickup

Raisin Nut Bran Cereal , 20.8 Oz
★★★★★ 3

Item Details

1 of 7

$3.64 $0.15 / oz
Price for in-store purchase only

Available now at
Mesa, 857 N Dobson Rd Options
Limited stock | Aisle A13

Shipping not available

Sold by Walmart

Add to list — Add to registry

Nature's Bakery Blueberry Fig Bars, 10...
$4.46
★★★★★ 25 reviews
Shop now

Customers Also Viewed These Products

(a) (b) (c)

Figure 1.1: Images (a) and (b) demonstrate the accessibility feature in the Walmart app, which identifies the product based on the barcode. Image (c) demonstrates the feature in Walmart app which shows in which the nearest store is the product located and in which aisle.

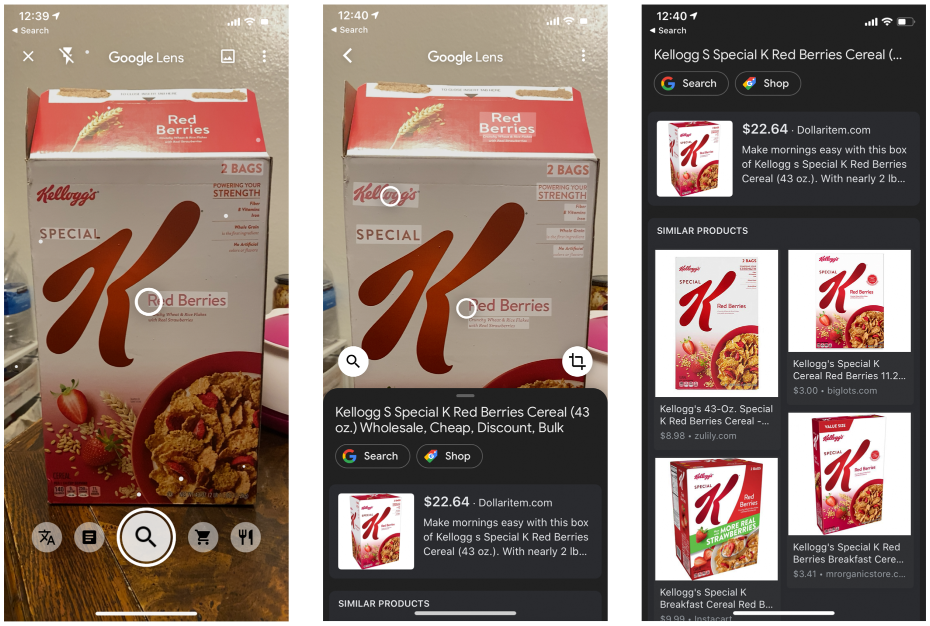
- Keeping track of your receipts so that the customers don't have to have physical copies of the receipts with them in case they want to return the product or get it repaired under warranty.

However, not all retail stores have such accessibility features. A majority of apps made by retail stores are geared towards making the customer shop through it as an online store rather than use it as an accessible way to shop in their retail stores.

One of the disadvantages of having only barcodes as the only accessible way of identifying a product for visually impaired people who are shopping in retail stores is that the placement of barcodes is different in different products and different even in the same product with different packaging. Having people move and rotate the product in all directions trying to scan a barcode with the product on the one hand and the mobile in another is not an efficient way of tackling this problem and is a recipe for disaster.

Microsoft has an app on Google's Play Store and Apple's App Store called "Seeing AI," which beeps with higher frequency the closer it thinks you are to a barcode. After experimenting with it, it doesn't look like it would solve the problem of making identifying the product more accessible.

Google search has one very accessible functionality called Google Lens, which helps in searching what a product is by taking an image of the product and searching the web-based on the text as well as the features found inside the image. This solution is very useful in improving the accessibility for visually impaired people while they are shopping as it helps in identifying the product and removes the burden of looking for the barcode off the user.



Kellogg S Special K Red Berries Cereal (43 oz.) Wholesale, Cheap, Discount, Bulk

$22.64 · Dollaritem.com
Make mornings easy with this box of Kellogg s Special K Red Berries Cereal (43 oz.). With nearly 2 lb...

Kellogg S Special K Red Berries Cereal (...

Kellogg's 43-Oz. Special K Red Berries Cereal -... $8.98 • zulily.com

Kellogg's Special K Cereal Red Berries 11.2... $3.00 • biglots.com

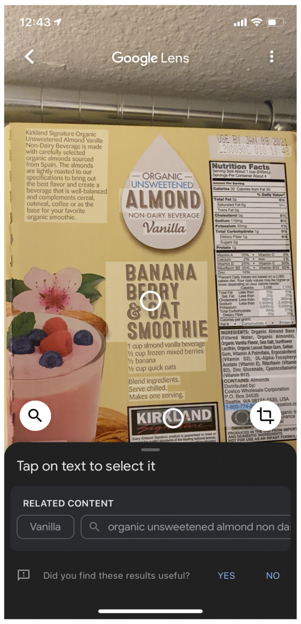
Kellogg's Special K Breakfast Cereal Red B... $9.99 • Instacart

Kellogg's Special K Red Berries Breakfast Cere... $3.41 • mrorganicstore.c...

(a) (b) (c)

Figure 1.2: Image (a) shows the Google Lens trying to find text on the product. Once the user clicks on the search button, it identifies the product as shown in Image (b). Image (c) shows the app shows the search results for the product which show from where the product can be bought.

As seen in Figure (1.2), While Google lens works very well to identify different products, it can be improved further, as is evident from Figure (1.3). Also, from



(a)

Figure 1.3: Image (a) shows the Google Lens App failing to identify a product in focus.

Figure(1.2), based on the results shown after the product is identified, it seems like the aim of the tool is to help you buy the product online instead of giving you more information about the product.

This dissertation looks into making progress in the direction of improving accessibility for the visually impaired by attempting to identify the different products in front of the person.

1.1 Goals And Motivations

The goal of this dissertation is to propose ways to classify images of products into different categories and create datasets that can be used to improve the classification. It seeks to look into different ways classification can be achieved, the advantages and disadvantages of different solutions, and potential ways of improving the proposed solutions. It also outlines the different directions for this research in the future.

1. *AI For Good*: AI for Good[1] was a challenge created by Microsoft in 2019 that challenged developers, students and data scientists to use AI to tackle some of society's greatest obstacles and build a more sustainable and accessible world. One of the focus of the challenge was to empower people with disabilities by using AI for hearing, seeing, and reasoning with increased accuracy to help people with the everyday task.

2. *AI for Accessibility*: Microsoft has another grand program called AI for Accessibility[2] which funds projects that use AI to improve the human capability for more than a billion people around the world with a disability. A few projects under this program that have been funded are (i) Making people aware of their surroundings for visually impaired people using their mobile device's sensors and cameras. (ii) Chatbot for people with disabilities.

3. *AI for Social Good*: This is a social program[3] from Google which revolves around using AI to approach problems to improve the people's lives.

4. *Personal Motivations*: As a vegan following the sattvic diet, many food products don't follow the dietary restrictions set by the sattvic diet. It is tough to identify these products without doing sufficient research. Many people also follow similar diets and don't have the necessary information while doing their shopping, either in retail stores or online. A system that helps them identify different products and shows them if they're as per their dietary requirements is sorely needed.

[1]`https://www.microsoft.com/en-us/ai/ai-idea-challenge`

[2]`https://www.microsoft.com/en-us/ai/ai-for-accessibility`

[3]`https://ai.google/social-good/`

1.2 Contributions

The contributions of the dissertation are as follows:

1. A dataset of 60 cereal box categories containing 1000 images to be used for classification.
2. A trained CNN classifier that classified an image into one of 60 cereal box categories.
3. A system that takes an image and using scene text recognition with character recognition classifies the image into 60 cereal box categories.
4. A dataset to improve the scene text detection and recognition models.
5. A dataset of twenty thousand products that contains the name, images, brand, marketing text, specifications, and nutrition of the products. This dataset would then be used to test the accuracy of the system designed to identify the products.

1.3 Dissertation Outline

Chapter 2 provides a literature review of papers and technologies used to implement and deploy the cereal image classifier and scene text recognition based classifier.
Chapter 3 provides an overview of creating a Resnet34 image classifier and its accompanying dataset.
Chapter 4 provides an overview of using text detection and scene text recognition to classify the cereal box images.
Chapter 5 provides an overview of creating a dataset for improving the performance of scene text detection and recognition models.
Chapter 6 provides an overview of creating a dataset of twenty thousand products that can be used to measure the accuracy of product identification systems.

Chapter 2

LITERATURE REVIEW

The field of deep learning and computer vision has come a long way. Open image datasets such as ImageNet (Russakovsky *et al.* (2014)), provide a large pool of images, with more than twenty thousand categories, to train a neural network on and compare the results with other neural networks. In addition to ImageNet, datasets such as COCO(Lin *et al.* (2014)) and Pascal VOC(Everingham *et al.* (2009)) also provide images that can be used to train neural networks for tasks such as image classification, object detection, instance segmentation, semantic segmentation and panoptic segmentation. An advantage of such challenges is that models trained on ImageNet, COCO, Pascal VOC or any other open data, can be used as a base for transfer learning to reduce the amount of time it takes to train a neural network for other such computer vision tasks.

Along with indispensable libraries such as OpenCV (Bradski (2000)) for achieving computer vision tasks, libraries to create neural networks with ease have also been a part of development in this field. Libraries such as Tensorflow (Abadi *et al.* (2015)) and PyTorch (Paszke *et al.* (2019)) have been popular in the field of deep learning with loads of documentation, tutorials, implemented papers and pretrained models, additional libraries on top of them to achieve additional task available in the open. One such library on top of PyTorch gaining popularity nowadays is FastAI (Howard and Gugger (2020)). On top of PyTorch, FastAI includes methods to easily save, export and load models, implementations of papers which improve the speed and accuracy of training, visualizing the output of the trained model and much more. This makes it very easy for a newcomer to jump into the field of deep learning.

One of the neural network architectures which is really popular to be used for different computer vision tasks is Residual Networks as mentioned in He *et al.* (2015). ResNets of varying depths are being used as backbones in other neural networks made to accomplish different computer vision tasks. For example, ResNet is used as a backbone in Mask R-CNN He *et al.* (2017) to accomplish Instance Segmentation in image datasets.

In addition to an image classifier, this disseration also works on creating a classifier using scene text detection and recognition techniques. Character Region Awareness for Text Detection or CRAFT, Baek *et al.* (2019b), is used to detect text in an image. The parts of image with text are then extracted and given as an input to a TPS-ResNet-BiLSTM-Attn model as developed by Baek *et al.* (2019a). This model reads the text inside the image segments and retuns the text as output.

Modern tools such as Jupyter notebooks, visual studio code, Anaconda make it easy to write, debug and run python programs. Python libraries such as FastAPI, uvicorn make the task of creating RESTful apis simple and straightforward. In this disseration FASTAPI is used to create a RESTful API that serves the cereal classifier and the scene text classifier model. To consume the RESTful APIs and to demonstrate that this disseration can be incorporated in real world scenarios, I created a mobile app using a cross platform framework known as Flutter. Flutter makes it very easy to write the code once and run it on all the major app platforms such as Android, iOS, Web, Windows, Mac and Linux. There are ports of Flutter which allow it to run on embedded systems such as Raspberry Pi as well.

Chapter 3

CEREAL CLASSIFIER

The objective here is to Create a deep convolutional neural network that can classify the photo of cereal boxes into one of the 60 predefined categories. We added one additional category called ‘None’ which would capture all the images not belonging to any of the cereal boxes.

3.1 Creating A Dataset

A deep neural network would require sufficient images of the cereal boxes to train it to classify 60 different cereal boxes. The different sources of images that I used were Google Images, Online Store Images and manually taken images. A detailed explanation of each technique and their associated challenges is described in detail in the following sections.

3.1.1 Google Images

Initially, I decided to use Google Images to download the images of the cereal boxes. The following are the steps I took to download the images:

- I used node.js and the puppeteer library to automate the search for images of a particular cereal box on Google Images.
- Using the puppeteer library, I extract the URLs of the images displayed on the Google images. The number of URLs I extract is 100.
- Once all the URLs have been extracted, I download the images and save them to a folder corresponding to the name of the cereal.
- If the downloaded image is less than 512x512, I remove it from the dataset.

As shown in Figure (3.1), Even though downloading images for a particular food item using Google Image can be easily automated, I faced the following problems with the above approach:

- Quality of Image is not always as to what I required for the classification task.
- Images of the cereal box from all around the world, with different packaging, is also mixed in with the desired Images.
- Images not of the cereal box were also in the mix of the downloaded images.
- Images with multiple cereal boxes of the same company in the same image.

Hence, while using this approach, the work required to improve the dataset after the images have been downloaded is a lot. The number of good images that we get from this approach is also very low.

3.1.2 Online Store Listing

Walmart.com is an excellent source for product information such as the name of the company making the product, the weight/volume/size, the ingredients, the nutrition information of the product and other meta-information such as the marketing writeup and reviews about the product. It also houses high-quality images of the product from different angles for a lot of products as shown in Figure (3.2).

I extracted the information from Walmart.com for the 60 cereal boxes using the following steps:

- Using node.js and puppeteer go to the URL of the cereal box on Walmart.com.
- Once the page has been fully rendered, extract the title, price, marketing writeup, ingredients, additional info table, nutrients, and image URLs from the page using puppeteer library.
- Download the images from the URLs gathered from the previous steps.
- Save the images and the JSON file containing the product information in a folder with the name corresponding to the name of cereal.

The naming convention for the folder of the cereal box is as follows:

```
{company_name}-{cereal_name}
```

3.1.3 Manually Taking Images

The previous method of scraping data from Walmart.com is great for extracting information about the product. However, even with both the previous method combined, we don't have enough images to train a neural network.

I decided to get more images of the 60 cereal boxes by going to the Walmart store and capturing it myself. I took 10 images of each cereal box where 5 images are of the cereal box sitting on the shelf and 5 images are of the cereal box being held in the hand as shown in Figure (3.3).

3.1.4 Compiling The Images

All the valid images from the above three steps are compiled and saved in their respective folder. Please refer to Table (3.1) for Image counts for different cereal boxes.

Category Name	Count of Images
cascadian_farm_organic − cinnamon_crunch	18
cascadian_farm_organic − fruit_nut_granola	15
general_mills − apple_cinnamon_cheerios	15
general_mills − blueberry_cheerios	15
general_mills − cheerios_cinnamon_oat_crunch	14
general_mills − cheerios_whole_grain_oats	41
general_mills − chocolate_cheerios	17
general_mills − chocolate_lucky_charms	14
general_mills − chocolate_peanut_butter_cheerios	14
general_mills − chocolate_toast_crunch	14
general_mills − cocoa_puffs	12
general_mills − cookie_crisp	14
general_mills − corn_chex	14
general_mills − drumstick_classic_vanilla	14
general_mills − fiber_one	14
general_mills − fiber_one_hone_clusters	14
general_mills − french_toast_crunch_syrup_cinnamon	15
general_mills − golden_grahams	14
general_mills − honey_nut_cheerios	38

general_mills − honey_nut_cheerios_medley_crunch	15
general_mills − honey_nut_chex	14
general_mills − lucky_charms	14
general_mills − maple_cheerios	15
general_mills − raisin_nut_bran	14
general_mills − rice_chex	15
general_mills − very_berry_cheerios	14
general_mills − wheaties	12
great_value − bran_flakes	18
great_value − raisin_bran	15
kashi − goplay_honey_almond_flax_crunch	12
kashi − gorise_original	12
kashi − organic_blueberry_clulsters	12
kelloggs − all_bran_original	12
kelloggs − chocolate_frosted_flakes	13
kelloggs − corn_flakes	22
kelloggs − froot_loops	9
kelloggs − froot_loops_marshmallows	12
kelloggs − frosted_flakes	45
kelloggs − frosted_mini_wheats_original	11
kelloggs − krave_chocolate	12
kelloggs − pop_tarts_cereal	13
kelloggs − raisin_bran	23
kelloggs − raisin_bran_crunch	13

kelloggs − rice_krispies	41
kelloggs − special_k_chocolatey_delight	12
kelloggs − special_k_fruit_yogurt	12
kelloggs − special_k_original	39
kelloggs − special_k_protein	12
kelloggs − special_k_red_berries	35
kelloggs − special_k_vanilla_almond	13
none	84
post − cocoa_pebbles	12
post − fruity_pebbles	13
post − great_grains_crunchy_pecan	13
post − honey_bunches_of_oats_almonds	10
post − oreo_os	13
post − sour_patch_kids	13
quaker − simply_granola_oats_honey_almonds	12
quaker − simply_granola_oats_honey_raisins_almonds	12
total − whole_grain_flakes	14

Table 3.1: Number Of Images For In Cereal Categories

3.2 Training A Neural Network

3.2.1 Selecting A Deep Learning Framework

The deep learning landscape has a lot of different frameworks that one can use to train a CNN classifier. While TensorFlow and PyTorch were more than capable of implementing such a classifier, I selected the Fast.ai framework. Fast.ai is built on top of PyTorch and has the following advantages:

- Intuitive API for loading data
- Helper functions to visualize the training data, debugging the model, visualize the result and much more.
- It has implementations of papers which improve the speed of training and for fine tuning the model.
- It has powerful data augmentation utilities which use the GPU to augment data which reduces the time it requires for data augmentation on the fly.
- Since the framework is built on top of PyTorch, all the advantages of using PyTorch apply to Fast.ai.

3.2.2 Selecting A Backbone For The Neural Network

The criteria for selecting a backbone were as follows:

- The cereal classifier model should be performant on CPUs.
- The ImageNet pretrained weights for the backbone should be available.
- The resulting network should have a top 1 accuracy of >90%.

Resnets are a powerful backbone network for a classification task. ResNet50 scores 81% top 1 accuracy in the ImageNet image classification task. task by transferring the learnings of ImageNet onto our own model. Hence, I selected ResNet34 to train the model initially since the current problem only has 60 categories to predict and check the results.

3.2.3 Data Augmentations

The following augmentation techniques were applied to the dataset:

- Random Rotation between -90 to 90
- Random Zooming in Image between 1 to 1.2
- Random Lightning and Contrast Change
- Random Symmetric Warping

Each of the above transformations are applied with the probability of 0.75 onto the image. Figure (3.4) shows an output of data augmentation techniques on one single cereal box image.

3.2.4 Training The Model

Once the dataset is in place, the backbone selected and the augmentations configured, the next step would be to train the model.

Using Transfer Learning To reduce the time to train the model, we use a ResNet model pretrained on the ImageNet dataset, remove the last layer of the downloaded model and add our own last layer corresponding to the number of cereal classes for which we're trying to train the model.

Freezing Layers During Training Since we're using a model pretrained on ImageNet, it has already learned to detect a good amount of features that can be reused to detect the cereal classes. However, our last layer is just random weights initially before we start training our model. So initially we can freeze all layers except the last and train only the last layer.

Using Variable Learning Rate Once we reach an accuracy where the last layer cannot be improved further, we can unfreeze all the layers so that they all can be used for training and train the model further. However, since the initial layers have learned features, training all the layers with the same learning rate can potentially overfit the model. Hence, we can train the last layer with a higher learning rate and the other layers can be trained with a lower learning rate.

3.2.5 Results

The model achieved classification accuracy of 96%.

3.3 Deploying The Model

To deploy the cereal image classifier, we would need to export the model so that it can run on the server without the need of additional model training code. To create a system that a user can use to run to classify cereal boxes, we would need to create two additional systems that would allow the user to interact with the trained model.

3.3.1 Creating An Api

First system that needs to be developed to allow the user to interact with the trained model is to create a RESTful API that serves the cereal classifier. It does not have to be a RESTful API as any other Remote Procedure Call (RPC) methods can be used to create a service that allows interacting with the model.

I created a RESTful API service using FASTAPI python framework. The following is the API signature of the endpoint that uses the cereal classifier to classify the images.

```
PATH: /cc

HTTP METHOD: POST

BODY TYPE: multipart/form-data

BODY:

    - file: string($binary)

RESPONSES:

    SUCCESS:

        STATUS CODE: 200

        STRUCTURE: {"category_name": "string"}
```

3.3.2 Creating A Flutter App

To be usable by an end user, I needed a way to create an app or a website that could allow the user to capture an image and get the classified category back.

Flutter framework allows creating a mobile app that can also be run as a website and as a desktop app as well.

Figure (3.5) shows how the app operates.

3.3.3 Putting It All Together

Connecting all the above systems, we end up with a solution that a user can use to classify a cereal box image into one of 60 cereal box categories.

Figure (3.6) shows how the entire system architecture and the flow of data.

3.4 Conclusion

While an image classifier reaches high levels of classification accuracy, we face the following problems when running it in production.

- Adding any new category to the model, means that the model needs to be retrained.
- If categories of products keep getting added, the size of the models would keep increasing reducing their portability. The accuracy and speed of inference of the models would also decrease.
- An image classification model is susceptible to inaccuracies when trying to classify products having similar packaging.

Hence, we need a system that can classify products with high accuracy, does not need to be retrained every time a new product is added to the dataset and is highly portable.

The code, the dataset and the trained model for this project can be downloaded from the GitHub Project Page (https://github.com/aksharpatel47/accessible-shopping).

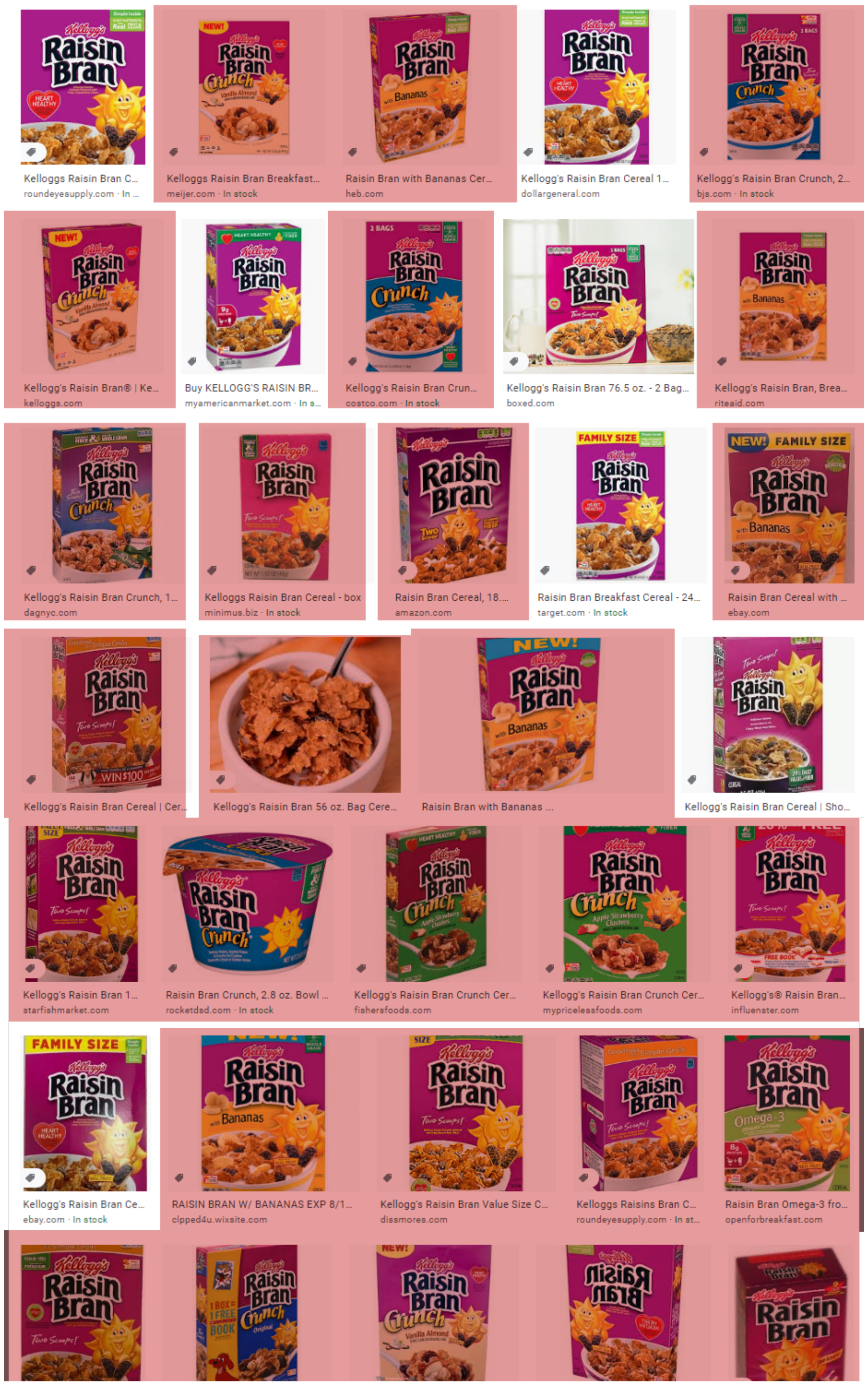

Kelloggs Raisin Bran C...
roundeyesupply.com · In ...
Kelloggs Raisin Bran Breakfast...
meijer.com · In stock
Raisin Bran with Bananas Cer...
heb.com
Kellogg's Raisin Bran Cereal 1...
dollargeneral.com
Kellogg's Raisin Bran Crunch, 2...
bjs.com · In stock
Kellogg's Raisin Bran® | Ke...
kelloggs.com
Buy KELLOGG'S RAISIN BR...
myamericanmarket.com · In s...
Kellogg's Raisin Bran Crun...
costco.com · In stock
Kellogg's Raisin Bran 76.5 oz. - 2 Bag...
boxed.com
Kellogg's Raisin Bran, Brea...
riteaid.com
Kellogg's Raisin Bran Crunch, 1...
dagnyc.com
Kelloggs Raisin Bran Cereal - box
minimus.biz · In stock
Raisin Bran Cereal, 18...
amazon.com
Raisin Bran Breakfast Cereal - 24...
target.com · In stock
Raisin Bran Cereal with ...
ebay.com
Kellogg's Raisin Bran Cereal | Cer...
Kellogg's Raisin Bran 56 oz. Bag Cere...
Raisin Bran with Bananas ...
Kellogg's Raisin Bran Cereal | Sho...
Kellogg's Raisin Bran 1...
starfishmarket.com
Raisin Bran Crunch, 2.8 oz. Bowl ...
rocketdsd.com · In stock
Kellogg's Raisin Bran Crunch Cer...
fishersfoods.com
Kellogg's Raisin Bran Crunch Cer...
mypricelessfoods.com
Kellogg's® Raisin Bran...
influenster.com
Kellogg's Raisin Bran Ce...
ebay.com · In stock
RAISIN BRAN W/ BANANAS EXP 8/1...
clpped4u.wixsite.com
Kellogg's Raisin Bran Value Size C...
dissmores.com
Kelloggs Raisins Bran C...
roundeyesupply.com · In st...
Raisin Bran Omega-3 fro...
openforbreakfast.com

Figure 3.1: Searching for Kellogs Raisin Bran on Google Image Search does not give accurate image results that can be used without verification. The red highlights incorrect search results.

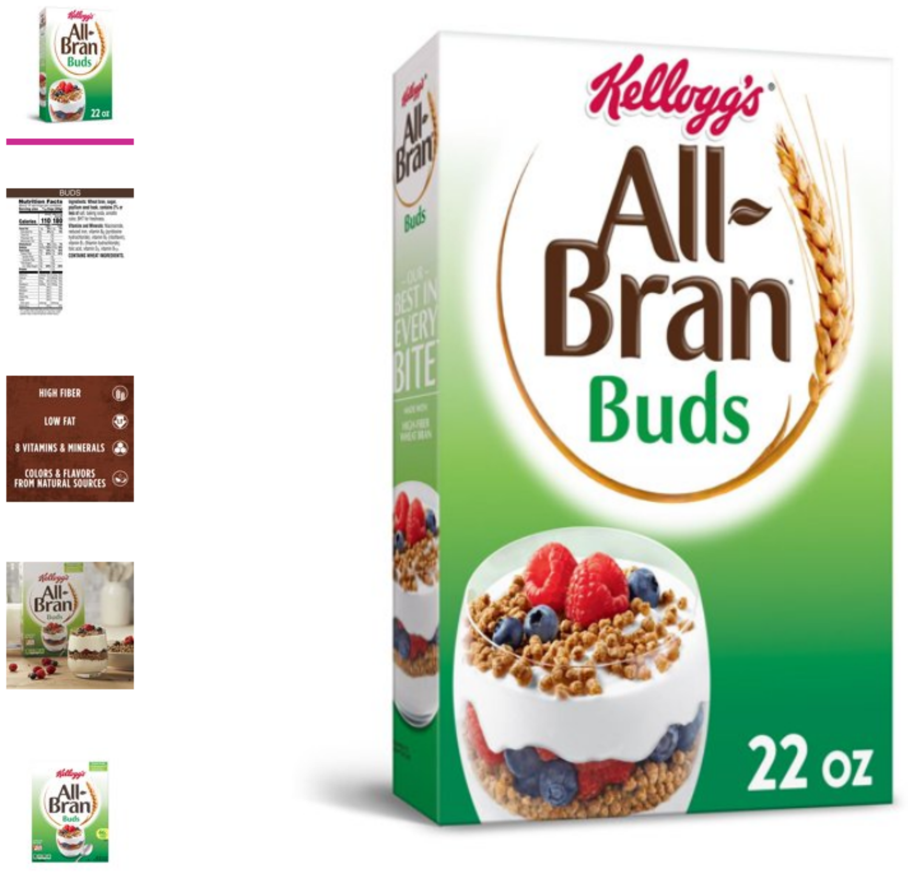


Figure 3.2: Online store Listings showing images of the a cereal box.

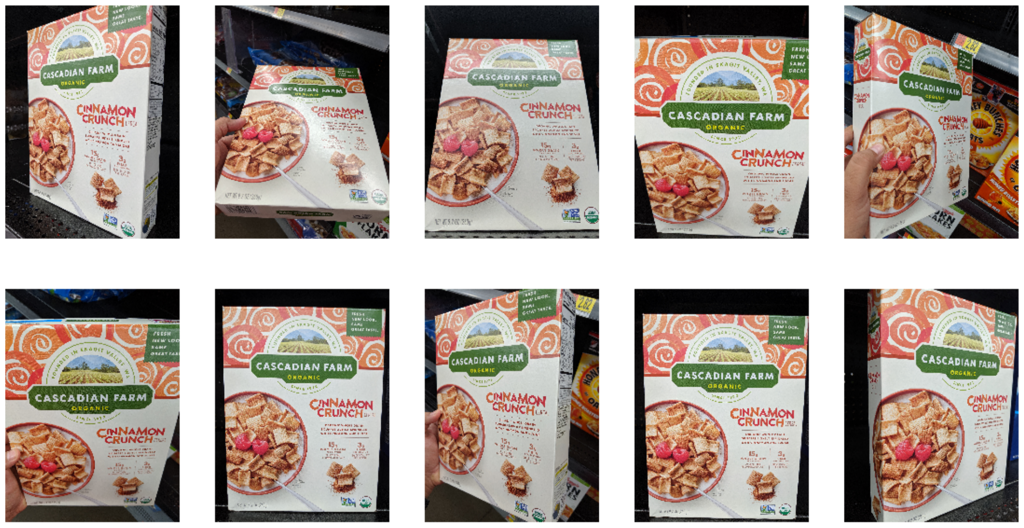


Figure 3.3: Example of images gathered manually for a cereal box category.

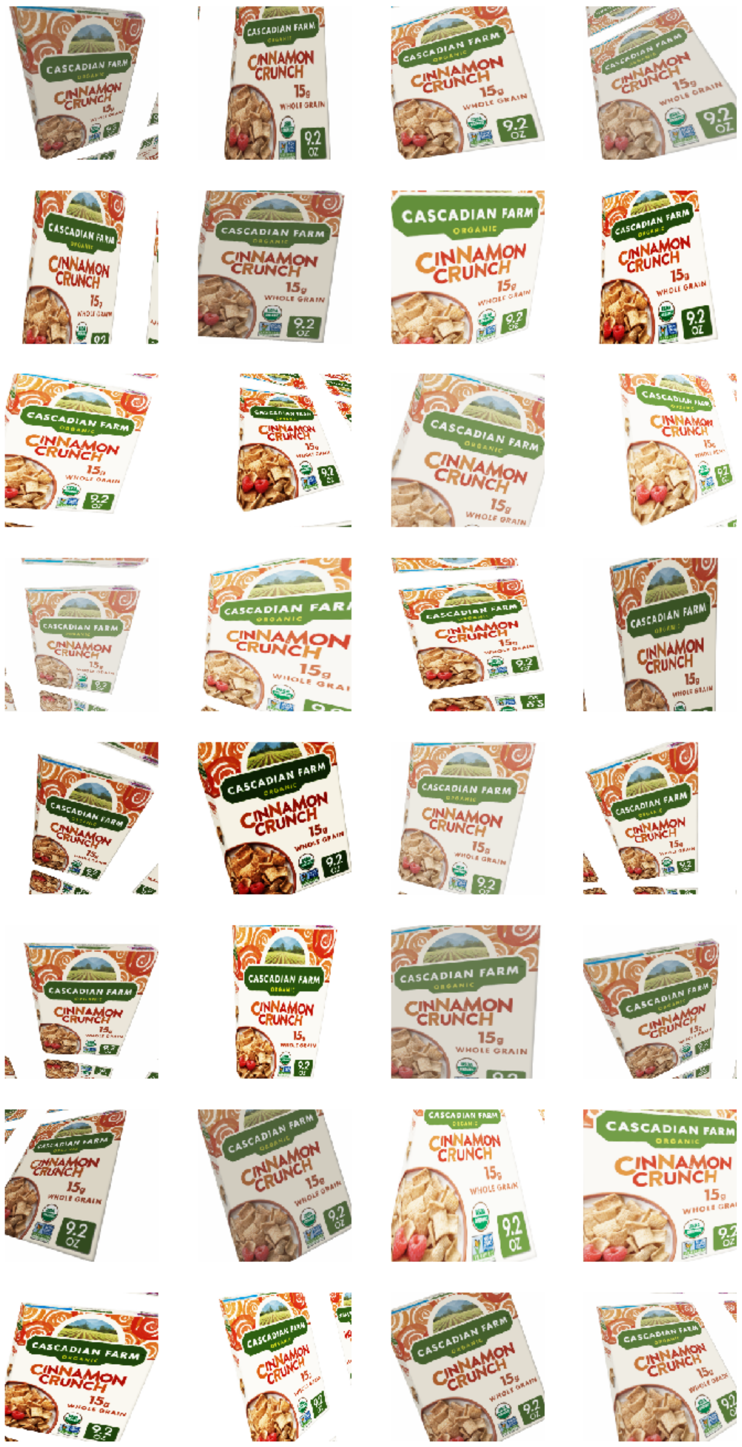


Figure 3.4: An example of how one single image is augmented using different techniques such as Rotation, Zooming, Lightning, Warping with values of predefined range applied with a probability of 0.75.

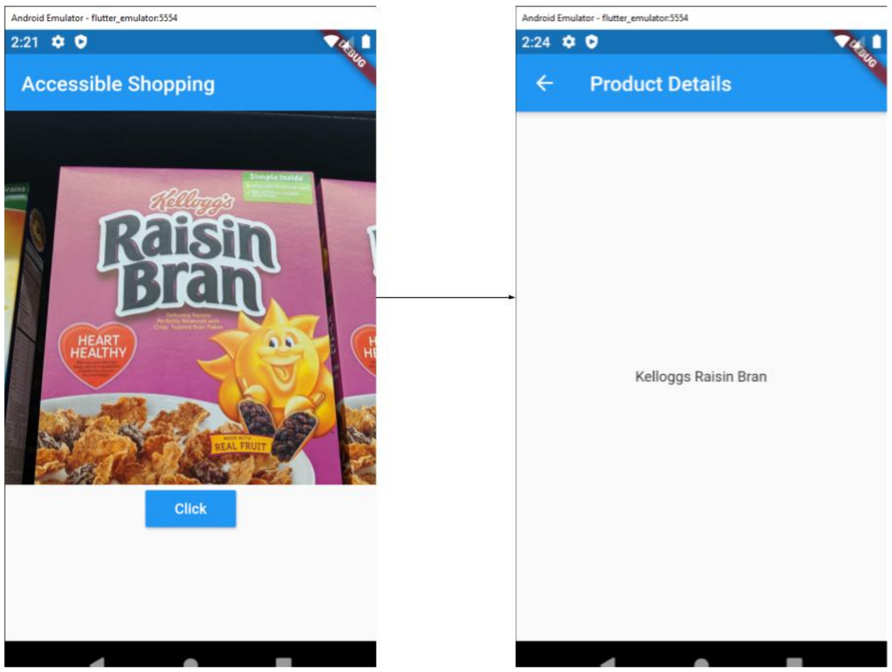

Android Emulator - flutter_emulator:5554
2:21
Accessible Shopping
Click
Android Emulator - flutter_emulator:5554
2:24
Product Details
Kelloggs Raisin Bran

Figure 3.5: Taking an image of the product, the app sends the image to the server, which returns the name of the category of the product which the system predicted. This category is displayed on the next screen.

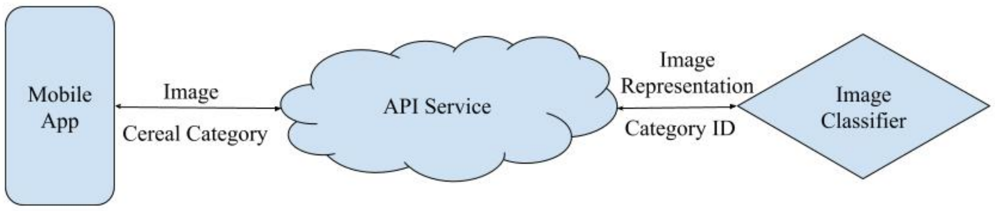


Figure 3.6: Taking an image of the product, the app sends the image to the server, which returns the name of the category of the product which the system predicted. This category is displayed on the next screen.

Chapter 4

SCENE TEXT CEREAL CLASSIFIER

Since a simple Convolutional Neural Network (CNN) classifier is ill suited for my usecase, I needed to find another way to classify an image into 60 cereal box categories. One technique that can be used to reduce the problem of having to train a new model every time a new product is added to the model is to make a multi-class classification model. This model would have different category names such as:

- Name of the brands that make the cereal e.g. Kellogs, Post
- Different types of cereals e.g. Raisin Bran, Red Berries
- Shape of the packaging e.g. Box
- Quantity of cereal in the box e.g. 12oz
- Special labels for the cereals e.g. Vegan, Gluten Free

This would allow the model to learn new products without the need to create a completely new model. However, if a new label needs to be added for a new product I want to classify, a new model needs to be trained with an additional label. Since, this technique cannot be used to scale the solution to thousands of products, a new system needs to be implemented that can accurately classify not only the 60 cereal box categories but tens of thousands of products without the need to train the system everytime a new product needs to be added.

Taking inspiration from Google Lens App, I decided to try to classify cereal box categories using the text recognized from the input image and using the individual words to identify its cereal box category.

4.1 Dataset

The dataset that I used for the scene text cereal classifier, is the same dataset which I used in the cereal classifier from chapter 3. In the initial system, I didn't train any model, but used pretrained models of state of the art neural networks. Hence, all the images from the cereal classifier are part of the validation dataset for the new system.

Since, I did not retrain the models, the accuracy of the implemented system would become a baseline to measure any improvements to any layer of the system.

Table (3.1) shows the count of number of images for each category of cereals.

4.2 Detecting Text

The first part of the new system for classifying the cereals is detecting which parts of the image contains text.

The neural network designed by Baek *et al.* (2019b), namely Character-Region Awareness for Text Detection, is used to detecting text in images of cereal boxes. This neural network uses VGG as a base network and predicts the region score and affinity score of the characters detected on the image. The region score predicted by the model shows the 2D Gauassian probability distribution of a detected character. The affinity score shows the 2D Gaussian probability distribution of two characters being part of the same word. Figure 4.1 shows the region score and affinity score outputs by CRAFT network.

Using the utilities provided by Baek *et al.* (2019b), I convert the region scores and the affinity scores into polygons. Figure (4.2) shows the detections being converted into polygons. Ideally, each polygon contains one word. The reason for using polygons

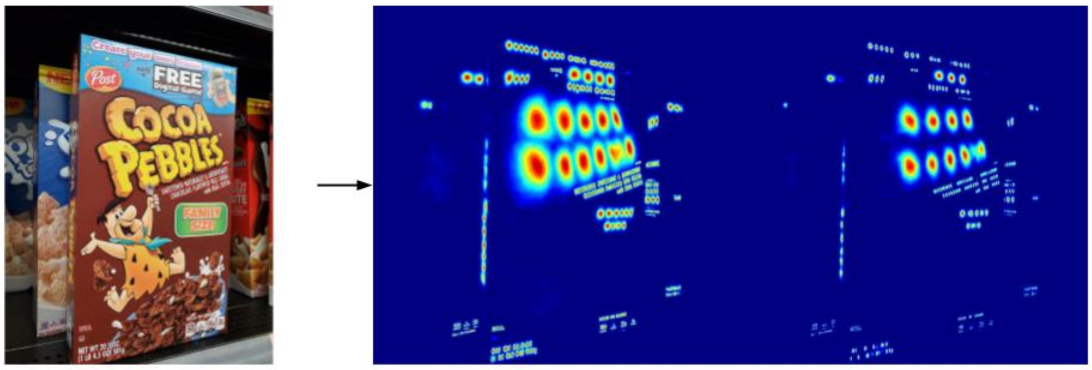


Figure 4.1: CRAFT when given an image as an input, identifies the region and affinity scores of the text found in images.

is so that the image section inside the polygon that image can be cropped, extracted and used as input for text recognition systems.

4.2.1 Successful Detections

Figure (4.3) shows parts of images where the CRAFT model is successful in idenfying the text as part of individual words.

4.2.2 Failing Detections

An failing detection is where the RAFT model does not identify the text in the image or identifies multiple words as a single word in the image. Figure (4.4) shows parts of image where the CRAFT model failed.

The reason we consider multiple words identified by the CRAFT model as failure is because the scene text recognition only identifies one word of all the words inside the polygon. This means we loose information when this happens.

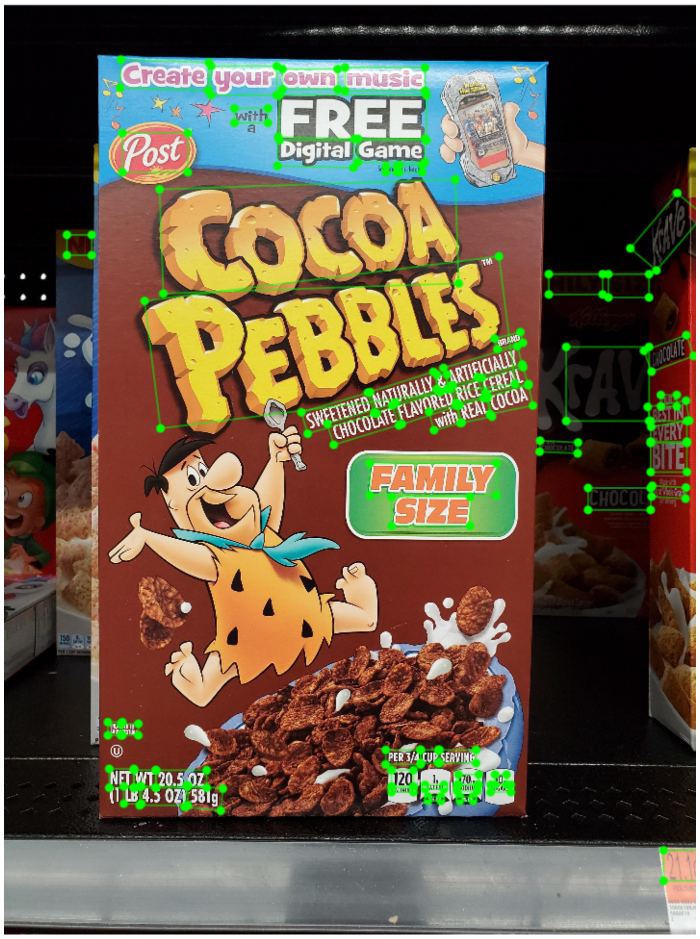


Figure 4.2: Using the Region and Affinity scores, CRAFT provides utilities to convert them into polygons. I used those polygons and converted them into a format that can be visualized using the labelme tool.

4.2.3 Rotating The Polygons

The polygons identifying the detected words, while having 4 vertices, are not always rectangles and also not always completely horizontal on the image. Figure (4.2) shows the words at an angle as detected on the image.

To extract the region of the image inside the polygon, I first compute a rectangle that tightly fits the polygon. This rectangle might not yet be completely horizontal. If I compute a bounding box for the rectangle without rotating the rectangle to be completely horizontal, the resulting bounding box and subsequently the cropped

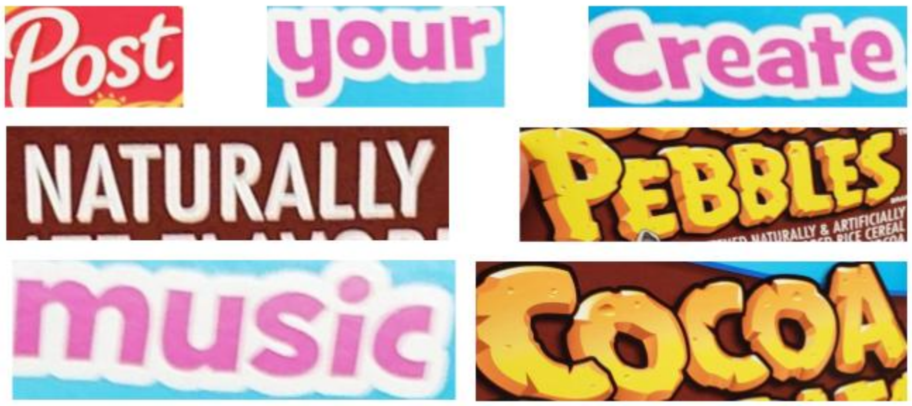


Figure 4.3: Shows parts of image which the CRAFT successfuly identified as single words.

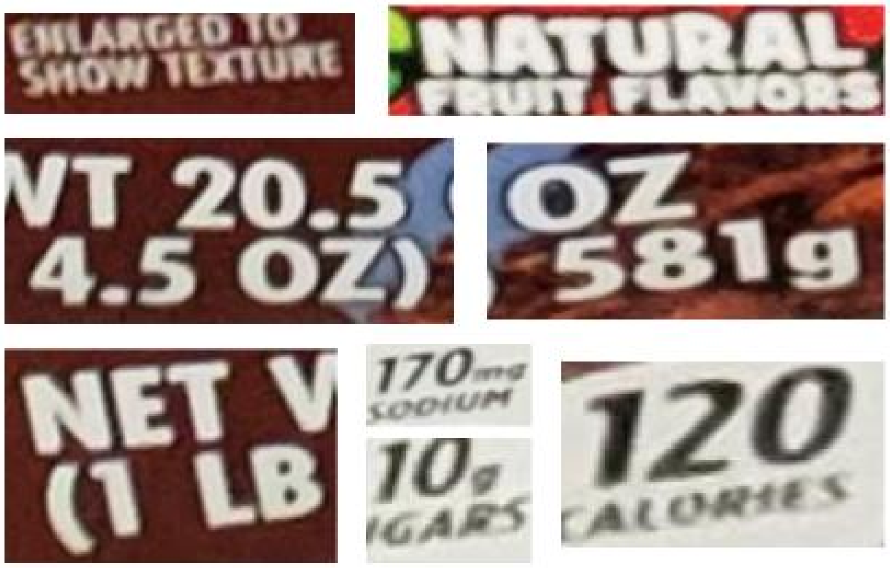


Figure 4.4: While CRAFT is generally very good at identifying text if it exists in the image, the model generally makes a mistake when it identifies multiple words as a single word in the image.

region would contain a lot more area of the image than it is required as shown in Figure (4.5). Looking at the bounding boxes, it becomes clear that we need a better way to crop the polygons which tightly capture the detected text.

Using OpenCV, I calculated the angles that the rectangles need to be rotated for to make the rectangle horizontal. The complete image along with the rectangle is

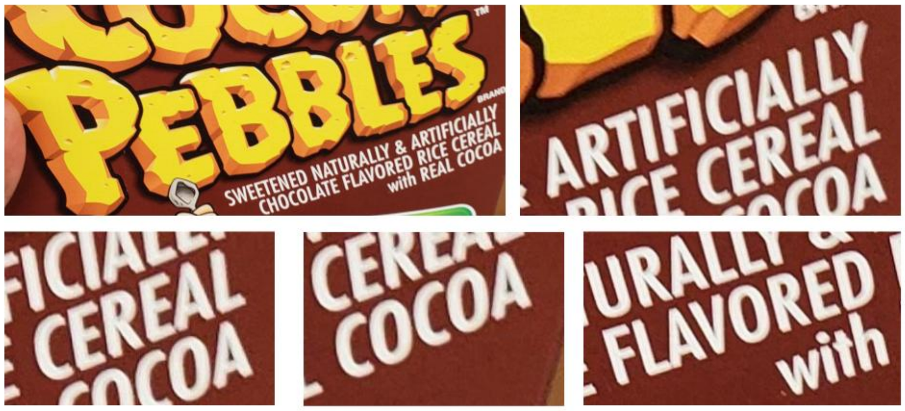


Figure 4.5: When using bounding box to crop the image segments, we extract more information than required. As seen in the above images, since it captures more text than required, it increases the probability of the text recognition model misclassifying the image segment

then rotated by this angle and the region under the rectangle is extracted as shown in Figure.

4.3 Scene Text Recognition

Detecting text in an image brings the solution one step closer to classifying the image. Next, I needed to recognize the text inside the detections in the previous section so as to be able to use the recognized words to classify the cereal boxes.

As per Baek *et al.* (2019a), I used a form of Convolutions Recurrent Neural Network (CRNN) to extract features from the sections of image with text, using a Convolutional Neural Network (CNN) and reconfiguring the extracted features using Recurrent Neural Networks (RNN) accurately predict the sequence of words. Baek *et al.* (2019a) suggest using a four step Scene Text Recognition (STR) framework that transforms the image into text.

As suggested by Baek *et al.* (2019a), to get the best accuracy, I used Residual Networks (ResNet) to extract visual features from the image and Bi-Directional Long

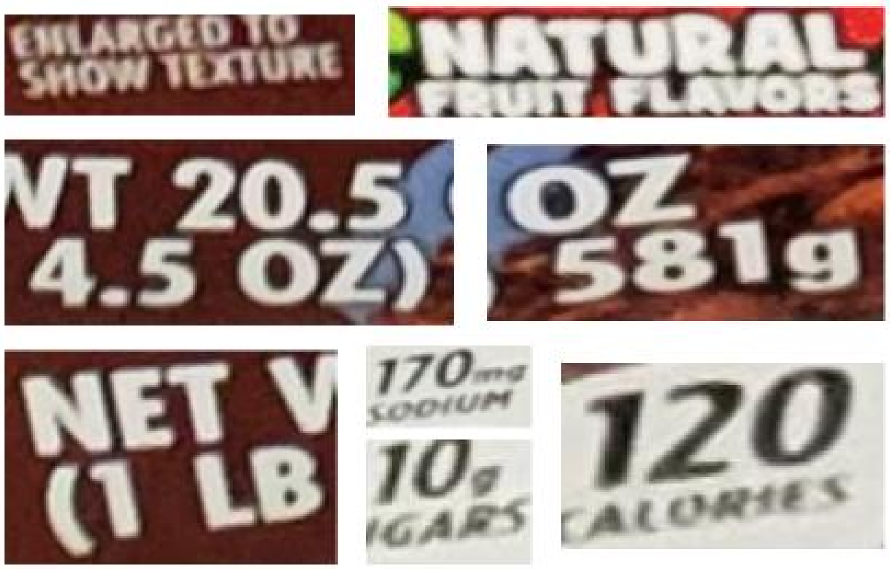


Figure 4.6: When using bounding box to crop the image segments, we extract more information than required. As seen in the above images, since it captures more text than required, it increases the probability of the text recognition model misclassifying the image segment

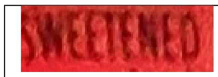
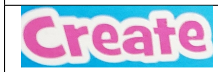
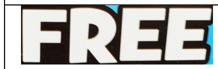
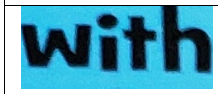
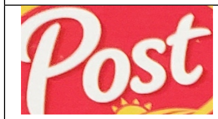
Short Term Memory Network (BiLSTM) to predict the sequence of the recognized characters.

The following subsections demonstrate the situations where the model works and where the model fails.

4.3.1 Successful Inputs

When the input to the scene text recognition model is a word, the model is very accurate in identifying the text in the input. Table (4.1) shows image segments with its predicted output text where the model succeeded.

Table 4.1: Scene Text Recognition Success Examples

Image	Predicted Text
	sweetened
	create
	free
	with
	post

4.3.2 Failing Inputs

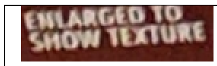
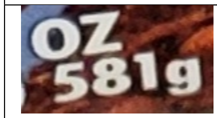
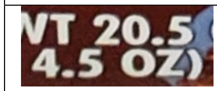
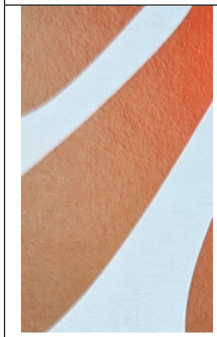
The Scene Text Recognition model fails when the input has multiple words or if the input does not capture the entirety of the word. A few examples are shown in Table (4.2).

4.4 Searching For The Product

At this point in the system, we have the text that the scene text recognition, TPS-ResNet-BiLSTM-Attention, model from the previous section recognized from the text sections detected in the image by CRAFT model from Section (4.2).

The next step in the system is to classify the input image into one of 60 categories using the recognized words. The system ranks the cereal categories for an image based on the number of word found in the data about a cereal category. The detailed explanation of this process is in the following subsection.

Table 4.2: Scene Text Recognition Failing Examples

Image	Predicted Text
	snownsture
	0
	the
	contralized

4.4.1 Ranking Cereal Categories

To classify an image into cereal categories, we assign weights to the cereal categories based on the number of partial word matches in the data of a cereal. Algorithm (4.4.1) details the ranking process:

4.5 Putting It All Together

If we consider the system developed in this chapter a blackbox, it works similar to the cereal classifier developed in Chapter (3). This system still takes an image as an input and returns the category of the image as classified by the system.

However, the internals of this system are much more complicated than the cereal classifier from Chapter (3). Figure (4.7) shows the entire architecture and flow of the implemented system.

Algorithm 1 Classify Image Text into Cereal Box Category

```
$n \leftarrow number\_of\_categories$
$category\_names$
$categories\_info$
$detected\_words \leftarrow scene\_text\_recognition\_output$
$category\_weights \leftarrow [0, 0, 0...0]$
**for** $wordindetected\_words$ **do**
    **for** $i \leftarrow 0, n$ **do**
        $category\_name \leftarrow category_names[i]$
        $category\_info \leftarrow categories_info[i]$
        **if** $wordpartialmatchescategory\_name$ **then**
            $category\_weights[i] \leftarrow category\_weights[i] + 5$
        **end if**
        **if** $wordpartialmatchescategory\_info$ **then**
            $category\_weights[i] \leftarrow category\_weights[i] + 1$
        **end if**
    **end for**
**end for**
$sorted\_indexes \leftarrow -argsort(category\_weights)$
**return** $category\_names[sorted_indexes[0]]$
```

4.5.1 Serving Via Api

In the RESTful API developed during Chapter (3) using FastAPI python framework, I add one more endpoint to serve our new system.

The signature of the new endpoint is given below.

```
PATH: /stcc
HTTP METHOD: POST
BODY TYPE: multipart/form-data
BODY:
- file: string($binary)
RESPONSES:
SUCCESS:
STATUS CODE: 200
STRUCTURE: {"category_name": "string"}
```

4.5.2 Mobile App

In the mobile app developed during Chapter (3) using the Flutter cross platform mobile app framework, I add one more page in the app which takes an image of the product in front of the user, uploads the image to the *POST/stcc/* endpoint of the API.

Figure (4.8) shows the app in action.

4.6 Conclusion

Using Text Detection and Scene Text Recognition to classify images into cereal box categories worked as intended. The advantages of using this technique are:

- Adding a new product to the classification catalog is as simple as adding information about the product.
- Adding a new product does not mean we need to retrain the entire system.
- The entire system is modular. This means, we can replace parts of the entire system would benefit from the updated module without needing to rewrite the entire system.
- Result of the classification can be attributed properly unlike the cereal classifier.

However, we saw that CRAFT model and TPS-ResNet-BiLSTM-Attn model can be improved further. In the next chapter I discuss about creating a dataset that can help improve these models further.

The code for the implemented system can be downloaded from the GitHub Project Page (https://github.com/aksharpatel47/accessible-shopping).

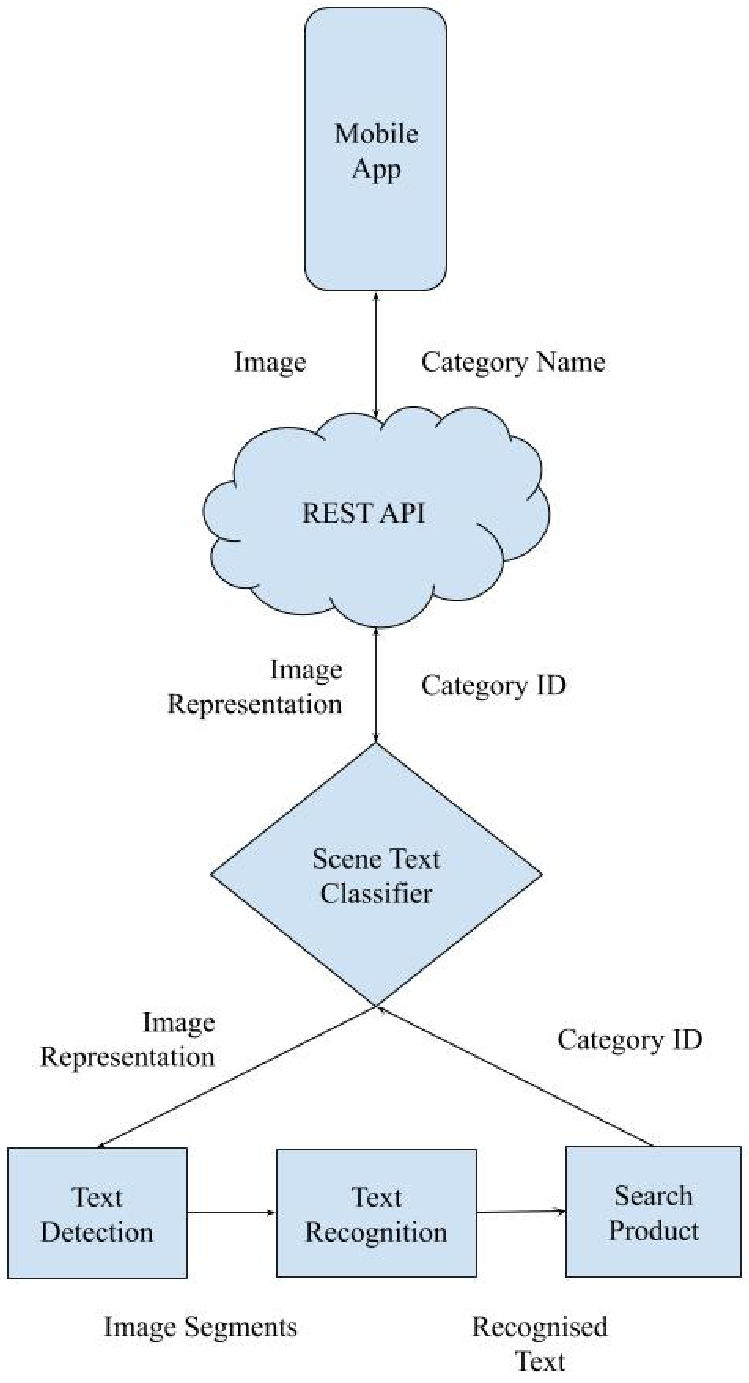


Figure 4.7: The image shows the architecture of the entire scene text classifier system.

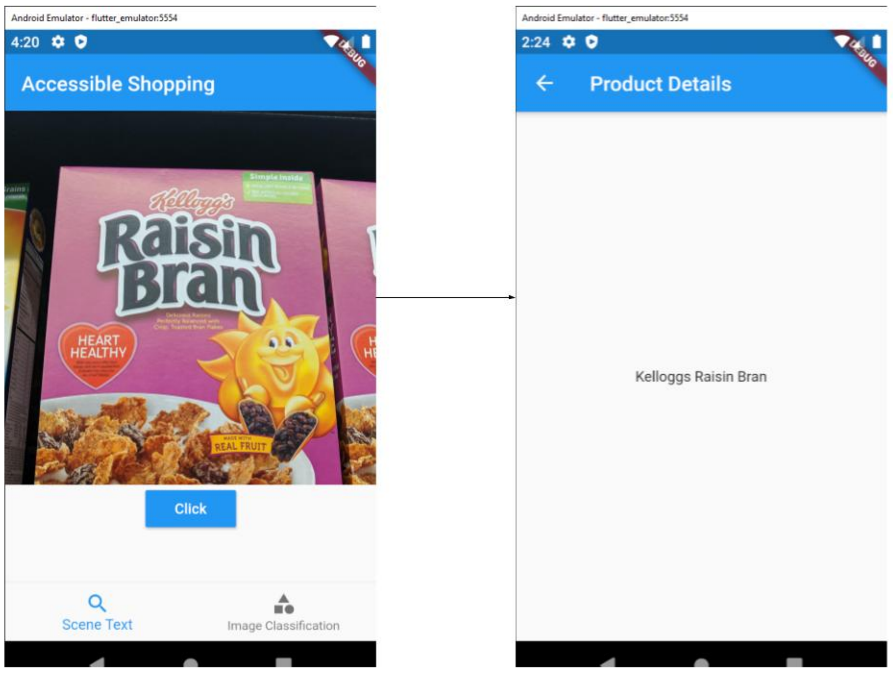


Figure 4.8: Mobile App that takes the image of the product in front of the user, sends it to the restful api and displays the category of the item classified.

Chapter 5

DATASET FOR SCENE TEXT RECOGNITION

After analyzing the system implemented in Chapter (4), we find that if the inputs to the classifying algorithm mentioned in Algorithm (4.4.1 are faulty, the classification fails to return the correct cereal category.

The two biggest pain points of the output of text detection and recognition models were:

1. Polygon of a text detection not tightly fitting the detected word.

2. Inaccurate recognition of text inside an image.

5.1 Improving Text Detection

As mentioned in Baek *et al.* (2019b), CRAFT uses weak supervised learning. It uses word level labels along with word length to learn the region and affinity scores. Hence, to improve CRAFT, I needed to create a dataset with improved word level labels.

To create a dataset that can improve CRAFT or any other text detection or recognition system, I improved the labels of text detection on each and every image in the dataset. The methodology adopted to improve the labels is simple: make the polygon labeling the text detection as tightly fitted as possible. Figure shows the before and after of an image with labels improved.

Category Name	Count of Text Labels
cascadian_farm_organic − cinnamon_crunch	1136
cascadian_farm_organic − fruit_nut_granola	1072
general_mills − apple_cinnamon_cheerios	854
general_mills − blueberry_cheerios	980
general_mills − cheerios_cinnamon_oat_crunch	700
general_mills − cheerios_whole_grain_oats	2464
general_mills − chocolate_cheerios	1359
general_mills − chocolate_lucky_charms	898
general_mills − chocolate_peanut_butter_cheerios	851
general_mills − chocolate_toast_crunch	873
general_mills − cocoa_puffs	461
general_mills − cookie_crisp	781
general_mills − corn_chex	473
general_mills − drumstick_classic_vanilla	543
general_mills − fiber_one	787
general_mills − fiber_one_hone_clusters	532
general_mills − french_toast_crunch_syrup_cinnamon	833
general_mills − golden_grahams	590
general_mills − honey_nut_cheerios	2166
general_mills − honey_nut_cheerios_medley_crunch	1592
general_mills − honey_nut_chex	637
general_mills − lucky_charms	611

general_mills − maple_cheerios	990
general_mills − raisin_nut_bran	1057
general_mills − rice_chex	611
general_mills − very_berry_cheerios	1557
general_mills − wheaties	424
great_value − bran_flakes	765
great_value − raisin_bran	767
kashi − goplay_honey_almond_flax_crunch	705
kashi − gorise_original	506
kashi − organic_blueberry_clulsters	462
kelloggs − all_bran_original	512
kelloggs − chocolate_frosted_flakes	518
kelloggs − corn_flakes	562
kelloggs − froot_loops	414
kelloggs − froot_loops_marshmallows	625
kelloggs − frosted_flakes	876
kelloggs − frosted_mini_wheats_original	508
kelloggs − krave_chocolate	770
kelloggs − pop_tarts_cereal	496
kelloggs − raisin_bran	1267
kelloggs − raisin_bran_crunch	690
kelloggs − rice_krispies	1204
kelloggs − special_k_chocolatey_delight	575
kelloggs − special_k_fruit_yogurt	660

kelloggs − special_k_original	942
kelloggs − special_k_protein	402
kelloggs − special_k_red_berries	1102
kelloggs − special_k_vanilla_almond	595
none	3076
post − cocoa_pebbles	581
post − fruity_pebbles	649
post − great_grains_crunchy_pecan	916
post − honey_bunches_of_oats_almonds	330
post − oreo_os	488
post − sour_patch_kids	492
quaker − simply_granola_oats_honey_almonds	863
quaker − simply_granola_oats_honey_raisins_almonds	901
total − whole_grain_flakes	789

Table 5.1: Number Of Text Detection Labels In Cereal Categories

5.2 Improving Text Recognition

To create a dataset that can improve Scene Text Recognition, I needed to improve the predictions of the TPS-ResNet-BiLSTM-Attn model. This meant going about each and every text detection image and verify the text recognition prediction of that image.

A way to uniquely identify each instance of the label is to keep the information of (i) Category Name (ii) File Name (iii) Coordinates of the polygon fitting the text detection.

I added the above mentioned information into a json string with structure:

```
{
"category": "...",
"file_name": "...",
"points": [...],
"text": "..."
}
```

The text key in the json data has the corrected text output. The above json string is then encoded into a base64 string and this string is set as the name of the file of the text detection cropped image.

5.3 Conclusion

The above two datasets, text detection and recognition dataset, provide accurate labels which can be used to improve scene text recognition neural networks and systems.

Both the datasets can be downloaded from the GitHub Project Page (https://github.com/aksharpatel47/accessible-shopping).

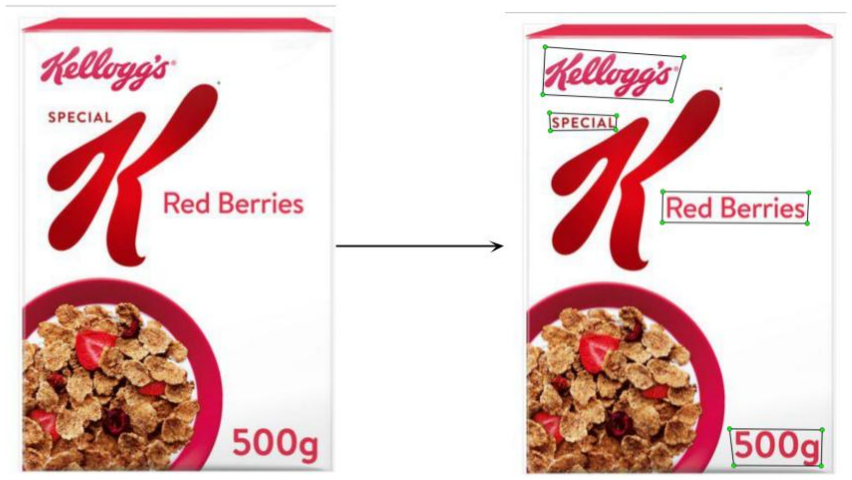


Figure 5.1: As shown in the above image, I added labels where the CRAFT model did not find text so that text detection models can be improved using the new dataset.

Chapter 6

20K PRODUCT DATASET

To accurately measure the performance of the system developed in Chapter (4), I needed to test its performance on a lot more products than 60 categories.

To increase the number of categories I needed the following information of new products:

1. High Quality Images of the Product
2. Name of the Product
3. Name of Brand which manufactures the product
4. About info of the product
5. Health Labels on the product
6. Warning labels on the product
7. Ingredients
8. Specifications of the product
9. Nutrition Information of the Product

To truly test the system on scale, I decided to create a dataset of twenty thousand products with the above described details and images.

In the following sections, I go into how I went about to creating the data.

6.1 Getting The Data

In similar fashion of getting data for the cereals, I used Walmart.com as a source for getting data for the twenty thousand different products. An example of a product page on Walmart.com can be seen in Figure (6.1)

6.1.1 Creating Product Category List

To automate the process of extracting data from Walmart.com, I created a text file containing a list of 46 different categories I wanted to find products of on Walmart.com.

The automated data extraction tool that I wrote, created a managed chrome browser, with 46 different tabs, each tab dedicated to a single category. In each tab, my script opens a search page for the said category.

6.1.2 Extracting Product List

Once all the tabs have opened their respective category search page, the script goes on to create a list of links of the product pages on the Walmart.com. Once the script has collected all the links on one page, it automatically goes to the next page and continues collecting the links. The script collects first 600 products returned for a particular category search.

6.1.3 Extracting Product Information

After completion of the previous step, the number of links I had collected had reached twenty four thousand. Once, I had the links, I wrote an automated script which initializes another managed chrome browser and opens multiple tabs that can be controlled through the script. Each open tab, navigates to the collected product page links and once the page has loaded completely, the script extracts the necessary information, mentioned at the start of this chapter, from the page and saves it as a json file inside the product named folder.

Figure shows product listing page of an example product with important information highlighted.

With all the production information downloaded, I wrote a python script that reads all the json files, puts together a list of images to be downloaded, downloads them and saves them in their respective product folder.

6.2 Improving The Dataset Quality

Since I automated the creation of such a huge dataset, it is bound to contain products with missing information. For the proposed system to work, the more information it has to classify the products the better.

6.2.1 Missing Images

Of all the products whose information I have collected, there are some who don't have any images associated with them. Since the number of such products are small, I decided to ignore such products.

6.2.2 Images Without Text

To validate if the system is able to identify the product just from the downloaded images, the system needs images from which text can be extracted from it. Products where the images contain no text, cannot be tested or validated by the system. Hence, I decided to ignore such products.

6.2.3 Images With Incomplete Information

Just as we cannot validate products having images without text, validating products ,which have text in image but the textual information is insufficient to identify the product, is also difficult. However, since there is some information available, I decided not to remove such products from the datasets, but to move them to a separate folder called $difficult$.

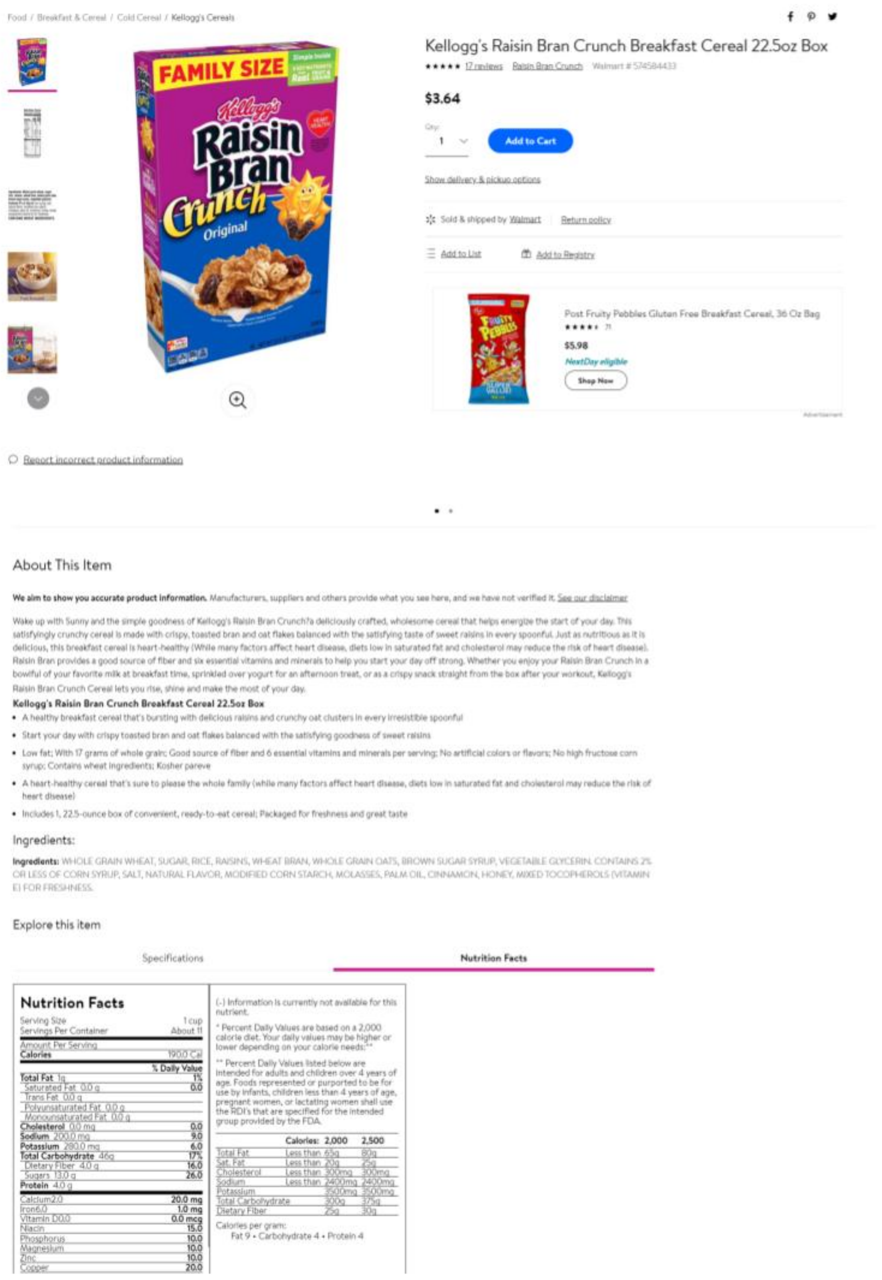
The methodology of selecting products for the difficult folder was as follows:

1. Run Scene Text Recognition on All the Images
2. Find products where the word count inside images <10.
3. Use the Jupyter Notebook to review the product information and mark it as difficult if required.
4. Move all the difficult marked products to a separate difficult folder.

6.3 Conclusion

In this chapter, I discussed about creating a dataset with twenty thousand products containing product images and information. This dataset would then be used to check the performance of a classification/search system developed on the basis of scene text recognition.

The dataset can be downloaded from the GitHub Project Page
(https://github.com/aksharpatel47/accessible-shopping).


Food / Breakfast & Cereal / Cold Cereal / Kellogg's Cereals

Kellogg's Raisin Bran Crunch Breakfast Cereal 22.5oz Box

★★★★★ 17 reviews Raisin Bran Crunch Walmart # 574584433

$3.64

Post Fruity Pebbles Gluten Free Breakfast Cereal, 36 Oz Bag

★★★★★ 71

$5.98

NextDay eligible

About This Item

We aim to show you accurate product information. Manufacturers, suppliers and others provide what you see here, and we have not verified it. See our disclaimer

Wake up with Sunny and the simple goodness of Kellogg's Raisin Bran Crunch?a deliciously crafted, wholesome cereal that helps energize the start of your day. This satisfyingly crunchy cereal is made with crispy, toasted bran and oat flakes balanced with the satisfying taste of sweet raisins in every spoonful. Just as nutritious as it is delicious, this breakfast cereal is heart-healthy (While many factors affect heart disease, diets low in saturated fat and cholesterol may reduce the risk of heart disease). Raisin Bran provides a good source of fiber and six essential vitamins and minerals to help you start your day off strong. Whether you enjoy your Raisin Bran Crunch in a bowlful of your favorite milk at breakfast time, sprinkled over yogurt for an afternoon treat, or as a crispy snack straight from the box after your workout, Kellogg's Raisin Bran Crunch Cereal lets you rise, shine and make the most of your day.

Kellogg's Raisin Bran Crunch Breakfast Cereal 22.5oz Box

- A healthy breakfast cereal that's bursting with delicious raisins and crunchy oat clusters in every irresistible spoonful
- Start your day with crispy toasted bran and oat flakes balanced with the satisfying goodness of sweet raisins
- Low fat; With 17 grams of whole grain; Good source of fiber and 6 essential vitamins and minerals per serving; No artificial colors or flavors; No high fructose corn syrup; Contains wheat ingredients; Kosher pareve
- A heart-healthy cereal that's sure to please the whole family (while many factors affect heart disease, diets low in saturated fat and cholesterol may reduce the risk of heart disease)
- Includes 1, 22.5-ounce box of convenient, ready-to-eat cereal; Packaged for freshness and great taste

Ingredients:

Ingredients: WHOLE GRAIN WHEAT, SUGAR, RICE, RAISINS, WHEAT BRAN, WHOLE GRAIN OATS, BROWN SUGAR SYRUP, VEGETABLE GLYCERIN. CONTAINS 2% OR LESS OF CORN SYRUP, SALT, NATURAL FLAVOR, MODIFIED CORN STARCH, MOLASSES, PALM OIL, CINNAMON, HONEY, MIXED TOCOPHEROLS (VITAMIN E) FOR FRESHNESS.

Explore this item

Specifications | **Nutrition Facts**

Nutrition Facts

Serving Size	1 cup
Servings Per Container	About 11
Amount Per Serving	
Calories	190.0 Cal
	% Daily Value
Total Fat 1g	1%
Saturated Fat 0.0 g	0.0
Trans Fat 0.0 g	
Polyunsaturated Fat 0.0 g	
Monounsaturated Fat 0.0 g	
Cholesterol 0.0 mg	0.0
Sodium 200.0 mg	9.0
Potassium 280.0 mg	6.0
Total Carbohydrate 46g	17%
Dietary Fiber 4.0 g	16.0
Sugars 13.0 g	26.0
Protein 4.0 g	
Calcium2.0	20.0 mg
Iron6.0	1.0 mg
Vitamin D0.0	0.0 mcg
Niacin	15.0
Phosphorus	10.0
Magnesium	10.0
Zinc	10.0
Copper	20.0

(-) Information is currently not available for this nutrient.

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:**

** Percent Daily Values listed below are intended for adults and children over 4 years of age. Foods represented or purported to be for use by infants, children less than 4 years of age, pregnant women, or lactating women shall use the RDI's that are specified for the intended group provided by the FDA.

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat. Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2400mg	2400mg
Potassium		3500mg	3500mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Calories per gram:
Fat 9 • Carbohydrate 4 • Protein 4

Figure 6.1: The image shows listing of Kellogs Raisin Branch Crunch cereals on Walmart.com. The data extractor uses the information available in various sections of the product page to create a product category in the output dataset.

REFERENCES

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems", URL `https://www.tensorflow.org/`, software available from tensorflow.org (2015).

Baek, J., G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh and H. Lee, "What is wrong with scene text recognition model comparisons? dataset and model analysis", (2019a).

Baek, Y., B. Lee, D. Han, S. Yun and H. Lee, "Character region awareness for text detection", (2019b).

Bai, J., F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange", `https://github.com/onnx/onnx` (2019).

Bradski, G., "The OpenCV Library", Dr. Dobb's Journal of Software Tools (2000).

Everingham, M., L. V. Gool, C. K. I. Williams, J. M. Winn and A. Zisserman, "The pascal visual object classes (voc) challenge", International Journal of Computer Vision **88**, 303–338 (2009).

He, K., G. Gkioxari, P. Dollár and R. Girshick, "Mask r-cnn", (2017).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", (2015).

Howard, J. and S. Gugger, "Fastai: A layered api for deep learning", Information **11**, 2, 108, URL `http://dx.doi.org/10.3390/info11020108` (2020).

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

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in "Advances in Neural Information Processing Systems 32", edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, pp. 8024–8035 (Curran Associates, Inc., 2019), URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "Imagenet large scale visual recognition challenge", (2014).

APPENDIX A

USING FLUTTER FOR ACCESSIBLE APP DEVELOPMENT

While implementing the dissertation, when the decision came to select a framework for creating an app that the end users can use to interact with the implemented system, the following criterias were used to evaluate different frameworks and toolkits.

- The framework should follow write once run anywhere philosophy
- It should be sufficiently performant
- It should have a working camera module
- It should have a working filesystem / file picker module
- There should be a possibility of calling C++ code and native system modules from the framework
- Creating a UI using the framework should be a trivial task.

Using the above mentioned criterias, I narrowed my search to Flutter and React Native.

Based on the direction that the Flutter framework is taking, Dartlang being a statically typed language, and the introduction of the Foreign Function Interface (FFI) system in Dartlang which enables calling C++ code from dart, I decided to implement the mobile app using Flutter.

APPENDIX B

IMPROVING INFERENCE TIMES

In the systems developed in the different chapters, the inference is run on the server where the model is served through a RESTful API. However, this appoach has its disadvantages such as:

- It has to upload an entire high quality image taken from the camera of phones. This increases the latency of the system.
- In places where there are no or bad internet connections, the app becomes useless.

These disadvantages can be resolved by implementing the following two methods.

RUNNNING THE MODELS ON THE APP

PyTorch has released libraries that allows PyTorch based models to run on the mobile phones. It also has libraries where it can run the models on the desktops. In addition to PyTorch, there are runtimes such as ONNX, Bai *et al.* (2019), which allow running PyTorch based models after converting them into ONNX format. ONNX also has a web library which enables running models on the web.

This is particularly important as when the user captures an image, instead of waiting for the server to classify the image, accessible information extracted from the text about the product can be given straight to the user while waiting for the server to accurately identify the product.

RUNNING INFERENCE CLOSER TO THE USER

There might be situations where running the model on the device of the user might not be feasable. This might be due to:

- The user device not having enough space to save the model.
- Model utilizing and draining battery of the user device.

In such situations, Retail stores can install a device such as Raspberry Pi along with Intel Compute Stick for inference on the edge of their network, usually at the location of their stores, on which the users can connect and run inference on the captured images.