Image Tagging for Search and Recommendation

Ashish Sahay, Ashwin Bagree, Kevin Susanto, Noble Padiyara, Utkarsh Chaturvedi

Abstract

Our paper focuses on improving the accuracy of the tag listing of a product by identifying tags using image recognition. This can enable C2C resale websites to give more accurate listings on their portals and reduce fraudulent use of text spamming.

1. Introduction

2nd hand markets and products are having a resurgence. But contrary to the era of hand-me downs, these products are creating a new industry. Also known as the second-hand economy, the resale fashion industry in the US has become a US $24 billion mammoth, challenging the current leaders of fast-fashion industry [CNBC, Forbes]. In India, the year 2018-19 saw the sale of 4 million second-hand cars compared to 3.6 million units of new car sales [LiveMint]. This boom across the world has both, been facilitated by and created the rise of another industry - online marketplaces.

The online marketplace industry, which is the focus in this paper has already had quite a success in the consumer-to-consumer (C2C) market [Economic Times]. Some successful examples are:

- Chinese website 'Pinduoduo' that reached gross merchandise value of above USD 14.8 Billion.
- Homegrown 'Carousell' has sold more than 23 million items in Singapore in which users have created 57 million plus listings in the website in 2016.
- Indian website 'OLX' which had more than 11 million-page views and 8.5 million transactions per month in 2014.

The rise of these e-commerce websites again depends on a host of factors among which are: trust in the product and the seller, a good search engine to get what the buyers want and a good recommendation engine to drive more sales and facilitate faster movement of seller goods. It is important to notice that apart from the first problem, the latter 2 problems are a unique challenge to new websites when compared to large marketplaces such as Amazon.

Since these algorithms are not as robust, the sellers of those products, may commit certain behavior to increase the listing's exposure and gain an unfair advantage over other sellers by keyword spam (inserting irrelevant keywords that do not accurately describe the products they are selling) or outright falsification. This would tamper with accuracy of the system and as a result, customer might shift to alternative sites for shopping.

1.1 Problem Statement

Our business problem tries to tackle the shortcomings mentioned above by looking into the feasibility of using item annotation through image recognition instead of just text. As an added advantage, image recognition system can capture elements that are not immediately noticeable by humans. This will then increase effectiveness of marketing metrics such as visitor-to-sale ratio and customer lifetime value (CLV). We believe that a comprehensive automated image tagging mechanism for these platforms will:

- Build trust as cases of counterfeit, contra-band and fraudulent products are flagged.
- Allow for more level playing field across seller and buyers where the seller cannot overwhelm the system using spam words.

1.2 Anticipated Challenges

- **Business Challenge:** The traditional recommender system, which works on the basis of similarity of item of interest (content based filtering), user's behavioral information (collaborative filtering), user's profile (demographic filtering) and the combination of all (hybrid filtering) that is identified widely by item's description (text). A tag-based recommendation system might require certain changes which could pose some unique problems.

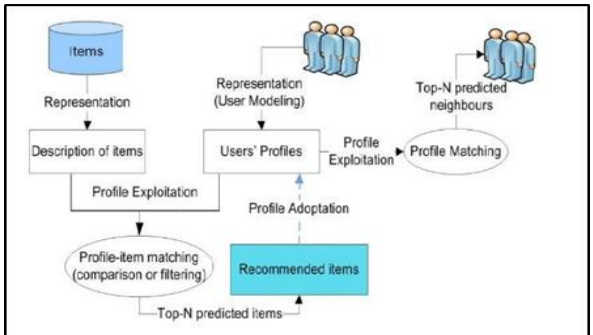

Figure 1. General Concept of Recommender Systems [8].

Fig 1

- **Technical Challenge**: The images uploaded in a C2C portal are of a wide dimension of variety and quality. Grainy low-resolution pictures to high end DSLR ones, all these images will need to be tagged, which would require an extremely robust system of recognition.

2. Data

2.1 Dataset

For the purpose of automatic tagging of images uploaded by users, we wanted a dataset containing pre tagged "dirty" images of a variety of products that are generally sold in second-hand e-commerce websites. We have chosen Amazon review data available on GitHub for the same.

The Amazon Review data was scraped from the Amazon website and contains 233.1 million reviews of 15.5 million products spanning over 24 main categories with reviews in the span of 1996-2018. These reviews often contain images uploaded by reviewers which fits our use perfectly. We have extracted the image URL's of the user submitted images, metadata (clean) product images, category tags and product ID which are useful for image tagging and cosine similarity matrix.

Since electronics is one of the most popular resale categories apart from fashion, a subset of data containing only '**Electronic'** products was chosen to build a Proof of Concept which can then be implemented to the entire dataset.

Amazon Fashion	5-core (3,176 reviews)	ratings only (883,636 ratings)
All Beauty	5-core (5,269 reviews)	ratings only (371,345 ratings)
Appliances	5-core (2,277 reviews)	ratings only (602,777 ratings)
Arts, Crafts and Sewing	5-core (494,485 reviews)	ratings only (2,875,917 ratings)
Automotive	5-core (1,711,519 reviews)	ratings only (7,990,166 ratings)
Books	5-core (27,164,983 reviews)	ratings only (51,311,621 ratings)
CDs and Vinyl	5-core (1,443,755 reviews)	ratings only (4,543,369 ratings)
Cell Phones and Accessories	5-core (1,128,437 reviews)	ratings only (10,063,255 ratings)
Clothing, Shoes and Jewelry	5-core (11,285,464 reviews)	ratings only (32,292,099 ratings)
Digital Music	5-core (169,781 reviews)	ratings only (1,584,082 ratings)
Electronics	5-core (6,739,590 reviews)	ratings only (20,994,353 ratings)
Gift Cards	5-core (2,972 reviews)	ratings only (147,194 ratings)
Grocery and Gourmet Food	5-core (1,143,860 reviews)	ratings only (5,074,160 ratings)
Home and Kitchen	5-core (6,898,955 reviews)	ratings only (21,928,568 ratings)
Industrial and Scientific	5-core (77,071 reviews)	ratings only (1,758,333 ratings)
Kindle Store	5-core (2,222,983 reviews)	ratings only (5,722,988 ratings)
Luxury Beauty	5-core (34,278 reviews)	ratings only (574,628 ratings)
Magazine Subscriptions	5-core (2,375 reviews)	ratings only (89,689 ratings)
Movies and TV	5-core (3,410,019 reviews)	ratings only (8,765,568 ratings)
Musical Instruments	5-core (231,392 reviews)	ratings only (1,512,530 ratings)
Office Products	5-core (800,357 reviews)	ratings only (5,581,313 ratings)
Patio, Lawn and Garden	5-core (798,415 reviews)	ratings only (5,236,058 ratings)
Pet Supplies	5-core (2,098,325 reviews)	ratings only (6,542,483 ratings)
Prime Pantry	5-core (137,788 reviews)	ratings only (471,614 ratings)
Software	5-core (12,805 reviews)	ratings only (459,436 ratings)
Sports and Outdoors	5-core (2,839,940 reviews)	ratings only (12,980,837 ratings)
Tools and Home Improvement	5-core (2,070,831 reviews)	ratings only (9,015,203 ratings)
Toys and Games	5-core (1,828,971 reviews)	ratings only (8,201,231 ratings)
Video Games	5-core (497,577 reviews)	ratings only (2,565,349 ratings)

Image 1

The 5-core dataset here (in Image 1) indicates that reviews have been pre-filtered such that each of the users and products have at-least 5 reviews each. The size of electronics review data and corresponding metadata was 3GB and 2GB respectively.

Our initial inclination was to load only images uploaded by users and train our model on these images. But these images were found to be too unreliable and random (detailed later in Section 3.1) and so, it was decided that a mix of metadata images (product images from the platform) and user images would be used to train the model.

2.2 Dataset Fields

Field Name	Description	Data Type
asin	ID of the product	Int
review_images	URL of user images	List
clean_image	URL of product images	List
category	List of categories the product belongs to	List

Table 1

The above table shows the final merged dataset extracted after downloading the 5-core reviews (Image 1) and metadata of the products (not shown here), There were 112,066 unique rows in the dataset with ~40,000 unique products and corresponding category tag lists. It is pertinent to here that the fields of image URL's and categories are in-fact entire lists of data and not just singular values.

3. Methodology

3.1 Data Wrangling

The process of data wrangling is split into 4 main sub-tasks as shown below (Fig 2). Each of these sub-tasks had their own challenges which have been detailed further along in this section:

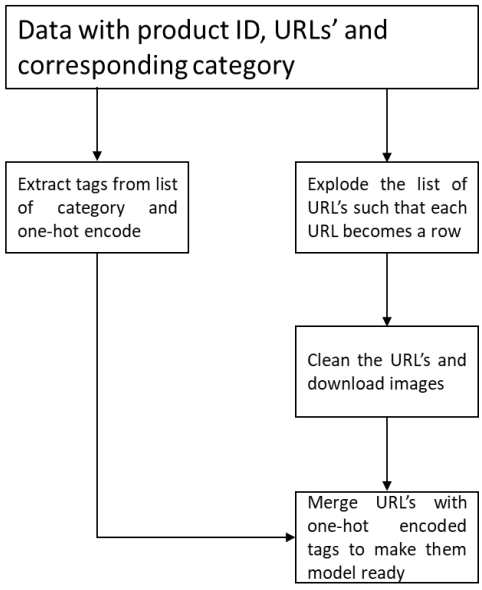

Fig 2

Extract tags & one-hot encode: We used a count-vectoriser with a custom tokeniser to create a data-frame of one-hot encoded tags from the list of categories available to us. This is possible because each image will have only 1 occurrence of the tag, therefore the count for any tag in an image would be either 0 or 1. 2175 distinct tags were found in the dataset (both user-generated and Amazon tagged), out of which 2 different tag datasets were created:

1. The 1st dataset had tags which were appearing at-least 0.5% of the times of the total corpus of tags. This generated 115 tags for 40,000 products. We soon realized though that these tags were too niche for our images to learn from.
2. The 2nd tag dataset was generated from tags that appeared at-least 5% of the time in corpus of tags. This generated around 14 tags.

The following cleaning steps were also necessary while generating the tags:

- Cleaning of the dataset field 'category' was required. Since it was being read as a text column, special characters (e.g. []) needed to be removed.
- Certain tags also appeared twice for a product (e.g. 'accessories', 'computer peripherals', 'keyboard', 'keyboard'). Such instances were reduced to just flags.
- The tag names as 'electronics' was present in every product – this tag was removed before generating the 2 tag datasets

Below is the tag count of some commonly occurring tags:

```
[('electronics', 112066),
 ('computers & accessories', 44377),
 ('camera & photo', 29599),
 ('accessories & supplies', 14063),
 ('computer accessories & peripherals', 12574),
 ('audio & video accessories', 10539),
 ('computer components', 8256),
 ('bags', 7547),
 ('cases & sleeves', 7536),
 ('accessories', 7200)]
```

Image 2

URL list explosion: Similar to 'category' field, the image URL fields for both metadata and user review data were lists. These fields were exploded to match such that each URL became a row matching the product ID. Since we were unaware of the count of URL's in metadata and user-review data, the explosion process for each was done separately. This generated 204,402 URL's for the metadata and 268,768 URL's for user-review data.

Clean URLs' and download images: The URL's had to be cleaned such that only .jpg images are present (instances of .gif were found in user-review dataset). Since images were scraped, certain video thumbnails were also captured as images (Image 3). These were removed as well.

Image 3: Video thumbnail jpg

Because of hardware and time constraints, we were unable to download all 450,000+ images. So, random sampling was used to pick images from both the metadata and user-review data.

We also address the necessity of taking both user-review images and metadata images. The biggest problem with user-review images was their randomness (especially when images for products such as a camera or camera-phone are concerned). Below are 4 random images from user-review data obtained for 1 single product:

Tags: ['Electronics', 'Camera & Photo', 'Digital Cameras', 'Mirrorless Cameras']

.*Image 4*: User-generated images for a camera device (exact product unknown)

From the set of images above, we can see that the user-generated images doesn't have a single image of the camera/camera-peripheral. And so, we decided to proceed by incorporating metadata review images into our data-set as well. The ratio of metadata images in this training set was always kept higher, so that the model could learn better from actual product images.

There were 2 versions of final data-set created:

- ~110,000 URL's by taking 2 randomly sampled metadata images and 1 randomly sampled review image. Some products were ignored as they did not have 2 metadata images.
- ~170,000 URL's by taking 5 randomly sampled metadata images and 2 randomly sampled review images. Some products were ignored as they did not have 2 metadata images. There were ~25,000 products taken into consideration. **We also note here that the images from this dataset was maxing out our available memory (13GB RAM on Kaggle) and hence we were unable to run any bigger dataset.**

Below is the training set for 1 product from 2nd dataset:

Tags: [Electronics, Computers & Accessories, Laptop Accessories, Bags, Cases & Sleeves, Messenger Bags]

Image 5: Training dataset for 1 product

The first two images in the image set above are obtained from user-review data. The latter five images come from the metadata of this product. It is also crucial to notice the difference in the sizes of the images from different sources (88x150 vs. 35x35). This size discrepancy would be something that is expected in an actual C2C web-portal as well and thus, we feel that this dataset would be a good dataset to train from, given the hardware limitations.

The next step was to download the images from the corresponding URL's. This was the most time-consuming component of our notebook, as well as the portion which required extensive preparation. The steps were as follows:

1. A function was generated which had a 2-fold task. Read the image from the URL (using urllib) and then convert it into a 3 channel array using the cv2 library
    a. Certain images were causing cv2 conversion errors, because of very small image size. Images with less than 1000 elements were ignored
2. We observed that image sizes varied a lot, and this variation was even higher for user-generated images. Hence, a decision to **resize** every image to a 35x35x3 array was taken. This is justified for the below reasons:
    a. The metadata images were much closer to a 35x35x3 array, and it was crucial that the model learns well from these images. We do not anticipate the loss from re-sizing the user-generated images to be substantial
    b. The precise size of 35 was chosen because some pre-trained CNN models have a minimum size requirement of 32 array size. So, reducing the size to 35 allowed us to use these models
    c. This array size allowed us to train on higher number of images given the hardware constraints
    d. While central cropping was another option available to us, the random size of images and the object of focus not being at the center were problems we were unable to tackle effectively
3. Another problem we faced was the time taken to download the images. Downloading the image one-by-one by applying the function onto the URL's was extremely slow, but computationally light. For reference, downloading images from 5000 URL's took approximately 7 minutes and was prone to frequent failures (due to URL errors such as *404: Image not found* or some other errors). To mitigate this, we decided to parallelize the image downloading process using *10 Pool* threads. This decreased the time to download 5000 images to < 30 seconds.
    a. For solving the errors related to URL or other connectivity issues, we implemented an exception handling component over the entire parallelizing and image manipulation function. So, the final dataset has lower number of images than the URL (60-70k images for the smaller dataset and 140-150k images for the bigger dataset)
    b. We also observed issues of CPU dropping to 0% usage and the process stopping while trying to parallelize

across the entire dataset of 100k + URL's. Our conjecture is that we were maxing out the available network given to us. This is because even though the loading of images isn't very computationally expensive, it is very network expensive. To mitigate this, loops of 10,000 URL's were run and these images were then appended. This way, we weren't maxing out the network capacity of the Kaggle Virtual Machine given to us.

Merge the tag dataset with one-hot encoded tags: As the tag creation happens on a product level, while image extraction on a product-URL level, we simply left-join the one-hot encoded tags onto the image dataset.

As a final step, all irrelevant columns are removed and a final check to ensure no missing values are present in the dataset is performed. These steps make the dataset ready for model building.

The final list of tags (for the 14 tags dataset) is in Image 6 below (loaded_img is the image array data):

```
array(['accessories', 'accessories & supplies',
       'audio & video accessories', 'bags', 'cables & interconnects',
       'camera & photo', 'cases & sleeves',
       'computer accessories & peripherals', 'computer components',
       'computers & accessories', 'headphones', 'laptop accessories',
       'portable audio & video', 'tablet accessories', 'loaded_img'],
      dtype=object)
```

Image 6: 14 tags + image ready for the model

3.2 Multi-label binary class problem

Every image can have multiple tags. So, we treat this problem as a supervised learning multi-label binary-class problem. Since we are working with one-hot encoded data, the last layer of our neural network has as many output nodes as the number of tags we are working with and a sigmoid activation to flatten the output between 0 and 1. We then use a pre-defined evaluation metric on our validation set to arrive at the best threshold and check it against the test data.

3.3 Dataset split

Our dataset is split into train-validation-test splits. The train & test split are in the proportion of (0.85:0.15), wherein the train data is further split into train and validation sets of the ratio (0.9:0.1).

3.4 Evaluation metric

The evaluation metric we use is something commonly used for a multi-label multi-class problem, which is the Hamming Score [6] (derived from the hamming loss). The formulae is given below:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

It tracks the number of correct 'True' label classifications and normalizes it against the union set of total number of "True predicted" and "True real". The number is then averaged over the total number of samples in the data (n).

Another metric we track, but do not make any decisions on is the Exact Match Ratio (Also known as subset accuracy or MR score). Unlike the Hamming score which tracks partially correct multi-label predictions, this metric considers a prediction to be correct only if all the labels match. This metric is a much more strict criteria and could be slightly problematic in cases with such as ours with high number of labels along with classes. The formulae for MR is:

$$\frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i)$$

3.5 Model Configuration(s)

With the ability to capture special and temporal dependencies, convolutional neural network or normally known as CNN is chosen for the image classification algorithm. In this paper, there are two main different CNN architectures being incorporated separately: vanilla CNN and VGG16 (we also tried a densenet architecture, but it was dropped very quickly on seeing the performance).

Vanilla CNN model: The built vanilla CNN model, whose architecture is inspired from the VGG model itself, is implemented as follows. First it takes an input of images with height x width pixels of 35x35 and input channel of 3 (RGB) which then being fed into two series of convolutional sets and fully connected layers with sigmoid output of 14 neurons (which represents 14 tags probability).

In the first convolutional set, we implemented Conv2D with 32 number of kernels with size of 3x3 each. Rectified Linear Unit (ReLU) activation function is used here with same padding. The output is then being fed to another layer of Conv2D with same configuration before it goes into a layer of max pooling of size 2x2 that is intended to aggregate information from each receptive field hence expected to be able to count for spatial invariance. To prevent overfitting, dropout is applied to the output with rate of 0.25 (probability of removing a

neuron of 25%). Dropout works as regularizer as in general, layer with lots of neurons develop co-dependency among each other hence curbing the individual power of each neuron.

The output of the first convolutional set then being channeled into the second convolutional set which has similar structure but with 64 number of kernels in the Conv2D layers instead of 32.

The resulting matrix is flattened and fed into 128 neurons of fully-connected layers with ReLU activation function and dropout with rate of 0.5 before gives the final 14 neurons sigmoid output. This architecture is displayed in image 7.

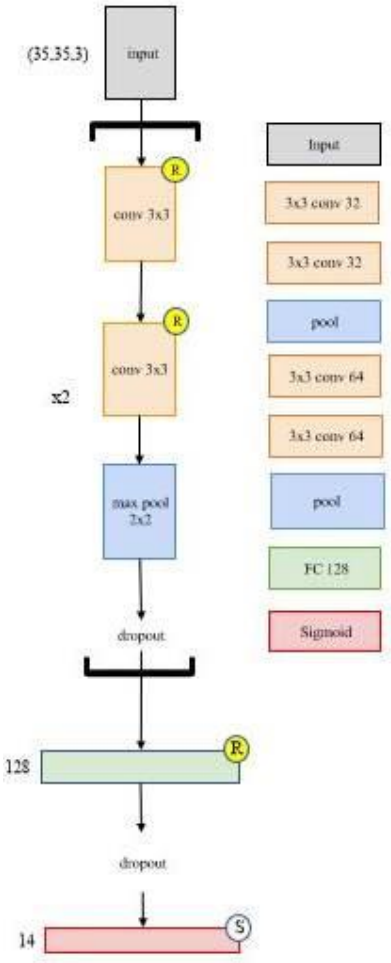

Image 7: Vanilla CNN model

VGG 16 model: The second model being incorporated as comparison is VGG16 which is a popular CNN architecture introduced in the year 2014 that was successful in achieving high accuracy of over 90% in the infamous ImageNet dataset. The detailed architecture is being pictured in image 8. It basically channels the input through five series of convolution sets with Conv2D with different numbers of kernels (64, 128, 256, 512, 512 at every convolution sets respectively) with size of 3x3 and 2x2 max pooling. To suit our application, we keep the weights pre-trained on ImageNet data until this part.

The output is then flattened and being fed into 128 ReLU fully connected layer with dropout of 25% rate before reaching the output sigmoid layer.

Both models employ Adam optimizer with learning rate of 0.0005 in which binary cross entropy is being used as loss function of the classification model. Fitting for both model are done in batch of 512.

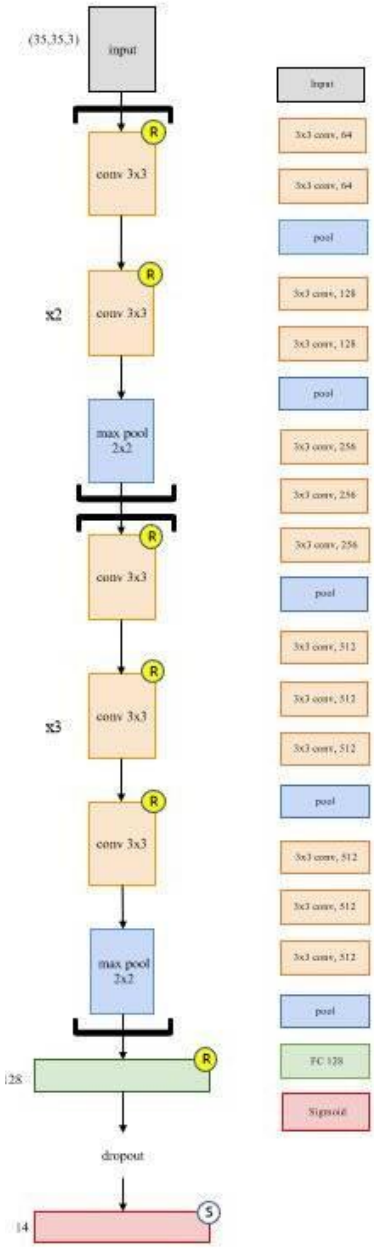

Image 8: VGG 16 model

Table 2: Comparison of results across models

MODEL	# TAGS	# TOTAL IMAGES	METADATA: USER IMAGE RATIO	THRESHOLD FROM HAMMING SCORE (Validation)	HAMMING SCORE (Test)	MR SCORE
Vanilla CNN	115	50-60k	2:1	0.15	0.1667	0.0
Vanilla CNN	14	50-60k	2:1	0.25	0.3628	0.2943
VGG16	14	50-60k	2:1	0.225	`0.2690	0.1964
DenseNet201	14	50-60k	2:1	0.05	0.2036	0.1059
Vanilla CNN	**14**	**150-160k**	**5:2**	**0.25**	**0.4528**	**0.3264**
VGG 16	14	150-160k	5:2	0.225	0.3555	0.1911

3.6 Recommendation System

We have built a very basic recommendation system to see if the tags generated by the images can actually be used to recommend other similar tags and thus help in enriching the searchable tags for the product and give it a targeted visibility.

Here, we have generated a cosine similarity matrix using a dense matrix for the set of 115 tags (same as used in model), this keeps a track of how similar the tags are to each other.

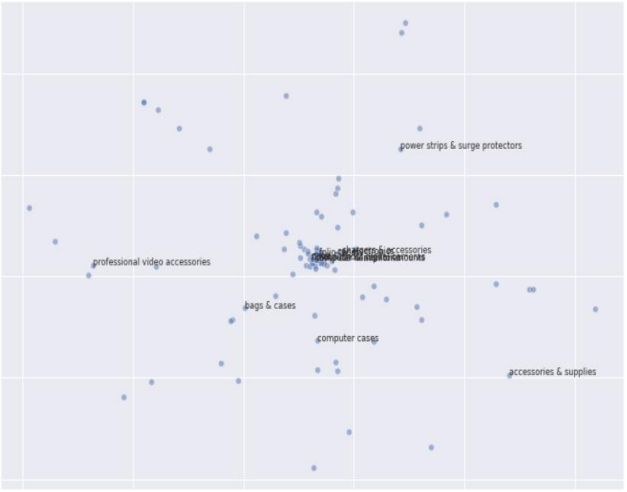

Image 9: Cosine Similarity of Tags

We also note here the tags above are **only for Electronics category.** Our hypothesis is that adding tags from more category of products to the matrix would generate distinct clusters which are far away from each other based on their main category with smaller sub-category of tags within them.

With the cosine similarity matrix ready, when an image comes through with a certain list of tags, the following steps should happen in order:

1. We look at the other tags each individual tag is similar to and remove the ones with 0 similarity.
2. The cumulative cosine similarity is calculated between all the tags.
3. A cutoff of 0.2 is decided and tags are displayed in descending order of similarity.
4. These tags along with the existing tags can be used for searchable tags to give the product more visibility.

```
cases                          0.458338
hubs                           0.395285
vehicle electronics accessories 0.373609
filters & accessories          0.355707
on-dash cameras                0.330404
hdmi cables                    0.312916
memory card accessories        0.295804
over-ear headphones            0.283088
```

Image 10: List of similar tags for an office electronic accessory containing tags [‘electronics & accessories’, ‘office accessories’, ‘labelling tapes’]

4. Results and Discussion

Table 2 is used as reference for all our discussions in this section.

4.1 Findings and Insights

- The Vanilla CNN model with an input ratio of Metadata: User Image of 5:2 is giving us the best hamming score of 0.4582 and with the corresponding exact match percentage of 32.64%. We believe that the performance here is quite good as for 32.64% of samples, all the labels are being correctly identified.
- A surprise to us was the performance of pre-trained models. Our hypothesis is that the densenet model (pre-trained on ‘ImageNet’ with frozen layers) could be performing the worst because our input image is simply too small for a model with so many layers. We also hypothesize that VGG16 model is performing worse because the images it is being trained on are simply too different compared to ImageNet images! We believe that more samples of images along with allowing the model to train on some bottom CNN layers of the architecture will give better

results (Currently, all weights apart from the last output layer is frozen).

- Looking at the count of number of tags predicted across samples, we find that our models are on the conservative side when compared to the truth. This might be an indication that our model has a high accuracy in terms of what it predicts (the hamming score could be less because there are more truth labels that our model did not catch) and we need more samples for higher accuracies.

4.2 Challenges to this model

1. The biggest challenges we faced while training the models were due to the user generated images. There were instances of random images not related to the tags being uploaded (shown in section 3.1) and the tags were also a mix of system generated and user generated. So, there was a very high probability that images would get associated to random tags not related to the product at all.
2. Another potential problem we foresee is identifying the correct number of tags to train the model on. As observed in (1), there are many user generated tags in the dataset. When running the model across categories, taking the correct number of tags to train on across categories along with sampling the dataset so that all categories are learnt will be another added exercise.

5. Summary

5.1 How will the system work?

Overall, once the models and the cleaning processes been perfected. The entire flow after a person uploading an image to tags generation for better recommendations and search categorization can be automated to give the new users an easy time and fair playing ground against the older users who know how to abuse the system.

Since the entire process of tags generation is automated, this makes the user experience better with the users on a much better playing field based on the quality of their products.

Figure 3 shows the flow we envision in a concise manner.

5.2 Potential for future exploration

An interesting avenue to explore would be how tags are generated. We observed some hierarchy in the tag list which can be taken advantage of. E.g. a hierarchy observed was:

Electronics -> Accessories -> Camera Accessories

It could be possible to take advantage of this system of hierarchies by combining a CNN model with a variant of an RNN model. This becomes particularly useful when we wish to predict 3-4 tags for each main category (it could mean predicting across 72-96 labels if using one-hot encoded). This variant of CNN + RNN, if trained well, would have a high accuracy given it is able to predict the main category tag well.

The drawback is, obviously, that such a model would rely heavily in predicting the tag for main category of product correctly and if that does not happen, then the entire sequence of tags could go wrong!

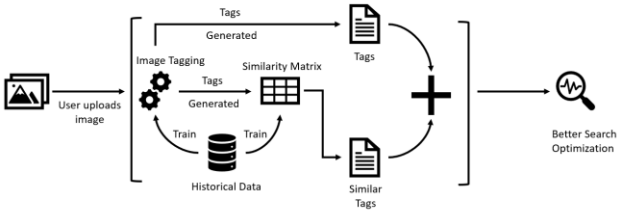

Fig 3: End-to-end process of image tag generation and usage

6. Github folder link

The link for the github folder containing our dataset and notebooks is: https://github.com/nusashish23/amlproject

7. References

The following references were used in preparing this report:

1. Khusro, Shah & Ali, Zafar & Ullah, Irfan. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. 10.1007/978-981-10-0557-2_112.
2. Mohamed, Marwa & Khafagy, Mohamed & Ibrahim, Mohamed. (2019). Recommender Systems Challenges and Solutions Survey. 10.1109/ITCE.2019.8646645.
3. Kurt, Zuhal & Ozkan, Kemal. (2017). An image-based recommender system based on feature extraction techniques. 769-774. 10.1109/UBMK.2017.8093527.
4. Jocic, Obradovic, Malbasa, Konjovic (2017). Image tagging with an ensemble of deep convolutional neural networks, 7th International Conference on Information Society and Technology ICIST 2017
5. Amazon Review Dataset: https://nijianmo.github.io/amazon/index.html
6. Mohammad S Sorower. A Literature Survey on algorithms for Multi-label Learning. Oregon State University