Image style embedding based
Fashion Recommendation System

Mohammed Almanassra
malmanassra3@gatech.edu

Bowen Chen
chen.bowen@gatech.edu

Erima Goyal
egoyal3@gatech.edu

Oscar Parrilla
oparrilla3@gatech.edu

GitHub Repo

https://github.com/chen-bowen/complete-the-look-recommendation-system

Abstract—The last two years of COVID, repeated lockdowns, and plain fear of stepping out of the house have fast-paced the adoption of eCommerce by many decades. Total online spending in May 2020 hit \$82.5 billion (about \$250 per person in the US), up 77% year-over-year [1]. Even though it is good news for retailers as they do not need to invest in numerous brick and mortar stores, they face the conundrum of serving an invisible customer. They cannot see the customer, do not have any insight into customer preferences and might have a limited transaction history for the customers as well as the products sales. Walking into a store, salespeople see the customer and can recommend clothes based on the customer's looks. Our motivation is to create a cold start recommendation engine that does not need any knowledge of user preferences, user history, item propensity or any other data to recommend products to the customer. In this paper we are trying to solve two problems, given a product what are the similar products and what are the complementary products that will complete the outfit. We drew inspiration from some previous work on Deep Fashion [2,3] and Complete the Look [4]. In this paper we will be discussing our approach, experimental set-up, and the results.

1 INTRODUCTION

Fashion retailers try to implicitly help customers and explicitly help themselves by creating advanced recommendation systems, which are based on large transactions and search data history. Many state-of-the-art algorithms exist that are based on user similarities - recommend products purchased by one user to other similar users [5]. As the data on users and their transaction history started becoming scarce, Amazon coined Item-to-Item collaborative filtering approach [6] that uses items similarity to recommend the products. However, the latter is also based on the premise of having a large transaction history of customers and the items purchased, but does not need a transaction history of the customer himself. Creating an item similarity vector can also be addressed by manual or machine-based image annotations, where tags for product type, color and style are created. Large scale work is done on annotation techniques such as FashionNet [2] and ModaNet [7] to create image data with massive attributes and clothing landmarks.

Recently, convolutional neural network (CNN) based fashion recommendation techniques, which automatically recommend matching clothes to the consumer, have been widely researched. Shin et al [3] have proposed a Style feature decomposition approach to extract image style vector from the query image. Viet et al. [8] employed the Siamese CNN to predict whether two clothing items from various categories fit together. However, the clothes vector obtained through this method is not effective in predicting compatibility between items from various categories since it contains mixed information from the category and style.

In this paper, the aim is to enable the fashion retailers to create a recommendation system without the need for any customer history, customer preference, product sale history, product annotations or any other such large-scale meta data. A two-layered solution has been devised.

(1) Recommend similar products to the customer i.e., similar style and same product category. For this, we are discussing a state-of-the art style embedding based approach that does not need any meta data (product category, color, style etc.) on the images, but only the image itself.

(2) Recommend complementary products i.e., a similar style product from another product category. For example, recommending a pair of pants that is "style similar" to a shirt in consideration. For this, we are leveraging the existing complete the look implementation and use the triplet loss approach to identify compatible images in other product categories.

If this work hits mainstream, the vast number of fashion retailers will not need customer and item transaction history to make recommendations. Also, since this approach does not store customer data two more problems are addressed:

(1) The vast amount of storage space needed to capture and store historical transaction data is not needed.

(2) With the worldwide push for ethics and fairness algorithms, this approach does not contain bias towards a particular customer's non-fashion related preferences and background.

2 APPROACH

2.1 Recommend Similar Products

2.1.1 *Problem statement*

Given a query image, the model will return products similar to the query image and in the same product category. For example, if the customer clicks on a shoe, the model returns shoes that are similar in style and appearance to the original shoe.

Product Type	# Of Images
Shoes	11,780
Shirts & Tops	7,506
Pants	6,553
Wallets & Cases	3,255
Coats & Jackets	2,485
Sunglasses	1,772
Shorts	1,533
Skirts	1,392
Earrings	926
Necklaces	909
Total	**38,111**

Table 1 — Unique list of products in the STL data

2.1.2 *Dataset*

For this approach, the modified STL data from Pinterest was used which has been the inspiration of Complete the Look: Scene-based Complementary Product Recommendation [9] work. The dataset has scene-product pairs, bounding boxes for products, and product category information, all of which are labeled by internal workers. The product data was created by cropping products from various scenes which results in 38,111 unique products across 10 product categories as listed in Table 1.

2.1.3 *Data pre-processing*

Images were resized to 256X256, converted to tensors, and normalized for the ResNet model and then the data was passed to a PyTorch data loader which would be used in the feature extraction.

2.1.4 *Similar Product Modeling*

2.1.4.1. *Image embeddings*

To extract image embeddings, the batched data was passed through ResNet18 and features were extracted from the last average pooling layer for each image in the batch. Embeddings from each batch were appended together to a final aggregated feature layer which was stored in a style embedding catalogue through a pickle file.

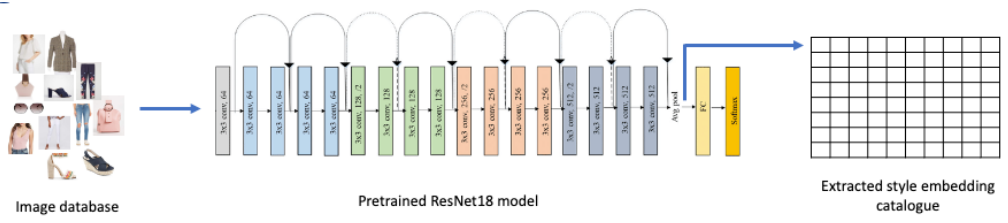

Figure 1 — Embedding extraction.

2.1.4.2. *Similarity and Recommendation*

To retrieve similar products, the cosine distance function was used as shown in figure 2. Cosine function is simple to interpret and similar images have higher cosine similarity value. Given the query product's feature vector, the pairwise cosine similarity between the query feature vector to all the embeddings in the style embedding catalogue is calculated and ranked in descending order. The top 5 similarity score will be the similar product for the query product, which will be returned as recommendations (shown in figure 3).

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 2 — Cosine Similarity

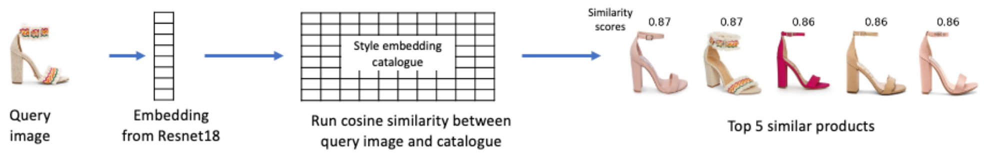

Figure 3 — Extracting images similar to the query image.

2.2 Recommend Complementary Products

2.2.1 *Problem statement*

Given a query product image, the model will predict the products from various categories that are compatible with original query product. For example, if the query product is a leather pair of pants, then what kind of blouse or shoes will go with it? In an ideal world, there are many solutions to this problem due to the underlying subjectivity and individual preferences to pair various products. Hence, it was decided to train a compatibility model to learn relationships between articles of clothing based on the Polyvore dataset created by Eileen et al [4].

2.2.2 *Dataset and Preprocessing*

The original data contains 1,006,519 outfits and 4,246,430 fashion objects. Only around 1/10 of the data was made available for public use. As a result, training was limited to 107,890 outfits which translates to 479,483 single products. As for testing data, 5029 outfits were held out, which is roughly 25,140 individual fashion items. The female-to-male clothing ratio is about 10:1 in the data. Table 2 shows all different product types.

Product Category	**# Of Images**
Shoes	98,680
Handbags	93,201
Shirts & Tops	61,235
Pants	54,697
Coats & Jackets	45,891
Dresses	28,829
Jewelry	26,337
Hats	19,663
Skirts	15,950
Sunglasses	10,179
Shorts	9,849
Scarves & Shawls	6,221
Watches	6,094
Belts	724
Socks	716
Rings	554
Jumpsuits & Rompers	321
Swimwear	162
Stockings	113
Gloves & Mittens	58
Neckties	9
Grand Total	**479,483**

Table 2 — Unique list of products in the CTL data

With the bounding boxes available in the data, single images were extracted for different polyvores. An image triplet dataset was created using the sampling methodology in [10]. Each triplet has (1) an anchor image, (2) a positive image sampled from different category of same Polyvore, and (3) a negative image sampled randomly from another image but has the same category with the positive image. For example, in figure 5, a selected shoe from one of the base images is set as the anchor, a jacket from the same base image as positive and another jacket from a different base image is set as negative – forming a single triplet. The triplet data was then used to train the compatibility model.

2.2.3 *Compatible Product Modeling*

The compatible product model is based on ResNet18 with two additional FC-layers on top with a batch normalization layer in between and a dropout layer, which outputs a set of 128-dimensional style embeddings for all products. Experiments showed that batch normalization was essential to the training process as it prevented the model from getting trapped into a trivial solution - making the embedding of anchor, positive and negative embeddings identical, resulting an extremely low loss. Making all embeddings identical would result in the pairwise compatibility score of 1 for all product pairs, causing failures in recommending any meaningful compatible products.

With the compatible model trained, all products were fed through the trained model to obtain the style embedding catalog, in which a pairwise compatibility score could be calculated, and recommendation could be performed.

The general process of the compatible modeling is shown below in figure 4, figure 5 and figure 6.

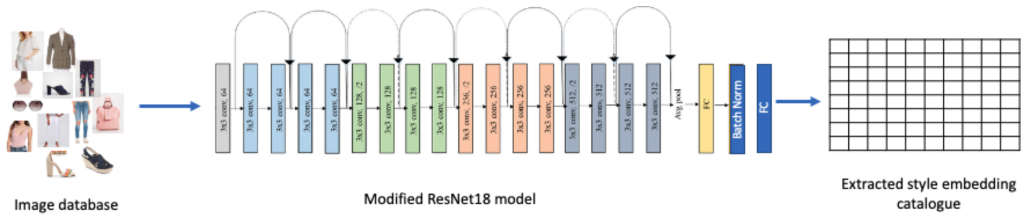

Figure 4— Flow of data as seen through the model

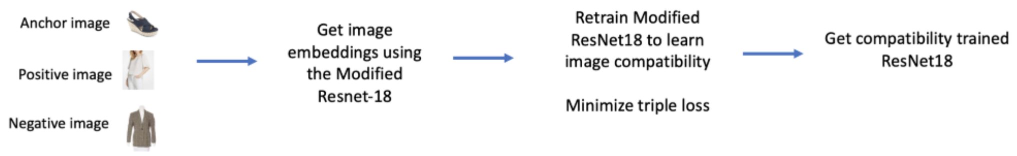
Anchor image
Positive image
Negative image
Get image embeddings using the Modified Resnet-18
Retrain Modified ResNet18 to learn image compatibility
Minimize triple loss
Get compatibility trained ResNet18

Figure 5— Model training to learn compatibility of images

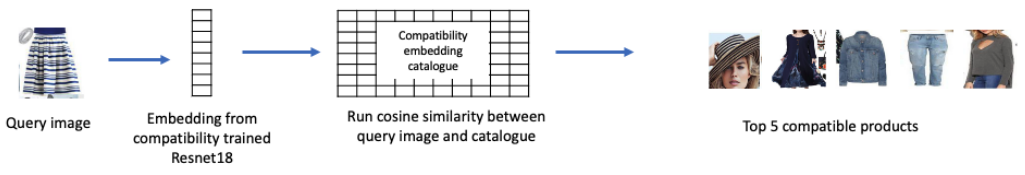
Query image
Embedding from compatibility trained Resnet18
Compatibility embedding catalogue
Run cosine similarity between query image and catalogue
Top 5 compatible products

***Figure 6*—** Extracting images compatible to the query image

2.2.4 *Training And Hyperparameters Tuning*

The model was trained with Triplet Loss, which is described as the following,

$$\mathcal{L}(A, P, N) = \max\left(\| \mathrm{f}(A) - \mathrm{f}(P)\|^2 - \| \mathrm{f}(A) - \mathrm{f}(N)\|^2 + \alpha, 0\right)$$

where A is embedding for anchor image, P is the embedding for positive image and N is the embedding for negative image. The goal is that the distance from the anchor to the positive is minimized, and the distance from the anchor to the negative input is maximized.

Out total 77,494 triplets in training data, 90% of triplets were used as training samples and 10% were held out for validation during training. During training, each member of the triplets was passed through the same network to obtain its feature embeddings, then the loss was calculated according to the above triplet loss formula. A final training loss of around 1.001 was achieved by training the model with 1 epoch, and final validation loss was around 0.9824. Due to compute limitations on the GPU, as each epoch took 7 hours to complete, we were able to train to only 1 epoch. The loss curves per 100 triplets from the training is shown in figure 7.

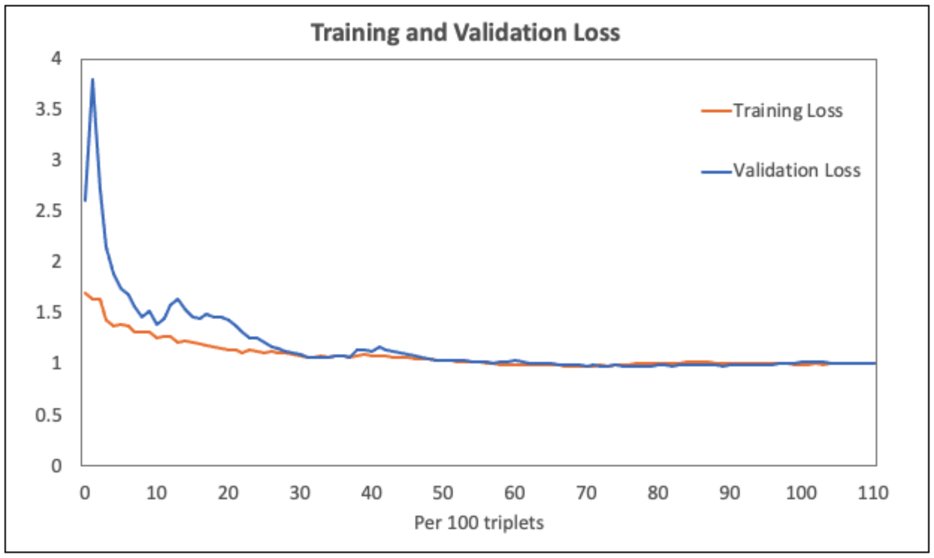

***Figure 7*—** Loss curve for training the compatible model

During the experiments, the best results were achieved with the learning rate of 0.005. A larger learning rate led to divergence of the loss, while smaller learning resulted in slow convergence which might require additional time to train.

The batch size was set to 64 to mitigate the bottleneck on the data loader – it was discovered that reading in the images in larger batches speed up the training process running on the GPU but would cause suboptimal performance in learning the style embedding representation.

The margin of the triplet loss was chosen to be 0.2 for it to converge faster. Larger values of margin were resulting in unstable loss.

2.2.5 *Evaluation*

The test data, consisting of 5029 outfits as base images, resulting in 25,141 sole product items was set aside for final evaluation. Following the pipeline described in section 2.2.3, the style embedding catalog was generated. For each outfit base image, a cropped image was picked from an outfit (anchor) and another cropped product was picked from the same image (positive). This pairing was compared to a pairing between anchor and another cropped product from a different outfit but the of the same category of the positive product image (negative). When the pairwise similarity score between the positive and anchor is bigger than the similarity score between negative and anchor, the model is deemed to be correct in that record. Specifically, out of 5029 outfits, correct records were in the range of 2800s (depending on the sampling seeds), making it ~%60 correct rate. This comparison is displayed in figure 8 and figure 9.

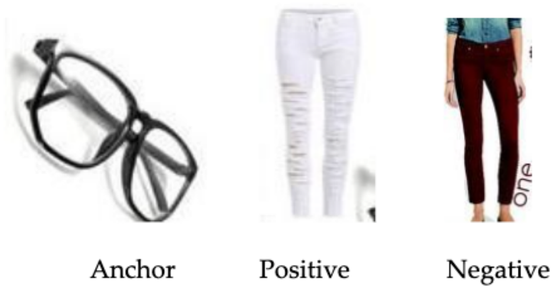

Figure 8— This triplet shows a correct identification in compatibility. The anchor-pos similarity score was .9972 meanwhile the anchor-neg similarity score was .8934.

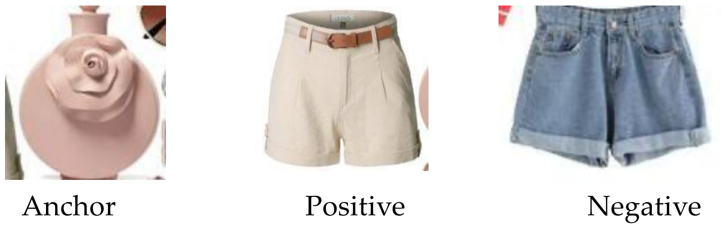

***Figure 9*—** In this triplet example the model determined an incorrect compatibility. It identified the anchor-pos score as .8735 and the anchor-neg score as .9892.

2.3 Challenges

There were several challenges throughout the project timeline.

1. *Datal Loader's Default Shuffling Behavior* – The default behavior for PyTorch data loader was randomly shuffling the data for training, which caused the mismatch between the image's feature vector and its metadata, resulting in the recommender returning erroneous image metadata even though the feature vectors were extracted correctly. The solution was to set shuffling to True when training the compatible model and turn shuffling off when extracting the style embedding catalog.
2. *Data Loader's Bottleneck* – Even with GPU, loading a triplet of images has significantly bottlenecked the training speed - with 7 hours for 1 epoch was the fastest our team was able to achieve. It was particularly hard to evaluate the compatible model's performance or tune the hyperparameter with such enormous training time. There were very few resources available to solve the problem, making it hard to isolate a great set of hyperparameters that would lead to better performances.
3. *Compute Power* – Deep learning on images is very compute intensive. For this work, local compute resources on individual laptops were not sufficient and hence, Google Cloud Platform (GCP) was used for various compute intensive tasks such as feature extraction and training. Even with GCP, to limit the usage cost, only 2 epochs could be run at a time. This severely limits the model's ability to train longer for more accurate results.

3 EXPERIMENTAL SET-UP AND RESULTS

3.1 Recommend similar images results

The goal was to find products similar to the query image for the same product category. For example, if the query image is a light color, knee length coat, the output should be products that are similar to it in style and the user might like. Some preliminary results of the "Recommend Similar Images" framework are shown in figure 10, and more extensive results are shown in appendix in figure 12, figure 13, and figure 14. Assessing the quality of recommendations is subjective to the person viewing it; however, the initial assessment of 100+ recommendations provide strong confidence in the accuracy of similar recommendations. As seen in figure 10, the 5 recommended coats are very similar to the original query image, yet not identical.

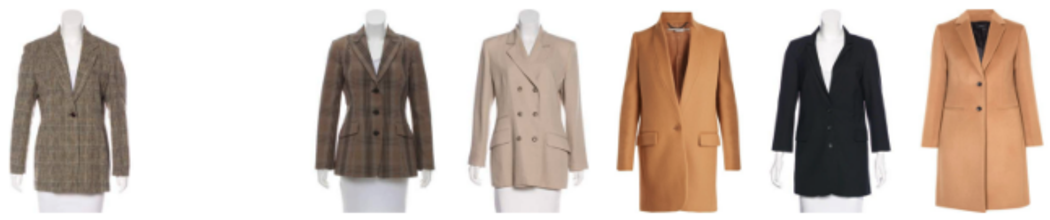

Figure 10—Image on the left are query image and images on the right are the ones returned by "Recommend Similar Images" framework

3.2 Recommend complementary images results

The goal was to retrieve products that are compatible in style to the query image, but not from the same product category. Some preliminary results can be seen in figure 11 and more detailed results are available in appendix – figure 15, figure 16 and figure 17. Similar to the previous approach, the evaluation of quality of recommendations is subject to human interpretation. Scanning through 100+ recommendations, one can confidently conclude that products recommended are compatible in style to the original query image. Observing the input query image of a black bag in figure 11, the recommended images are leather jacket, a pair of sneakers, a funky looking hoodie, a fur hat, and a watch that all seem similar to the original query image.

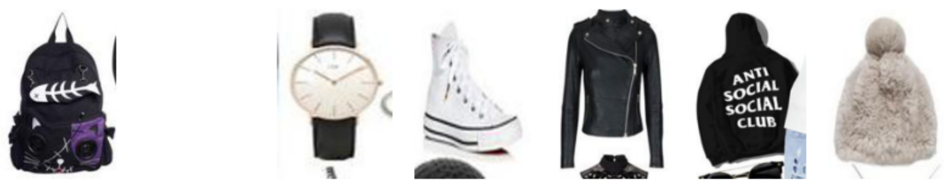

Figure 11— Image on the left are query image and images on the right are the ones returned by "Recommend Complementary Images" framework

3.3 Streamlit

To highlight the results in an interactive fashion, the "Recommend Similar Images" model was deployed using Streamlit.io, an open-source Python library used to build custom web apps for machine learning and data science projects. While the deployment was easy and quick with Streamlit, the application is limited by design and interactivity. Currently, the application is driven by a manual input of product ID to display similar images. Live Link: https://share.streamlit.io/markpython86/complete_the_look/main

4 FUTURE WORK

Recommend Similar Images and Recommend Complementary Images, both frameworks return exceptionally satisfactory results. From visual inspection, it is apparent that the experiment was successful.

As future work, firstly, some experiments can be done with the Triplet Loss function such as soft triplet loss [11] to further enhance the complementary recommendations. Secondly, an experiment with models other than ResNet-18 as its base model to extract the embeddings could prove useful in showing improvements. Thirdly, for a true commercial representation, a combination of the approach recommended in this paper layered with transaction data insights to generate truly applicable recommendations. Finally, in the future, the Streamlit application can be enhanced to where users can upload images of their own clothes and the application would recommend the complement style or similar style.

5 APPENDIX

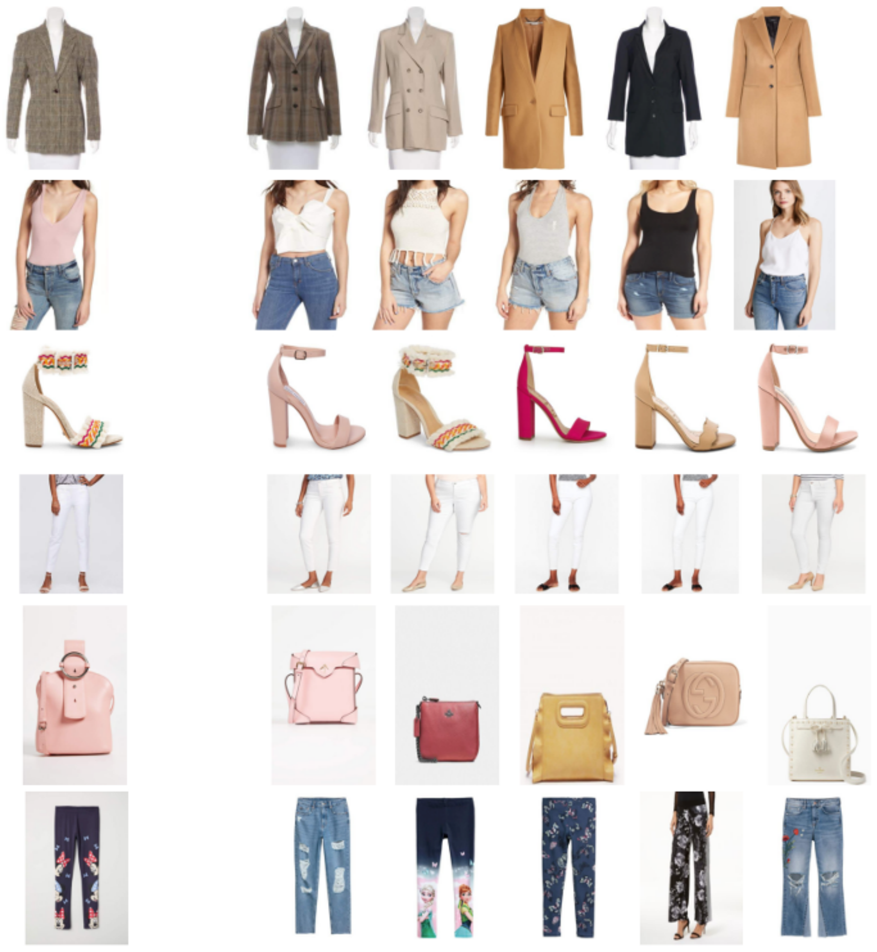

Figure 12—Images on the left are query image and images on the right are the ones returned by "Recommend Similar Images" framework

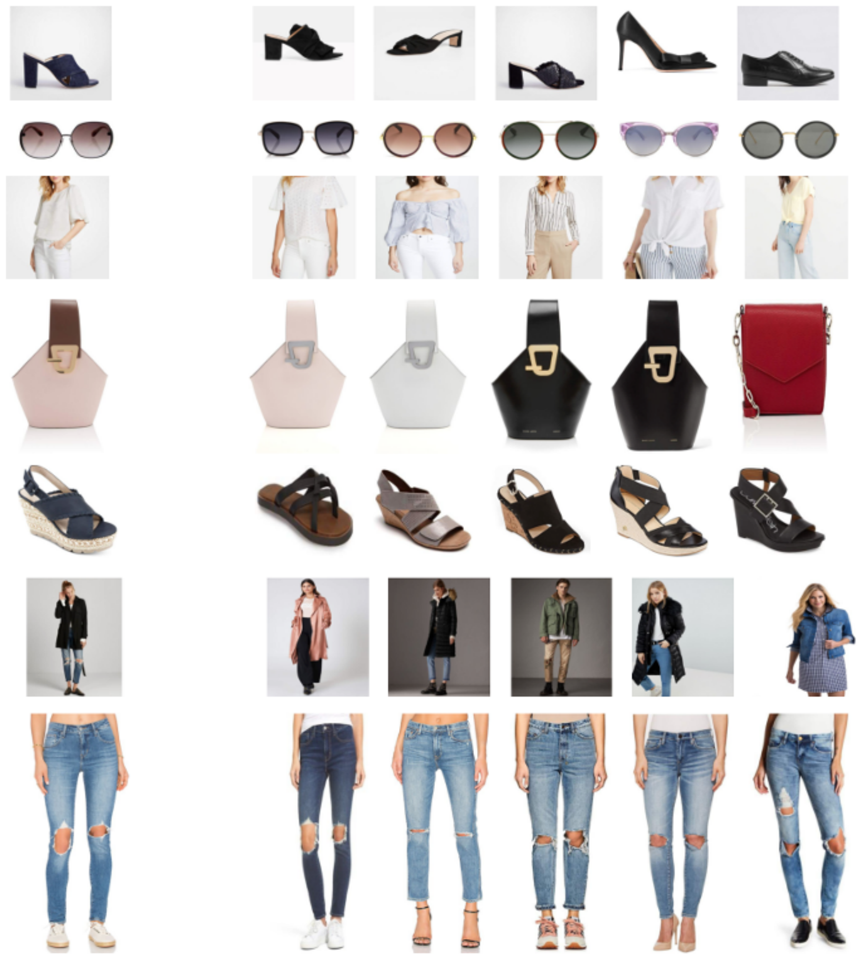

Figure 13—Images on the left are query image and images on the right are the ones returned by "Recommend Similar Images" framework

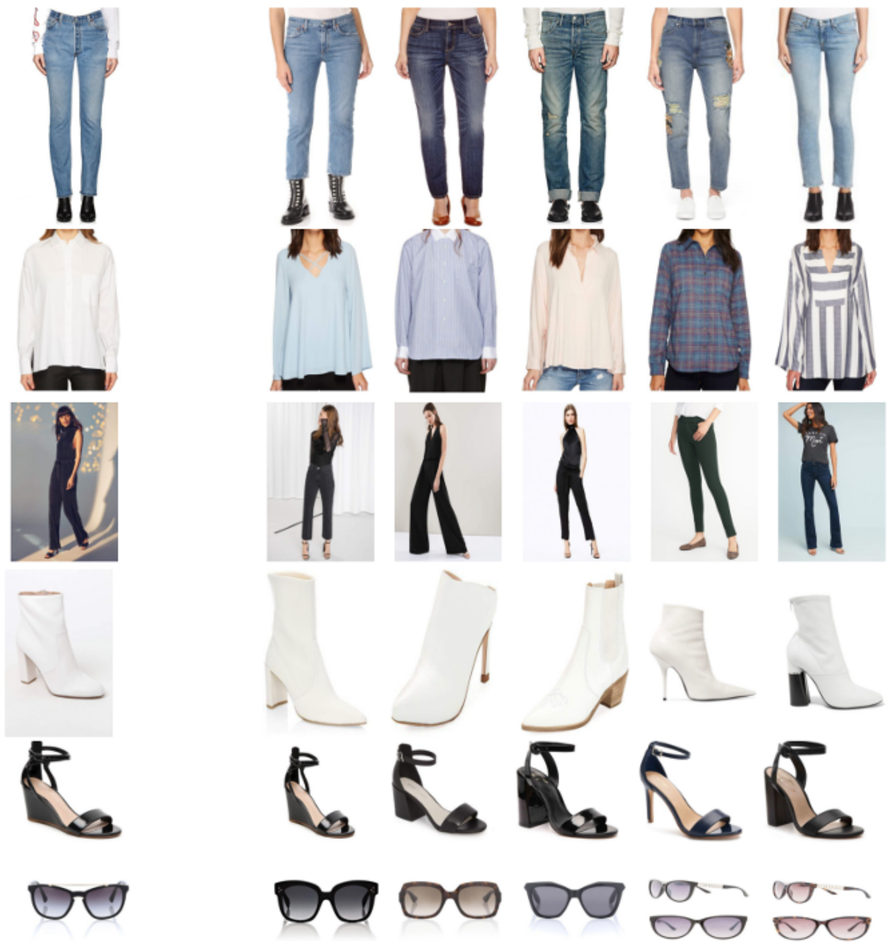

Figure 14—Images on the left are query image and images on the right are the ones returned by "Recommend Similar Images" framework

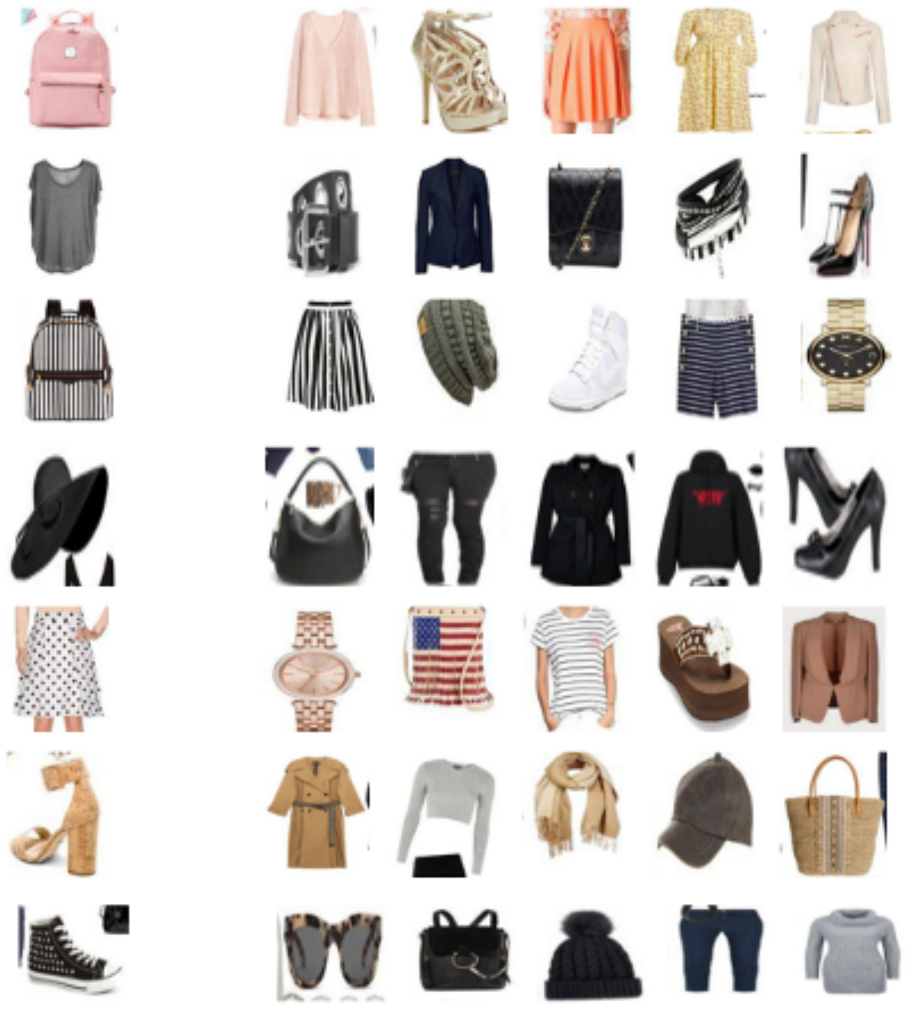

Figure 15—Images on the left are query image and images on the right are the ones returned by "Recommend Complementary Images" framework

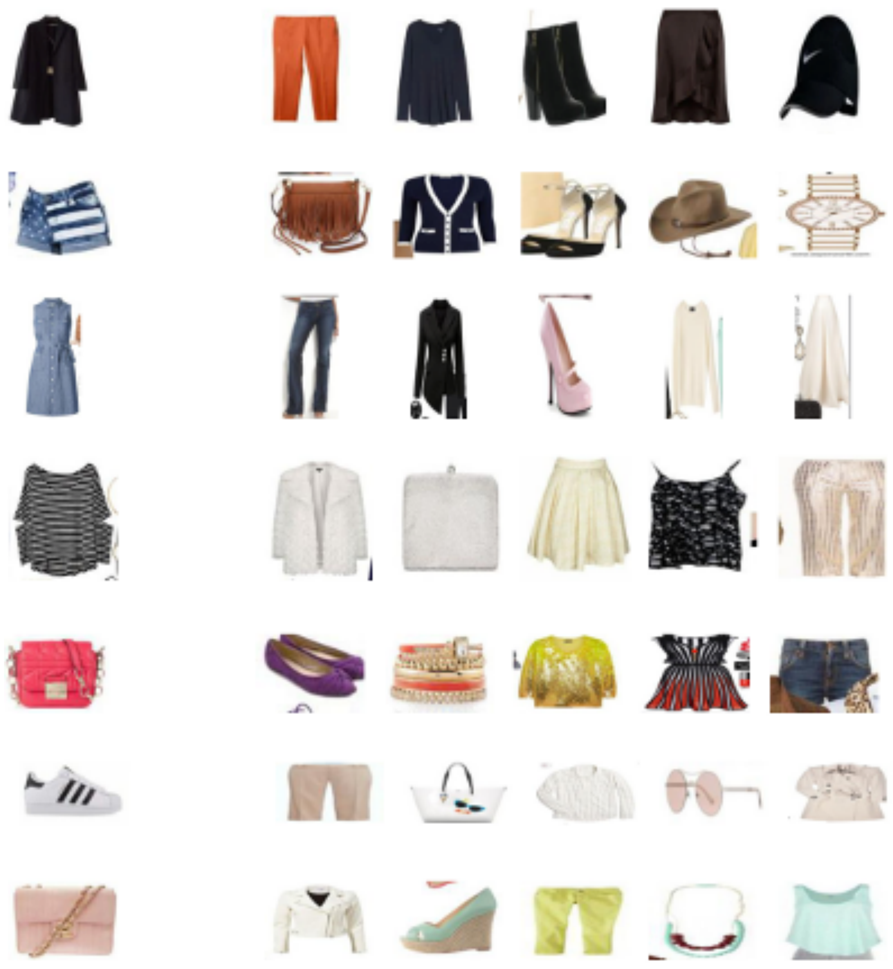

***Figure 16*—** Images on the left are query image and images on the right are the ones returned by "Recommend Complementary Images" framework

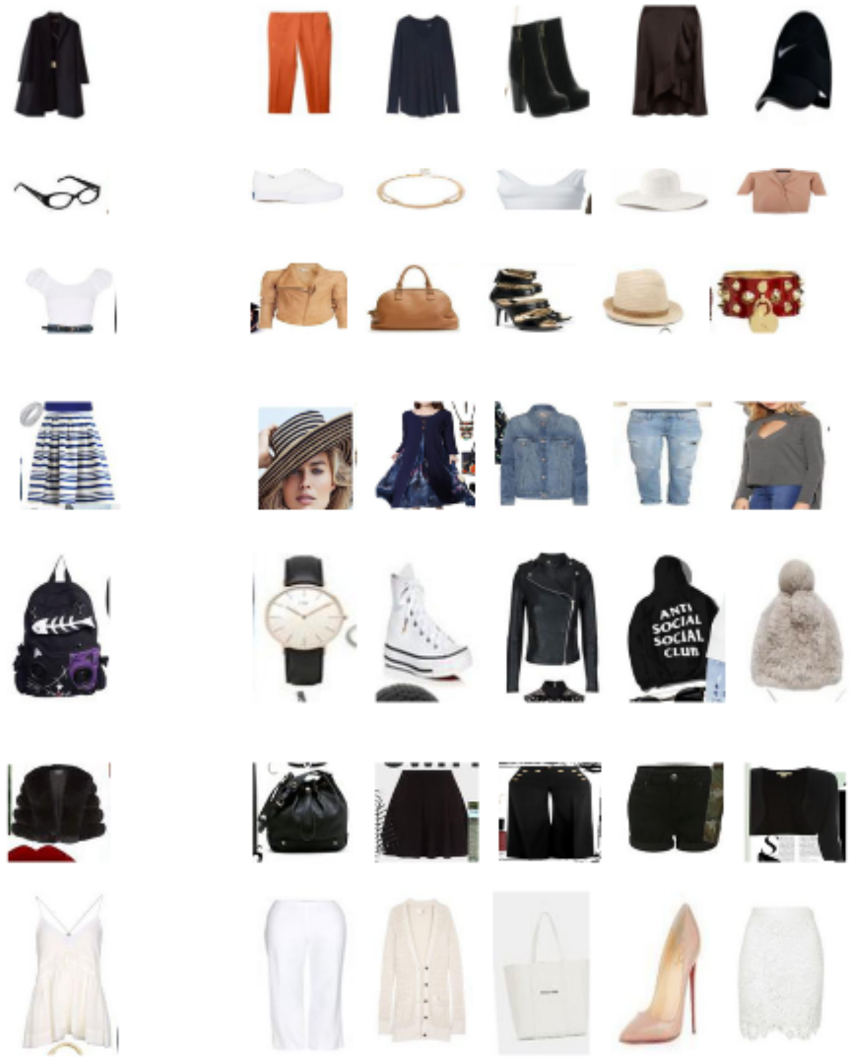

Figure 17—Images on the left are query image and images on the right are the ones returned by "Recommend Complementary Images" framework

6. REFERENCES

[1]https://www.forbes.com/sites/johnkoetsier/2020/06/12/covid-19-accelerated-e-commerce-growth-4-to-6-years/?sh=6a4adc1f600f

[2] Z. Liu, P. Luo, S. Qiu, X. Wang and X. Tang, "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1096-1104, doi: 10.1109/CVPR.2016.124.

[3] Y. Shin, Y. Yeo, M. Sagong, S. Ji and S. Ko, "Deep Fashion Recommendation System with Style Feature Decomposition," 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), 2019, pp. 301-305, doi: 10.1109/ICCE-Berlin47944.2019.8966228.

[4] Eileen Li, Eric Kim, Andrew Zhai, Josh Beal, Kunlong Gu, "Bootstrapping Complete the Look at Pinterest", arXiv:2006.10792, June 2020

[5] Mahamudul Hasan, Shibbir Ahmed, Md. Ariful Islam Malik, and Shabbir Ahmed, "A comprehensive approach towards user-based collaborative filtering recommender system," DOI:10.1109/IWCI.2016.7860358December 2016

[6] Greg Linden, Brent Smith, and Jeremy York, "Recommendations Item-to-Item Collaborative Filtering," Amazon.com Industry Report 2003

[7] Shuai Zheng, Fan Yang, M. Hadi Kiapour, Robinson Piramuthu, "ModaNet: A Large-Scale Street Fashion Dataset with Polygon Annotations," arXiv:1807.01394, July 2018

[8] Andreas Veit, Balazs Kovacs, Sean Bell, Julian McAuley, Kavita Bala, Serge Belongie, "Learning Visual Clothing Style with Heterogeneous Dyadic Co-occurrences," 1509.07473v1 [cs.CV] 24 Sep 2015

[9] Wang-Cheng Kang, Eric Kim, Jure Leskovec, Charles Rosenberg, Julian McAuley, "Complete the Look: Scene-based Complementary Product Recommendation," arXiv:1812.01748, April 2019

[10] Ruining He, Charles Packer, and Julian McAuley, "Learning Compatibility Across Categories for Heterogeneous Item Recommendation," arXiv:1603.09473, September 2016

[11] Qi Qian, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, Rong Jin, "SoftTriple Loss: Deep Metric Learning Without Triplet Sampling," arXiv:1909.05235v2 [cs.CV] 15 Apr 2020

[12] Tong He, Yang Hu, "FashionNet: Personalized Outfit Recommendation with Deep Neural Network," arXiv:1810.02443, October 2018

[13] Chakraborty, S.; Hoque, M.S.; Rahman Jeem, N.; Biswas, M.C.; Bardhan, D.; Lobaton, E., "Fashion Recommendation Systems, Models and Methods: A Review," Informatics 2021, 8, 49. https://doi.org/10.3390/informatics8030049

[14] Leon Gatys, Alexander S Ecker, and Matthias Bethge, "Texture synthesis using convolutional neural networks," In Advances in neural information processing systems, pages 262–270, 2015.

7 TEAM CONTRIBUTION

Name	Contribution
Mohammed Almanassra	• Embedding extraction for Recommend similar style • Loss and evaluation • Deployment via Streamlit • GPU set-up
Bowen Chen	• Data Downloading • Embedding extraction for Recommend complementary style • Model building • Training • Recommendation payload function • Evaluation
Erima Goyal	• Embedding extraction for Recommend similar style • Image Utils • PowerPoint • Similarity Utils
Oscar Parilla	• Embedding extraction for Recommend complementary style • Data Loading and Dataset • GPU set-up • Evaluation