PROMAP: DATASETS FOR PRODUCT MAPPING IN E-COMMERCE

Omkar Metri, Saptarshi Mondal, Shubham S. Darekar, Sparsh N. Prabhakar
Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA
{metri,saptarshi,sdarekar,sprabhak}@usc.edu
https://github.com/shubhamdarekar/CSCI-567-Product-Mapping-In-Ecomm

1 INTRODUCTION

In the modern age of online shopping, buyers aim to make informed decisions by comparing products across various e-commerce platforms. However, the challenge lies in accurately identifying and aligning similar products across different e-shops for effective price comparisons. The manual process of mapping these products is not only time-consuming but also prone to errors, particularly when dealing with variations in specifications like quantity, weight and brand. Finding a solution to this issue is crucial for providing consumers and market analysts with a reliable and hassle-free way to confidently compare similar products.

Product mapping or product matching (PM), plays a crucial role in e-commerce by facilitating the comparison of identical products across different online platforms. This process involves aligning products from various e-shops, each described by diverse graphical and textual data, making it a valuable tool for general marketplace analysis and price comparison. The absence of a universal product identification system across websites makes product mapping challenging. To address this, models measuring product similarity must be trained using textual and image data. Existing freely available datasets are limited as they often lack comprehensive product information and primarily consist of distant non-matching pairs, simplifying the task of training predictive models.

To bridge this gap, the paper Macková & Pilát (2023) introduced two new datasets: ProMapCz for Czech product mapping and ProMapEn for English product mapping. These datasets were curated by selecting products from diverse categories in one e-shop and manually searching for matching pairs in another. Information such as product name, price, images, long and short descriptions, specifications, and source URLs were scraped and compared manually without relying on any form of identification.

The project aims to predict whether two given products exhibit a matching or non-matching relationship by implementing the methodologies associated with ProMapEn dataset in Macková & Pilát (2023). As a baseline, we conducted scraping and attribute preprocessing from Table 1. Subsequently, we trained multiple machine-learning models with neural network-based models yielding F1 scores of 0.5921 and 0.6593 for train and test respectively. Also, we assessed the model's performance on publicly available datasets and compared the results, demonstrating the ProMapEn dataset is indeed more challenging and provides a good benchmark for product mapping models.

Name		Type
url1	url2	url
name1	name2	text
short_description1	short_description2	text
long_description1	long_description2	text
specification1	specification2	text/json
images1	images2	url
price1	price2	number
match		1/0

Table 1: Dataset Attributes

In the rest of the paper, we discuss current product matching datasets and explore related work within the Product Matching (PM) domain. Section 3 outlines the project's scope and implementation structure. Section 4 details the preprocessing performed on the dataset and the following section contains results obtained by number of different machine learning methods.

2 Related Work

Primpeli et al. (2019) introduced a WDC training dataset for product mapping, which consisted of about 20 million pairs of offers referring to same products. This was extracted from 43 thousand websites having annotated data. The authors verified about 2200 pairs of products manually. Using this, they classified the products into different categories and used the properties like name sku, mpn, etc. to match the products. However, if the categories are not present, it is manual work to annotate the data here. To aid this, Pawłowski (2022) discusses a product classifier using machine learning with the aim of helper system for searching. This is also highly dependent on the search phrases dataset and is influenced by data consistency. AlabdullatifAisha & AloudMonira (2021) also discusses the use of artificial neural networks and deep learning techniques for product matching. This paper has used electronics specific dataset from three different websites and benchmarks it on a classifier based on K-means.

Reddy et al. (2022) introduced the Shopping Queries dataset with ESCI relevance judgments and accompanying Amazon product information in multiple languages, for research in semantic query-product matching tasks. Choi et al. (2020) introduced structured matching module (SMM) that extends Siamese transformer structures with fielded representations and lexical signals, validated through a large-scale empirical evaluation, showcasing promising performance in semantic product search for e-commerce. Kumar & Sarkar (2021) pre-trained RoBERTa with fashion corpus, fine-tuned using triplet loss and achieved 7.8% improvement in MRR, 15.8% in MAP and 8.8% in NDCG on product ranking task.

Zhou et al. (2020) presents a fast localization iris recognition algorithm, which while not being directly applicable, showcased a complex and fast way to identify key characteristics of an image. Dhiman et al. (2023) gives a general idea of how images are processed and what factors play a crucial part in image processing and provided an idea of what features may be important in the image. Dixit & Shirdhonkar (2019) proposes a framework for preprocessing of document image for analysis, which seemed interesting and comparable to the approach used.

Li et al. (2020) demonstrates product mapping across different e-commerce websites using a novel neural matching model. The dataset on which the authors have experimented and trained the model contains real-world data of thousands of product on two e-commerce websites. With the method adopted by authors, the model can take advantage of all the attributes of both product and outperform the then state-of-the-art model significantly. Tracz et al. (2020) uses BERT-based models in a similarity based setup to solve the product matching problem. The authors classified the problem of product matching as zero-shot multi-class classification as the model should be able to handle new instances as there are additions happening everyday in the product catalog on e-commerce websites. Shah et al. (2018) states that matching a seller listed item to an appropriate product has become the most significant and challenging step for e-commerce platforms. In order to solve this problem, the authors have tried two approaches, classification based on shallow neural network and similarity based on deep siamese network. The results of both approaches are encouraging as the models surpass the baseline by 5% in terms of accuracy.

Existing datasets for PM suffer from two shortcomings, i.e., incompleteness, lacking comprehensive product data, and limited utility in real-life scenarios due to the absence of closely resembling non-matching products. Hence, Macková & Pilát (2023) has addressed these issues by constructing dataset, followed by validating its credibility.

3 Scope and structure of Implementation

Image Scraping: The process begins by scraping the images of the products from various e-shops. During scraping, we encountered instances where certain product links were no longer available, resulting in inability to download. Listing 1 depicts usage of `ImageScrape` and the list of important features is provided in Appendix C and E.

```python
class ImageScrape:
    def process():
        # Initialize the input parameters and img_extension=".jpeg"
        # function to download images to local machine using requests

scraper = ImageScrape(img_urls=["http://url1", "http://url2"],
                      dataset_name="ProMapEn",
                      dataset_type="train",
                      ecomm_shop="walmart",
                      destination_dir="images/")
scraper.process()
```

Listing 1: Image Scraping

Image Processing: This involved three steps, i.e., standardizing images through object identification, creating memory-efficient 64-bit image hashes using Perceptual Hashing and computing overall image similarity by comparing hashes between product pairs. Lisitng 2 depicts example usage of `ImageProcessing` and the list of important features is provided in Appendix D and E.

```python
class ImageProcessing:
    def preprocess_and_compute_hash():
        # initilize the parameters, preprocess and generate the hash
        # size=1024, hash_size=8 bytes, threshold=0.9
        return hash_similarity

img = ImageProcessing(source_images="src_path", dest_images="dest_path")
hash_similarity = img.preprocess_and_compute_hash()
```

Listing 2: Image Processing

Text Processing: We have derived numerical vectors from preprocessed textual attributes. Additional vectors were created from keyword extraction and similarity measures to obtain cosine and jaccard similarity between corresponding attributes in product pairs. Listing 3 depicts example usage of `Preprocessing` and comprehensive code is available on github [Appendix F].

```python
class Preprocessing:
    # remove useless characters, separate the units from numbers
    # remove stopwords, lemmatization, extract the brand names
    # create all text column

text_processor = Preprocess(dataframe=promapen)
processed_df["name1"] = text_processor.process_text_column("name1")
```

Listing 3: Data Cleaning

Model Selection and Validation: We have conducted hyperparameter tuning using random and grid search CV for linear regression, support vector machines, decision trees, random forests and neural network models. Listing 4 depicts example usage of custom `RandomAndGridSearchCV` and comprehensive code is available on github.

```python
lg_rand = RandomAndGridSearchCV(X_train, y_train, X_test, y_test,
                                base_model_id=0,
                                classifier=LogisticRegression,
                                params=logistic_param_grid)
lg_rand.estimate_best_params()
```

Listing 4: Model Selection and Validation

4 Methodology

Our primary objective is to derive numerical representation of images using perceptual hashing and text using similarity computations.

4.1 Image Processing

Image preprocessing: We have addressed variations in size, color and centering in product images. To mitigate these issues, images are resized to 1024 × 1024 for memory preservation and speed enhancement. Object detection is employed to identify the largest object within the image. Additional steps include introducing a white border for improved object detection, conversion to grayscale, apply Canny Edge detection and contour finding using OpenCV. Subsequently, the image is cropped to its largest bounding box.

Image hash creation: This process involves splitting the pixel array into blocks and applying kernel operations to capture the main features of the image while preserving crucial information. We used a hash size of 8 with 8 blocks, resulting in 64-bit hashes. This optimization is crucial for balancing information retention and computational efficiency.

Image hash similarity computation: We computed the Hamming distance for each image in the source product set against all images in the target product set. A similarity threshold of 0.9 is applied to filter out images with similarities below 90%. The overall image similarity is computed by summing up these precomputed similarities.

4.2 Text Processing

Data Cleaning: Using columns namely *name*, *short description*, and *long description*, we have created an additional column named *all_texts* by concatenating the mentioned columns. Next, we have removed extraneous characters, separated units and numbers (e.g. 5oz or 15”), employed a word tokenizer to break sentences into words and applied lemmatization to extract root words. To ensure case-insensitive matching, all columns have been converted to lowercase.

Compute Text Similarity: The similarity of attributes in each product pair is determined by converting text into numerical vectors via TF-IDF. Thereafter, cosine similarity between the corresponding vectors is defined as

$$cosine(v1, v2) = \frac{v1 \cdot v2}{|v1||v2|}$$

Keywords Detection and Similarity Computation: We identified keywords like IDs, brands, numbers, and descriptive terms in the text and computed their similarity in product pairs using the Jaccard similarity, expressed as the ratio of common keywords to the total number of keywords for each type.

$$jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Each keyword detection type is described below:

- **Brand Detection:** Utilized the JSON-formatted text within *specification* attribute to create *brand* vocabulary, followed by detecting brands across various attributes.
- **ID Detection:** Selected unique words that are more than 5 characters in length which are not present in the English vocab of Paracrawl dataset [Bañón et al. (2020)].
- **Numbers Detection:** Utilized regular expressions to identify numbers in the data. If no unit is identified alongside the number, it is considered a free number (e.g. model numbers).
- **Units Detection:** Identified units followed by numbers, thereby providing valuable information like dimensions, capacity, and quantity.
- **Descriptive Words:** Selected top k words occurring in a maximum of p percent of the documents with both k and p set to 50.
- **Words:** Ratio of same words to all words in the same attribute of the product listings.

Keywords List Comparison: Using identified keywords such as IDs, brands, numbers, and units, we constructed lists for each product and compared the ratio of matching values in those lists between product pair.

Specification processing: Leveraged the JSON based *specification* attribute to extract similarity between keys and key-value pairs for direct attribute comparison in product mapping.

To summarize, we created 34-feature vector from attribute distances (see Appendix B) and paired it with labels to train machine learning models for predicting match/non-match between product pairs across the two e-shops. Details regarding model selection and evaluation are described in the next section.

5 RESULTS AND DISCUSSION

Training and Finetuning: Logistic Regression (LR), Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), and Neural Network (NN) classifiers were employed to predict whether pairs are matching or non-matching. The dataset was divided into training and test data in an 80:20 ratio, comprising 1,244 vectors for training and 311 for testing. Grid search and random search were employed to identify optimal hyperparameters for each model, utilizing 20 percent of the training set as validation set. Possible parameter values are summarized in Table 2. We determined the optimal hyperparameters by maximizing the F1 score. Subsequently, the models were trained using these best hyperparameters on the combined training and validation sets.

Table 2: Parameter settings for grid search CV

Models	Parameters	Possible Values	Best Setting
Logistic Regression	penalty	l1, l2, elasticnet, none	none
	solver	lbfgs, newton-cg, liblinear sag, saga	lbfgs
	max_iter	10, 20, 50, 100, 200, 500	10
SVM	kernel	linear, poly, rbf, sigmoid	rbf
	degree	2, 3, 4, 5	2
	max_iter	10, 20, 50, 100, 200, 500	200
Decision Tree	criterion	gini, entropy	entropy
	max_depth	5, 10, 15, 20	20
	min_samples_split	2, 5, 10, 15, 20	2
Random Forest	n_estimators	50, 100, 200, 500	50
	criterion	gini, entropy	gini
	max_depth	5, 10, 20, 50	20
	min_samples_split	2, 5, 10, 20	10
Neural Networks	hidden_layer_sizes	(10, 10), (50, 50), (10, 50), (10, 10, 10) (50, 50, 50), (50, 10, 50), (10, 50, 10)	(10,50)
	activation	relu, logistic, tanh	tanh
	solver	adam, sgd, lbfgs	adam
	learning_rate	constant, invscaling, adaptive	constant
	learning_rate_init	0.01, 0.001, 0.0001	0.01
	max_iter	50, 100, 500	100

The dataset contains 1049 non matching pairs and 509 matching pairs. Due to class imbalance, choosing F1 score over accuracy as metric is preferable in such situations. The F1 score considers both precision and recall, providing a more balanced assessment, especially when one class greatly outnumbers the other. The F1 score combines precision (the accuracy of positive predictions) and recall (the fraction of actual positives predicted correctly) into a single metric, striking a balance between false positives and false negatives.

Evaluation: To test the transfer learning capabilities of the model trained on ProMapEn, we picked the best model and tested the transfer learning capabilities on three different datasets: ProMapCz, Amazon Walmart [AmW], and Amazon Google [Köpcke et al. (2010)] with 299, 629 and 647 test vectors respectively. As each of the datasets contain different attributes, we filled the values of the missing attributes with zeroes and discarded the additional superfluous attributes.

Results: The neural network-based models yielded the most favorable results with an F1 score of 0.5921 closely followed by logistic regression and random forests as outlined in Table 3 and Appendix G. It is evident that neural networks tend to exhibit a more balanced precision and recall,

Table 3: Training scores with the best parameters from random and grid searches

Model	F1 Score	Precision	Recall
LG-RAND	0.5576	0.6695	0.4878
LG-GRID	0.5624	0.6805	0.4854
SVM-RAND	0.5272	0.6880	0.4290
SVM-GRID	0.5436	0.5787	0.5246
DT-RAND	0.4993	0.5012	0.5050
DT-GRID	0.5076	0.4955	0.5247
RF-RAND	0.5511	0.6640	0.4832
RF-GRID	0.5536	0.6836	0.4782
NN-RAND	0.5609	0.6308	0.5124
NN-GRID	0.5921	0.6182	0.5735

Table 4: Comparison of the best neural network model, trained on ProMapEn dataset and evaluated on all datasets to show the difficulty of ProMapEn dataset

	F1 Score	Precision	Recall
ProMapEn	0.6593	0.7407	0.5941
ProMapCz	0.5836	0.4717	0.7653
Amazon Walmart	0.5350	0.9767	0.3684
Amazon Google	0.5604	1.0000	0.3893

whereas random forests exhibit larger discrepancies between these values. Additionally, there does not seem to be significant difference between the results found by the grid search and random search.

The transfer learning and test scores on original dataset are shown in Table 4. Nearly identical performance on the training set and test set is a good outcome. In other words, the model is effectively performing the defined task. The slightly lower metrics confirm that ProMapEn dataset is more challenging. There could be several reasons for that, such as fewer IDs in the descriptions of the products, different selection of products or the need for further finetuning of preprocessing and feature extraction techniques.

6 Conclusion and Future Work

We have replicated end-to-end implementation for product mapping by automating extraction of various product attributes from English e-shops. Our approach involved text and image preprocessing, training multiple machine learning models to predict matching and non-matching pairs across e-shops with neural networks outperforming other models in terms of F1 score. We also tested transfer learning capabilities of the model by using other publicly available datasets. These datasets and models serve as a baseline for future enhancements in the product mapping task.

Exploring the factors contributing to lower metrics such as potentially smaller dataset, fewer unique identifiers in product descriptions on English e-shops, variations in product and category selection, or the potential need for additional fine-tuning in preprocessing, presents scope for future work. A more in-depth analysis of these differences and the investigation of alternative machine learning methods, such as deep neural networks, will be the focus for future research.

7 Acknowledgments

Prof Dani Yogatama, TAs, CPs for guidance and feedback
Kateřina Macková, Martin Pilát for wonderful paper and dataset

References

Amazon walmart dataset. `https://hpi.de/naumann/projects/repeatability/datasets/amazon-walmart-dataset.html`. Accessed: 2023-12-08.

AlabdullatifAisha and AloudMonira. Araprodmatch: A machine learning approach for product matching in e-commerce. *International Journal of Computer Science and Network Security*, 21 (4):214–222, 4 2021.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. ParaCrawl: Web-scale acquisition of parallel corpora. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4555–4567, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.417. URL `https://aclanthology.org/2020.acl-main.417`.

Jason Ingyu Choi, Surya Kallumadi, Bhaskar Mitra, Eugene Agichtein, and Faizan Javed. Semantic product search for matching structured product catalogs in e-commerce, 2020.

Poonam Dhiman, Amandeep Kaur, V. R. Balasaraswathi, Yonis Gulzar, Ali A. Alwan, and Yasir Hamid. Image acquisition, preprocessing and classification of citrus fruit diseases: A systematic literature review. *Sustainability*, 15(12), 2023. ISSN 2071-1050. doi: 10.3390/su15129643. URL `https://www.mdpi.com/2071-1050/15/12/9643`.

Umesh D Dixit and MS Shirdhonkar. Preprocessing framework for document image analysis. *International Journal of Advanced Networking and Applications*, 10(4):3911–3918, 2019.

Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1–2):484–493, sep 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920904. URL `https://doi.org/10.14778/1920841.1920904`.

Lakshya Kumar and Sagnik Sarkar. Neural search: Learning query and product representations in fashion e-commerce, 2021.

Juan Li, Zhicheng Dou, Yutao Zhu, Xiaochen Zuo, and Ji-Rong Wen. Deep cross-platform product matching in e-commerce. *Information Retrieval Journal*, 23:136–158, 2020.

Kateřina Macková and Martin Pilát. Promap: Datasets for product mapping in e-commerce, 2023.

Mieczysław Pawłowski. Machine learning based product classification for ecommerce*. *Journal of Computer Information Systems*, 62(4):730–739, 2022. doi: 10.1080/08874417.2021.1910880. URL `https://doi.org/10.1080/08874417.2021.1910880`.

Anna Primpeli, Ralph Peeters, and Christian Bizer. The wdc training dataset and gold standard for large-scale product matching. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, pp. 381–386, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366755. doi: 10.1145/3308560.3316609. URL `https://doi.org/10.1145/3308560.3316609`.

Chandan K. Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. Shopping queries dataset: A large-scale esci benchmark for improving product search, 2022.

Kashif Shah, Selcuk Kopru, and Jean-David Ruvini. Neural network based extreme classification and similarity models for product matching. In Srinivas Bangalore, Jennifer Chu-Carroll, and Yunyao Li (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pp. 8–15, New Orleans - Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-3002. URL `https://aclanthology.org/N18-3002`.

Janusz Tracz, Piotr Iwo Wójcik, Kalina Jasinska-Kobus, Riccardo Belluzzo, Robert Mroczkowski, and Ireneusz Gawlik. BERT-based similarity learning for product matching. In Huasha Zhao, Parikshit Sondhi, Nguyen Bach, Sanjika Hewavitharana, Yifan He, Luo Si, and Heng Ji (eds.), *Proceedings of Workshop on Natural Language Processing in E-Commerce*, pp. 66–75, Barcelona, Spain, December 2020. Association for Computational Linguistics. URL `https://aclanthology.org/2020.ecomnlp-1.7`.

Weibin Zhou, Xiaotong Ma, and Yong Zhang. Research on image preprocessing algorithm and deep learning of iris recognition. *Journal of Physics: Conference Series*, 1621:012008, 08 2020. doi: 10.1088/1742-6596/1621/1/012008.

A Appendix

A.1 Author Contributions

Omkar Metri: Performed image scraping, dataset organization and background research through the analysis of three papers. In the image processing phase, implemented perceptual hashing and computed image similarity using Hamming distance. In text processing, extracted features associated with *keyword list comparison* and *specification processing*. In model building and fine-tuning, found optimal hyperparameters through GridSearchCV and RandomizedSearchCV by deploying Random Forest and Neural Networks. In evaluation, tested on the original dataset and transfer learning capabilities. Documented a detailed README file with execution steps, created poster and report writing.

Saptarshi Mondal: Worked on image processing to detect objects from images, model building (Logistic Regression, Decision Tree and SVM), poster making (general aesthetic of poster and the methodology part of the poster), wrote section of the related work part of the report and proofread the report.

Shubham S. Darekar: Worked on setting up the git repository, data cleaning and feature engineering by text processing. Reviewed and updated the code for *Brand Detection*, *ID detection*, *Descriptive Words*, *Words* keywords detection and *Keywords list comparison*. Updated the code parameters in text processing for improving the model accuracy.

Sparsh N. Prabhakar: Literature survey of the domain by analysing previous papers on Product Mapping and Product Matching. Worked on feature-engineering for the text processing part. Generated keywords list and built features using *Brand Detection*, *Numbers Detection*, *Units Detection* and *Words*. Also, finetuned parameters to increase model accuracy and documentation of the report.

B Appendix

Table 5: Precomputed Similarity Features used for model training

name_brand	all_texts_descriptives	specification_text_numbers
name_cos	all_texts_id	specification_text_units
name_descriptives	all_texts_numbers	short_description_brand
name_id	all_texts_units	short_description_cos
name_numbers	all_texts_words	short_description_descriptives
name_units	all_units_list	short_description_id
name_words	long_description_cos	short_description_numbers
all_brands_list	long_description_descriptives	short_description_units
all_ids_list	long_description_numbers	short_description_words
all_numbers_list	long_description_units	hash_similarity
all_texts_brand	specification_key	
all_texts_cos	specification_key_value	

C Appendix

```python
class ImageScrape:
    def __init__(self, img_urls, dataset_name, dataset_type, ecomm_shop, destination_dir, img_extension=".jpeg"):
        self.img_urls = img_urls
        self.dataset_name = dataset_name
        self.dataset_type = dataset_type
        self.ecomm_shop = ecomm_shop
        self.img_extension = img_extension
        self.destination_dir = destination_dir

    @property
    def absolute_path(self):
        return f"{self.destination_dir}/{self.dataset_name}/{self.dataset_type}/{self.ecomm_shop}/"

    def download_images(self, img_urls, row_number):
        print(f"------ Download images from row: {row_number} ------")

        path = f"{self.absolute_path}/{row_number}"
        os.makedirs(path, exist_ok=True)

        for img_ind, image in enumerate(img_urls):
            img = requests.get(image, stream=True)
            if img.status_code == 200:
                with open(f"{path}/{img_ind}{self.img_extension}", "wb") as fp:
                    shutil.copyfileobj(img.raw, fp)
            else:
                print(f"Image download failed: {row_number}: {img_ind}")

    def process(self):
        """
        Process the URLs for every dataset, type and ecomm_site
        """
        for index, url_list in enumerate(self.img_urls):
            self.download_images(url_list, index)
```

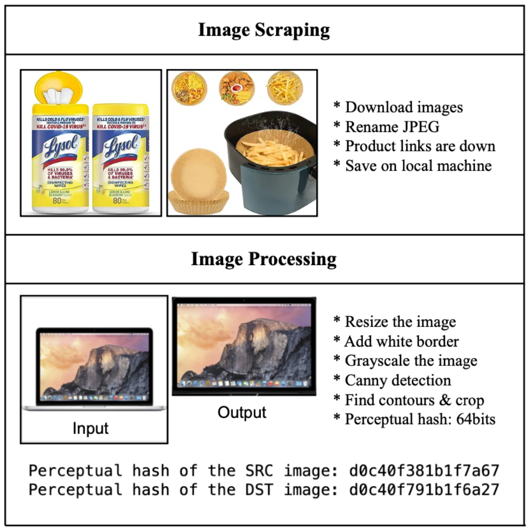
Listing 5: Important Functions and Methods of Image Scraping

D Appendix

```python
class ImageProcessing:
    """Generate hash similarity"""
    def __init__(self, source_images, dest_images, size=1024, hash_size=8, threshold=0.9):
        self.source_images = source_images
        self.dest_images = dest_images
        self.size = size
        self.hash_size = hash_size
        self.threshold = threshold
        self.source_hashes = []
        self.dest_hashes = []

    def find_largest_contour_image(self, image_path):
        # Read, resize to 1024*1024, add the white border and grayscale
        img = cv2.imread(image_path)
        img = self.resize_image(img)
        img_with_border = self.add_border(img)
        gray_img = cv2.cvtColor(img_with_border, cv2.COLOR_BGR2GRAY)

        # Create a black and white mask of the object using canny edge detection
        edges = cv2.Canny(gray_img, 30, 100)
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Find the largest contour
        largest_contour = max(contours, key=cv2.contourArea)

        # Create a mask for the largest object and apply the mask to the original image
        mask = np.zeros_like(gray_img)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=cv2.FILLED)
        masked_img = cv2.bitwise_and(img_with_border, img_with_border, mask=mask)

        # Crop the image to the bounding box of the largest object
        x, y, w, h = cv2.boundingRect(largest_contour)
        cropped_img = masked_img[y:y+h, x:x+w]

        return cropped_img

    def compute_hash(self, image):
        pil_image = Image.fromarray(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

        return imagehash.phash(pil_image, hash_size=self.hash_size)

    @staticmethod
    def hamming_distance(hash1, hash2):
        return bin(int(str(hash1), 16) ^ int(str(hash2), 16)).count('1')

    def compute_image_similarity(self, source_hashes=None, dest_hashes=None):
        overall_similarity = 0

        for hash1 in source_hashes:
            most_similar_similarity = 0

            for hash2 in dest_hashes:
                distance = ImageProcessing.hamming_distance(hash1, hash2)
                similarity = 1 - (distance / 64.0)

                if similarity > most_similar_similarity:
                    most_similar_similarity = similarity

            if most_similar_similarity > self.threshold:
                overall_similarity += most_similar_similarity

        return overall_similarity

    def preprocess_and_compute_hash(self):
        for img in self.source_images:
            try:
                contour_img = self.find_largest_contour_image(img)
                hash_value = self.compute_hash(contour_img)
                self.source_hashes.append(str(hash_value))
            except:
                continue

        for img in self.dest_images:
            try:
                contour_img = self.find_largest_contour_image(img)
                hash_value = self.compute_hash(contour_img)
                self.dest_hashes.append(str(hash_value))
            except:
                continue

        return self.compute_image_similarity(self.source_hashes, self.dest_hashes)
```

Listing 6: Important Functions and Methods of Image Processing

E Appendix



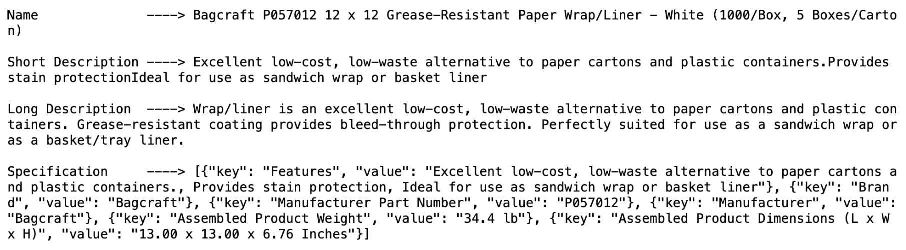
* Download images
* Rename JPEG
* Product links are down
* Save on local machine

* Resize the image
* Add white border
* Grayscale the image
* Canny detection
* Find contours & crop
* Perceptual hash: 64bits

```
Perceptual hash of the SRC image: d0c40f381b1f7a67
Perceptual hash of the DST image: d0c40f791b1f6a27
```

Figure 1: Image Scraping and Processing Example

F Appendix



```
Name              ----> Bagcraft P057012 12 x 12 Grease-Resistant Paper Wrap/Liner - White (1000/Box, 5 Boxes/Carto
n)

Short Description ----> Excellent low-cost, low-waste alternative to paper cartons and plastic containers.Provides
stain protectionIdeal for use as sandwich wrap or basket liner

Long Description  ----> Wrap/liner is an excellent low-cost, low-waste alternative to paper cartons and plastic con
tainers. Grease-resistant coating provides bleed-through protection. Perfectly suited for use as a sandwich wrap or
as a basket/tray liner.

Specification     ----> [{"key": "Features", "value": "Excellent low-cost, low-waste alternative to paper cartons a
nd plastic containers., Provides stain protection, Ideal for use as sandwich wrap or basket liner"}, {"key": "Bran
d", "value": "Bagcraft"}, {"key": "Manufacturer Part Number", "value": "P057012"}, {"key": "Manufacturer", "value":
"Bagcraft"}, {"key": "Assembled Product Weight", "value": "34.4 lb"}, {"key": "Assembled Product Dimensions (L x W
x H)", "value": "13.00 x 13.00 x 6.76 Inches"}]
```

Figure 2: Textual Data Example

G Appendix

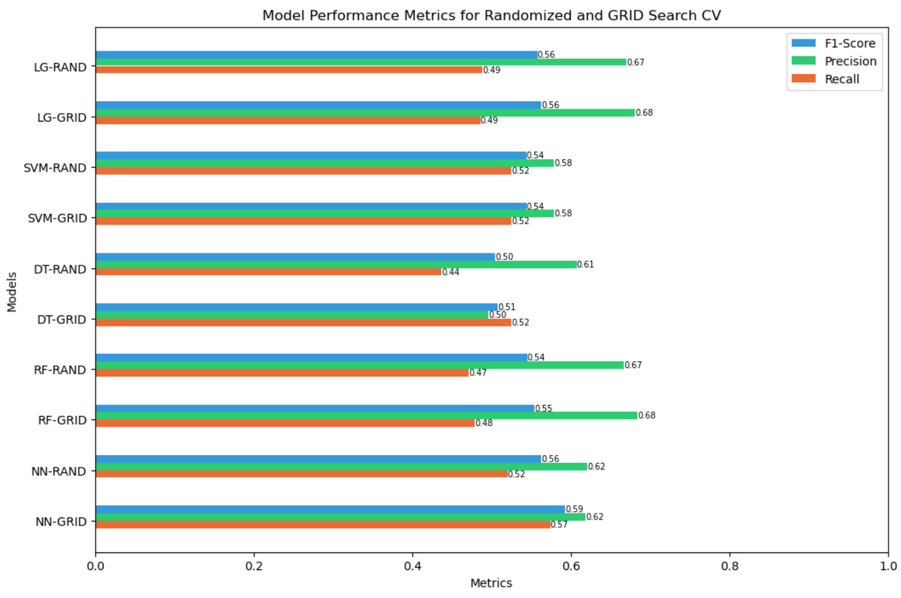


Figure 3: Comparison of Training Accuracy

H Appendix

The code was run on below machines

- i7-8750H with 32 GBs of RAM with NVidia 1050Ti Graphics processor
- MacBook Pro M2 8GB of RAM