Machine Learning Final Project Report

Marco Christiani
Virginia Tech
Blacksburg, VA
mchris@vt.edu

Beau Wong
Virginia Tech
Blacksburg, VA
beauwong@vt.edu

Abstract

For our Machine Learning project, we decided to build a learning model to accurately predict the cost of an item from an image of that item. The intended application is to take pictures of certain items in the real world and be able to give a close estimation to what that type of item would cost. This project requires knowledge of web scraping, machine learning techniques, and image recognition algorithms. The project spanned 3 months and was programmed in Python. We implemented a classification method using neural networks and the TensorFlow library. At the end of the project, we were able to produce a model that produced 90% training accuracy and 28% testing accuracy. Unfortunately, we conclude that this model is unreliable for classifying prices of products; however, we are able to make recommendations on how our approach could be improved in the future.

1 Motivation

When visiting a grocery store, shelves are usually littered with price tags. This chaos can make it hard to tell what the actual price of an item is. We wanted to produce an algorithm that could remedy this problem. In addition to this, both team members had an interest in image processing and web scraping, so price prediction from images was a great fit.

2 Project Overview

We were able to break our project down into two main phases: data collection and modeling. In the data collection phase we collected product data, cleaned it, and sourced product images with web scraping techniques. In the modeling phase, we experimented with modeling approaches before settling on a price tier classification approach and evaluated our model's performance using the testing data.

3 Data Collection

In order to train our model, we needed to choose from a number of different online retailers in order to gain pricing information and links to product images. Due to time constraints, we decided against trying to parse multiple datasets from multiple retailers. Thus, we considered datasets specific stores such as Walmart, JCPenney, and Target. However, we concluded that the diversity of Walmart's product lineup allowed us to expose our model to a variety of images while using only one dataset. We sourced a 6 month old dataset from Kaggle [1] for Walmart's inventory, which included about 30,000 different products.

The dataset included many details about each product, so the first step was to determine and extract only the features which were most relevant to our project. After printing out one sample product we concluded we only needed to extract the product's `Unique ID` (UID), `Product URL`, `Name`, `List`

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

`Price`, `Category`, and `GTIN` (Global Trade Identifier Number). In order to gather the corresponding product images, we had to employ some web scraping techniques. To do this, we wrote an Python script that navigates to the product URLs, parses the webpage searching for the correct image HTML tag, and extract the `src` attribute. Next, the script uses the string extracted from the `src` attribute to construct a URL that points to the product image. This image URL along with the aforementioned columns we deemed most relevant are then written to a CSV. It is worth noting that after this process was completed, nearly 13.93% of the product links in the dataset were dead links. We speculate that these dead links are caused by products being discontinued.

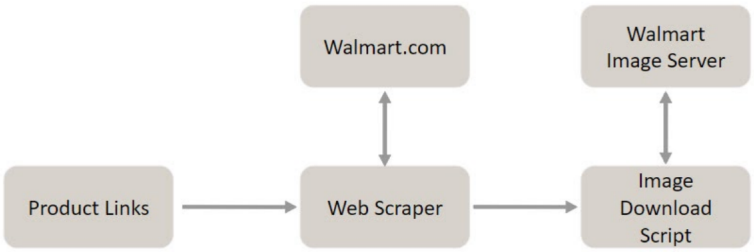

Figure 1: Outline of the data collection process

Next, we loaded the CSV created in the previous step into a new Python script on Google Colab. At this stage, we needed to make an implementation decision between visiting the links with HTTP requests while training and testing our model, or saving the images to Google Drive and reading from disk on Google Colab. We chose the second option since the first option would have likely slowed the training process considerably, and we will likely be training several models. Therefore, we wrote a script which navigates to each image link and downloads the image. The image is then saved using its corresponding `GTIN` as a unique identifier (some sample images can be seen in Figure 2).

Figure 2: Two sample product images scraped and saved by Python scripts

4 Modeling

We decided to use classification for our method of training the model. By dividing the range of products and prices into several, similarly-sized price tiers, we could train a model that would predict which price tier a certain product could go into. We created a distribution of all of the different prices found in the dataset. From there, we were able to assign each item into one of 10 different price tiers. Finally we moved all of the different images we gained from web scraping to the folder of its product determined price tier.

Two methods rose from utilizing price tiers. One would utilize the tiers in a continuous fashion, where one price tier would end where the next price tier would start. This method would encompass all of the data we scraped, but there was potential that items that existed on the edges of each boundary to be misclassified. Another method would be to include gaps in between each price tier. This would

make each price tier much more defined from the others, but at the sacrifice of half of the collected data.

TensorFlow is a library in Python that is capable of building Convolutional Neural Networks to classify images [2]. We decided to use this library when training our data due to its extensive documentation and widespread use. Tensorflow allowed us to augment our images to a form that would be easier for Tensorflow's methods to build a model [3]. Making changes like normalizing the scale of color values and reproducing the image at a fixed resolution expedited the process of modeling.

Our chosen network consisted of three convolutional layers separated by max pooling layers and followed by a dense layer. To address some issues discussed in the next section, we later added a dropout layer in an attempt to mitigate overfitting.

5 Results and Analysis

While our first model was able to achieve approximately 90% accuracy on the training data, the test accuracy lagged behind considerably. As can be seen in the first accuracy plot, the testing accuracy plateaus at around 26% accuracy after 4 epochs. This clearly indicated to us that we had an overfitting problem that needed to be addressed. We attempted to address this issue using two techniques: dropout layers and data augmentation. First, a dropout layer was added after the final max pooling layer that would randomly drop 20% of the outputs. Next, we used data augmentation to apply random transformations such as rotations and flips to the training images. The results from our new modeling approach can be seen in the second accuracy plot.

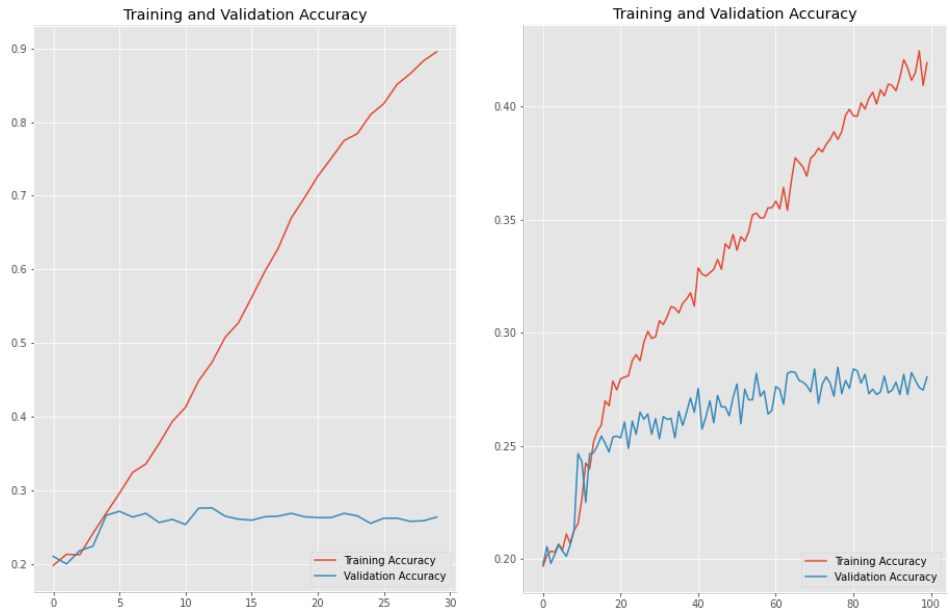

Figure 3: Comparison of accuracy before and after overfit-mitgation strategies were applied.

Key observations to make from these plots is that before the overfit prevention techniques were applied, the testing accuracy stalled at just the fourth epoch and never exceeded 26% accuracy. Contrast this with the second plot, which demonstrates the testing accuracy continuing to increase until the 80th epoch and approaching 29% accuracy. While these results are underwhelming, we are hopeful that our approach could be improved and result in better models in the future.

6 Conclusions

Although the final model was not able to classify images reliably, our project was still able to demonstrate many important machine learning techniques. Furthermore, both of us have gained experience in the planning and process of building a machine learning model.

In hindsight, although the Walmart database was seemingly huge, we believe that the primary reason why our model did not perform to expectation was due to the lack of data. Since the product selection at Walmart is so diverse, there were not enough examples of each type of product for the model to learn effectively. It may have helped to try and incorporate other retailers into our data, as their prices would not have differed too much from Walmart's pricing. It is also possible that the price of items are dependent on features that are not adequately represented in the image. For example, a product whose price is dependent on weight would be quite difficult to learn from an image alone. This proves especially difficult with web scraped images, since each image is scaled to be of relatively the same size for the webpage.

If we were to continue working on this project, we would like to increase the size of the dataset as well as potentially try different characteristics of items to see if it would yield better results. Another approach we have considered, is classifying objects as categories (i.e. grocery items, toiletries, etc) before predicting the price. Finally, we would ideally implement a way for a user to take a picture of an item and see what that item's typical price is. This could also be used to inform the user if the price of a certain product is too high at a certain retailer, encouraging them to purchase it elsewhere.

References

[1] https://www.kaggle.com/promptcloud/walmart-product-data-2019

[2] https://www.tensorflow.org/tutorials/images/classification

[3] https://www.tensorflow.org/tutorials/images/data_augmentation